

Tracklytics: een DevOps library voor het monitoren van operationele mobiele applicaties

Stef Van Gils

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
computerwetenschappen,
hoofdspecialisatie Software
engineering

Promotor:
Prof. W. Joosen

Assessor:
Dimitri Van Landuyt

Begeleider:
Dimitri Van Landuyt

© Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor als de auteur is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wend u tot het Departement Computerwetenschappen, Celestijnenlaan 200A bus 2402, B-3001 Heverlee, +32-16-327700 of via e-mail info@cs.kuleuven.be.

Voorafgaande schriftelijke toestemming van de promotor is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Voorwoord

Dit is mijn dankwoord om iedereen te danken die mij bezig gehouden heeft. Hierbij dank ik mijn promotor, mijn begeleider en de voltallige jury. Ook mijn familie heeft mij erg gesteund natuurlijk.

Stef Van Gils

Inhoudsopgave

Voorwoord	i
Samenvatting	iv
Lijst van figuren en tabellen	v
1 Inleiding	1
2 Context	3
2.1 Relevantie monitoren mobiele applicaties	5
2.2 Development Scenarios	9
3 Doelstelling	11
3.1 Library	11
3.2 Data Visualisatie	14
4 Architectuur	15
4.1 Component Diagram	15
4.2 Deployment Diagram	19
4.3 Belangrijkste flows	19
4.4 Uitbreidingen	21
5 Implementatie	25
5.1 Details Implementatie	25
5.2 Aggregatie	36
5.3 Openstaande uitdagingen	39
6 Tutorial	43
6.1 Installatie	43
6.2 Initialisatie	43
6.3 Counter	44
6.4 Gauge	44
6.5 Histogram	45
6.6 Timer	45
6.7 Meter	46
7 Evaluatie	47
7.1 Performance	47
7.2 Schaalbaarheid	49
7.3 Developer effort	50

7.4 Conclusie en bespreking resultaten	52
8 Besluit	69
A IEEE Artikel	75
B Poster Tracklytics	81
Bibliografie	83

Samenvatting

In dit **abstract** environment wordt een al dan niet uitgebreide samenvatting van het werk gegeven. De bedoeling is wel dat dit tot 1 bladzijde beperkt blijft.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Lijst van figuren en tabellen

Lijst van figuren

4.1	Component diagram Tracklytics library.	16
4.2	Deployment diagram Tracklytics library.	19
4.3	Aanmaken van een core object van de Tracklytics library.	20
4.4	Aanmaken en aanpassen van een core object van de Tracklytis library via de update call.	20
4.5	Gegevens verzenden vanuit de library naar de back end	21
5.1	Klassediagram Tracklytics library.	26
5.2	Structuur van de Tracklytics database.	31
5.3	Dashboard: Counter overzicht.	34
5.4	Dashboard: Gauge overzicht.	35
5.5	Dashboard: Histogram overzicht.	35
5.6	Dashboard: Meter overzicht.	36
5.7	Dashboard: Timer overzicht.	36
5.8	Dashboard: Timer overzicht per versie.	37
5.9	Dashboard: Instellingen overzicht.	37
7.1	Grafiek vergelijking tijden aanmaken van een Counter.	54
7.2	Grafiek vergelijking tijden verhogen van een Counter.	58
7.3	Grafiek vergelijking tijden verlagen van een Counter.	58
7.4	Grafiek vergelijking tijden aanmaken van een Timer.	59
7.5	Grafiek vergelijking tijden stoppen van een Timer.	59
7.6	Grafiek vergelijking tijden aanmaken van een TimerAggregate object. . .	60
7.7	Grafiek vergelijking tijden stoppen van een TimerAggregate object. . . .	60
7.8	Grafiek vergelijking tijden aanmaken van een Histogram.	61
7.9	Grafiek vergelijking tijden addEntry van een Meter.	61
7.10	Grafiek vergelijking tijden addEntry van een MeterAggregate object. . .	62
7.11	Grafiek vergelijking tijden aanmaken van een Gauge.	62
7.12	Grafiek vergelijking tijden aanmaken van een GaugeAggregate.	63
7.13	Grafiek vergelijking tussen het aanmaken van een Gauge en een GaugeAggregate object.	64

7.14	Grafiek vergelijking tussen het aanmaken van een Gauge en een GaugeAggregate object zonder opslaan op de harde schijf.	65
7.15	Grafiek vergelijking tussen het aanmaken van een Meter en een MeterAggregate object.	65
7.16	Grafiek vergelijking tussen het aanmaken van een Meter en een MeterAggregate object zonder opslaan op de harde schijf.	66
7.17	Grafiek vergelijking tussen het aanmaken van een Timer en een TimerAggregate object.	66
7.18	Grafiek vergelijking tussen het aanmaken van een Timer en een TimerAggregate object zonder opslaan op de harde schijf.	67
7.19	Grafiek vergelijking tussen het aanroepen van de stop methode op een Timer en het aanroepen van de stop methode op een TimerAggregate object.	67
7.20	Grafiek vergelijking tussen het aanroepen van de stop methode op een Timer en het aanroepen van de stop methode op een TimerAggregate object zonder opslaan op de harde schijf.. . . .	68

Lijst van tabellen

7.1	Resultaten call tijd Counter (in seconden)	55
7.2	Resultaten call tijd Timer (in seconden)	56
7.3	Resultaten call tijd Timer met Aggregatie (in seconden)	56
7.4	Resultaten call tijd Histogram (in seconden)	56
7.5	Resultaten call tijd Meter (in seconden)	57
7.6	Resultaten call tijd Meter met Aggregatie (in seconden)	57
7.7	Resultaten call tijd Gauge (in seconden)	57
7.8	Resultaten call tijd Gauge met Aggregatie (in seconden)	63
7.9	Resultaten metingen van het aanmaken van een Counter (in FPS) . . .	64
7.10	Resultaten metingen van het stoppen van een Timer zonder opslaan op harde schijf (in FPS)	64

Hoofdstuk 1

Inleiding

DevOps is een concept, gelanceerd in 2008 [35], om developers en IT operation professionals beter de laten samenwerken. Dit zorgt ervoor dat, op moment van schrijven, het concept nog relatief nieuw is en er nog veel onderzoek gedaan kan worden naar gebieden waar DevOps kan helpen om de kwaliteit van software te verbeteren en hoe software sneller op de markt kan gebracht worden. Één van die gebieden is het ontwikkelen van mobiele applicaties. Hoewel dit aspect van software ontwikkeling steeds populairder wordt, is het onderzoek hiernaar nog steeds minimaal. De uitdaging is om de DevOps concepten te combineren met het ontwikkelen van mobiele applicaties.

Deze thesis gaat deze uitdaging aan en probeert DevOps te integreren in het ontwikkelen van mobiele applicaties. Er wordt gefocust op één aspect van de DevOps concepten, namelijk het monitoren van mobiele applicaties. Dit is de rode draad doorheen deze thesis. Er wordt gezocht naar mogelijkheden om deze uitdaging te volbrengen.

Om dit onderzoek verder in te leiden wordt de context uitgelegd waarin deze thesis zich bevindt. In het context hoofdstuk worden de DevOps concepten uitgelegd om een beeld te schetsen waar het onderzoek zich op focust. Later wordt er besproken waarom het relevant is dat mobiele applicaties gemonitord worden. Deze stellingen worden ondersteund door het geven van enkele development scenario's die in de realiteit kunnen voorkomen. Er worden enkele voorstellen tot oplossingen voor het probleem beschreven met hun voor- en nadelen.

Uit de voorgestelde oplossingen wordt in deze thesis één oplossing gekozen, namelijk het monitoren van mobiele applicaties door een library in te bouwen. In het hoofdstuk doelstellingen wordt er beschreven welke doelstellingen deze library moet hebben om een succesvolle library te vormen. De doelstellingen worden opgesteld aan de hand van twee aspecten, namelijk de impact op de performance van de applicatie en de hoeveelheid moeite een developer in het inbouwen van de library moet steken. Nadien wordt er besproken hoe data kan worden weergegeven om de

developer te kunnen helpen bij het ontwikkelen van een mobiele applicatie.

De doelstellingen worden omgevormd in een architectuur van de library die ontwikkeld wordt in deze thesis. Aan de hand van diagrammen wordt aangegeven hoe de componenten van de library samenwerken om de applicatie te monitoren. Ten slotte worden uitbreidingen besproken die de library verrijken met extra functionaliteit om developers meer vrijheid te geven in het monitoren van de applicatie.

Deze architectuur gecombineerd met enkele uitbreidingen is geïmplementeerd in een functionele library voor iOS, het mobiele besturingssysteem van Apple. In het hoofdstuk over de implementatie wordt uitgelegd hoe de structuur van de library in elkaar zit en welke technologieën gebruikt zijn om de library te schrijven. Daarnaast wordt de implementatie van de back end uitgelegd en er wordt besproken hoe de mobiele library zich connecteert met de back end. Om de data weer te geven is een dashboard ontwikkeld waarop deze data grafisch wordt weergegeven. Welke technologieën gebruikt worden en welke keuzes gemaakt zijn om dit dashboard te ontwikkelen en de data data weer te geven worden in dit hoofdstuk uitgelegd. Ten slotte worden de openstaande uitdagingen bekeken die als uitbreiding kunnen worden geïmplementeerd in deze library.

Het is belangrijk dat developers weten hoe ze deze library kunnen inbouwen in de applicatie die ze wensen te monitoren. Om developers hierbij te helpen wordt er een tutorial gegeven die stap voor stap aangeeft hoe de library ingebouwd kan worden in de applicatie. Er wordt uitgelegd hoe elk type meetobject gebruikt kan worden en welke parameters nodig zijn om de library te doen functioneren zoals het hoort.

Om te kijken of de ontwikkelde library voldoet aan de doelstellingen die voorop werden gesteld, hebben we de library geëvalueerd. Er wordt eerst gekeken naar de impact die de library heeft op de performance van de applicatie door te kijken naar hoe lang het duurt om de methodes uit te voeren enerzijds en anderzijds te kijken naar welke invloed de library heeft op de framerate van een applicatie. Daarnaast wordt er gekeken naar hoe schaalbaar de back end van de library is. Naast de performance van de applicatie wordt er gekeken hoeveel moeite een developer nodig heeft om de library in te bouwen in de applicatie. Ten slotte wordt er uit deze resultaten een conclusie getrokken omtrent de library.

Om deze thesis af te sluiten wordt er een besluit getrokken uit alle conclusies en stellingen die gemaakt zijn in de voorgaande hoofdstukken.

Hoofdstuk 2

Context

Zoals de titel al aangeeft is deze thesis gebaseerd op het concept DevOps. DevOps staat voor Developer Operations en probeert de communicatie, samenwerking en integratie tussen developers (Dev) en IT operation professionals (Ops) te verbeteren [33]. Het volgende citaat is de definitie van DevOps:

DevOps is the union of people, process, and products to enable continuous delivery of value to our end users.
Donovan Brown, DevOps Senior Program Manager at Microsoft [17]

Met DevOps willen developers en managers het ontwikkelingsproces van software versnellen. Het DevOps proces bestaat uit een set van verschillende tools om dit doel te bereiken, namelijk [35]:

- Code: code ontwikkeling en continuous integration tools
- Build: version control, samenvoegen van code
- Test: het testen van de applicatie om bugs en performance issues eruit te halen
- Package: het samenvoegen van de code om een software pakket van te maken
- Release: het uitgeven van de software, automatisatie hiervan
- Configure: het configureren van de infrastructuur
- Monitor: Applications performance monitoring, End user ervaring

In deze thesis wordt er gefocust op het monitoren van de software om developers te helpen bij het verbeteren van de software. Het monitoren van software baseert zich op twee eigenschappen van software systemen, namelijk: het meten van performance en het meten van de end user ervaring met de software. Door de software te monitoren kan men Feedback-Driven Development uitoefenen [15]. Feedback-Driven development is het integreren van de runtime monitoring data in de tools die developers gebruiken om de software te ontwikkelen (bv. IDE). Door deze integratie

kunnen developers sneller en gemakkelijker bugs in de software vinden, omdat de gebruikte tools dit door de gemonitorde data kunnen aangeven.

Runtime monitoring is het monitoren van een software systeem dat in productie genomen is om zo problemen te ontdekken die in de testfase niet uit het systeem gehaald konden worden. Met een systeem om de software at runtime te monitoren kunnen de developers en managers ontdekken hoe gebruikers de software werkelijk gebruiken en waar de performance problemen zich bevinden. Runtime monitoring systemen bestaan al langer, enkele voorbeelden zijn New Relic [4], Google Analytics [22] en Metrics [3]. Dit soort systemen collecteren data uit een software systeem en geven deze weer in een dashboard.

Continuous delivery van software is het hoofddoel van DevOps. Continuous delivery is het onmiddellijk aanleveren van nieuwe features of oplossingen voor problemen zonder een echt release plan en is een onderdeel van agile software development [13] [18]. Agile software development methodes moeten ervoor zorgen dat de kost om tijdens het ontwikkelen van een software project zaken te veranderen zo laag mogelijk blijft [24]. Het baseert zich op twee concepten: de meedogenloze eerlijkheid van werkende code en de effectiviteit van mensen die met goodwill samenwerken. DevOps is ontstaan uit de toenemende populariteit van agile software development. Hoewel DevOps en agile development soortgelijkend zijn, verschillen ze op een aantal belangrijke aspecten. Agile development is een verandering van denkwijze van developers, terwijl DevOps een verandering in de cultuur van de organisatie is.

Deze thesis situeert zich in de wereld van mobiele applicaties. Het ontwikkelen van mobiele applicaties is in vele opzichten gelijkend aan het ontwikkelen van software voor andere embedded systemen. Ze delen de volgende gemeenschappelijke problemen: security, performance, betrouwbaarheid en gelimiteerde opslagruimte [34]. Het ontwikkelen van mobiele applicaties verschilt van het ontwikkelen van applicaties voor embedded systemen in de volgende aspecten:

- Potentiële samenwerking tussen verschillende applicaties
- Het aanroepen en gebruiken van verschillende sensoren
- Native en hybrid (mobile web) applicaties. Hybrid applicaties kunnen in een mobiele webbrowser uitgevoerd worden, dit is onmogelijk bij embedded systemen.
- Ondersteuning van verschillende hardware en besturingssystemen
- Security tegen malware
- User Interface. Een embedded applicatie kan eender welke UI hebben, terwijl de een mobiele applicatie een deel van het design uit het systeem zelf moet halen (denk maar aan een statusbar met de tijd in, de standaard terug knoppen, ...)

- Test complexiteit. Een hybrid mobiele applicatie testen is complex, omdat deze de uitdagingen van web applicaties delen, maar er moet ook rekening gehouden worden met de eigenschappen van mobiele telefoonnetwerken.
- Batterijverbruik. Embedded applicaties kunnen geoptimaliseerd worden om zo weinig mogelijk batterij te verbruiken, terwijl mobiele applicaties deels afhankelijk zijn van het resource management van het besturingssysteem.

Het doel is om de DevOps concepten te gebruiken in mobiele applicaties. In dit geval dus het monitoren van mobiele applicaties. Er zijn al enkele bestaande oplossingen, namelijk New Relic [4] en Google Analytics [22].

New Relic is een closed source mobiele library en kiest ervoor om zich puur te focussen op prestaties. De library genereert crash rapporten en gedetailleerde performance rapporten en geeft deze weer in een online dashboard.

Google Analytics is een closed source library ontworpen door Google, inc die ingebouwd kan worden in een mobiele applicatie. Google Analytics focust zich vooral op het monitoren van het gebruik van de applicaties, zoals welke schermen men het vaakst bezocht, welke acties de gebruiker onderneemt in de applicatie, etc.

In deze thesis wordt er gefocust op het monitoring aspect van DevOps in mobiele applicaties. In de rest van dit hoofdstuk wordt er gekeken naar de relevantie van het monitoren van mobiele applicaties. Er wordt afgesloten door te kijken naar een aantal development scenario's om de relevantie van het monitoren van mobiele applicaties te ondersteunen.

2.1 Relevantie monitoren mobiele applicaties

Op het moment van schrijven is het 2016; het jaar waarin ongeveer één op de vijf mensen een smartphone gebruikt in het dagelijkse leven [31]. Een smartphone bevat meerdere mobiele applicaties, ontwikkeld door developers wereldwijd. In de App Store van Apple staan ongeveer 1.5 miljoen verschillende applicaties. Een karakteristiek van de mobiele app wereld is dat er voor een bepaalde app vaak een of meerdere alternatieven bestaan. Dit wil zeggen dat de eigenaars van de app zich moeten proberen onderscheiden van de andere apps. Dat kan op meerdere manieren: door een praktisch design, door een snellere app te hebben, etc. Door tijd en/of geld te investeren in het verbeteren van de applicatie kan de applicatie de meest gebruikte applicatie worden en blijven in zijn categorie.

Met het design wordt de structuur van de gebruikersinterface van de applicatie bedoeld. Dit design bepaalt waar welke UI elementen komen te staan. Er wordt hier niet bedoeld op de esthetiek van de gebruikersinterface. Het design wordt meestal op voorhand vastgelegd alvorens de applicatie ontwikkeld wordt, zodat de developer dit kan gebruiken bij het implementeren. Er kan niet objectief gezegd worden wat

een goed of een slecht design is door een ontwikkelaar omdat dit een persoonlijke mening is. Het design kan enkel écht beoordeeld worden door de gebruikers van de applicatie. Dit komt omdat er een verschil kan zijn in hoe de eigenaars van de applicatie denken dat de applicatie gebruikt wordt en in hoe de gebruikers de applicatie gebruiken. Als er een verschil in deze denkwijze zit, dan zou het design best aangepast worden naar de smaak van de gebruikers. Dit bevordert enerzijds de kwaliteit van de applicatie en anderzijds creëert het een band tussen gebruiker en developer. Indien er geen input komt van de gebruikers kan het nog steeds zijn dat de applicatie niet gebruikt wordt hoe de eigenaar het denkt. Er moet dan ontdekt worden welke elementen gebruikt worden en welke niet. Het monitoren van deze elementen kan ervoor zorgen dat er een goed beeld gevormd wordt voor de eigenaar die met deze informatie zijn inzicht in de applicatie kan veranderen. Dit zorgt ervoor dat de eigenaar betere beslissingen kan maken omtrent de toekomst en de verdere ontwikkeling van de applicatie. Zelfs al zijn er reviews van de gebruikers, dan nog is het voor de eigenaar meestal onmogelijk om te ontdekken welke elementen in de applicatie gebruikt worden en welke niet. Het is dus belangrijk om de applicatie te monitoren, ookal is er veel input van de gebruikers.

De prestaties van een applicatie zijn belangrijk voor een eigenaar omdat deze een impact hebben op de gebruiksvriendelijkheid van de applicatie. De meeste problemen die te maken hebben met prestaties zijn op te lossen door voldoende testen te schrijven en hiermee bugs en trage code sequenties uit de code te halen, maar sommige prestatie problemen doen zich enkel voor bij een significant gebruikersaantal. Drie uit de top tien van meest vermelde klachten van mobiele applicaties hebben te maken met de prestaties van een applicatie [26], namelijk: *Resource-Heavy*, *Slow or lagging* en *Frequent Crashing*. Deze klachten worden vaak in reviews van gebruikers geuit. De eigenaars kunnen uit deze reviews opmaken dat er een probleem is, maar developers weten niet zeker waar het fout loopt uit die reviews. Om uit te vinden waar de prestatie problemen zitten is het voor de developers handig om de applicatie te monitoren waar bottlenecks kunnen ontstaan. Hierdoor kunnen ze de exacte oorzaak van het probleem vinden. Op deze manier kan een potentiële probleem al ontdekt worden voor deze in de reviews van de gebruikers opduikt. Dit zorgt ervoor dat de tevredenheid niet naar beneden gaat, omdat het probleem op voorhand al ontdekt wordt.

De reviews die door de gebruikers worden gelezen moeten met een korrel zout genomen worden; mensen hebben de intentie om te overdrijven. De eigenaars kunnen hiervoor gebruik maken van een methode, ontwikkeld door de Carnegie Mellon Universiteit [19], die een analyse uitvoert van deze reviews en de inconsistente reviews eruit filtert. Zo kan er een goed standpunt gevormd worden rond de kwaliteit van de applicatie. Maar zelfs met die methode zijn reviews nog steeds subjectief als het gaat over prestaties. De enige manier om objectief te redeneren hierover is door middel van metingen uit te voeren. Er zijn meerdere mechanismen om dat soort data te collecteren, met elk zijn voor en nadelen, namelijk: *Built-in OS support*, *een monitoring library*, *een dedicated test environment*, etc.

Built-in OS support is een mechanisme dat door de ontwikkelaars van het besturingssysteem ingebouwd moet worden. Dit mechanisme kan dan verschillende data meten en collecteren, zoals bv. CPU load, hoeveelheid gebruikt RAM geheugen, netwerkgebruik, etc. Het voordeel aan zo'n systeem is dat dit low level kan ingebouwd worden in de code van het besturingssysteem. Dit wil zeggen dat deze data rechtstreeks gecollecteerd kan worden. Deze data moet dan nog doorgegeven kunnen worden aan de developers zodat zij deze kunnen verwerken en beoordelen. Het nadeel hieraan is dat developers niet kunnen aangeven welke data er gecollecteerd wordt. Ze kunnen zich enkel baseren op de data die er voor hen gecollecteerd wordt door het besturingssysteem. Met dit mechanisme is het mogelijk om problemen te ontdekken, maar is het moeilijker om deze te identificeren in de applicatie, omdat bv. het mechanisme kan aangeven dat er enorm veel RAM gebruikt wordt, maar men weet dan niet meteen waar de bug zich bevindt die het RAM geheugen vol heeft gestoken.

Een monitoring library is een high level softwarepakket dat ingebouwd moet worden in de applicatie en waarmee developers hun applicatie kunnen monitoren terwijl de applicatie al in productie genomen is. De developer kan meetpunten aangeven om data te kunnen collecteren. Deze data moet dan worden weergegeven aan de developer. Deze weergave verschilt per monitoring library. Het voordeel hieraan is dat developers enkel de data collecteren die ze nodig hebben van de applicatie en zo een compacter overzicht krijgen van deze data. Dit zorgt tevens voor meer vrijheid voor de developers. Het nadeel is dat deze implementatie trager is dan dat dit in het besturingssysteem ingebouwd zou worden. Met een monitoring library kan er ook enkel high level data gecollecteerd wil worden, wat wil zeggen dat vele informatie niet beschikbaar is omdat de API van het besturingssysteem dit niet aanbiedt. Indien een monitoring library een bug ontdekt als de applicatie in productie is, dan kunnen de developers dit probleem sneller identificeren, omdat ze weten waar ze de meetpunten geplaatst hebben.

Een dedicated test environment is een omgeving waar er verschillende tests op de applicatie worden uitgevoerd. Via deze methode kan men, alvorens men de applicatie in productie brengt ervoor zorgen dat bugs en de meeste performance issues uit de applicatie kunnen gehaald worden. Een dedicated test environment kan enkel in de development fase gebruikt worden en is een voorbereiding voor het in productie brengen van de applicatie. Nadat de applicatie in productie is genomen kan de test environment enkel gebruikt worden om proberen problemen op te lossen die binnengekomen zijn als bug reports van klanten. Via dit test environment kunnen ze dan proberen om deze bug te identificeren, repliceren en op te lossen.

De developer zal dus een keuze moeten maken welke oplossingen hij gebruikt in zijn applicatie. Het beste zou zijn om deze oplossingen te combineren, omdat er dan voor gezorgd kan worden dat bijna elke bug eruit kan gehaald worden en dat developers bugs (en de plaats waar ze voorkomen) kunnen ontdekken indien de

applicatie al in productie is. Dit zorgt ervoor dat er veel tijd kan bespaard worden in het zoeken naar bugs. Er moet wel een evenwicht gevonden tussen het ontdekken en oplossen van bugs en de moeite die een developer moet doen om ervoor te zorgen dat dit ontdekt kan worden. Dit staat geheel in het kader van DevOps, omdat men zo snel mogelijk nieuwe versies wil uitrollen en indien de developer dan veel moeite moet steken in het ontdekken van bugs in de applicatie alvorens en tijdens de applicatie in productie is genomen, dan kan de uitrol niet zo snel gebeuren als gehoopt.

Een mobiele applicatie draait op een besturingssysteem dat speciaal ontworpen is voor de mobiele telefoons. Android en iOS zijn de twee besturingssystemen die het waard zijn om te vermelden [28]. Een mobiele applicatie wordt dus meestal voor beiden ontwikkeld om een zo groot mogelijk aantal gebruikers te bereiken. Applicaties ontwikkelen voor de andere besturingssystemen is meestal niet rendabel voor de moeite en het geld dat erin moet gestoken worden. Zoals al vermeld in dit hoofdstuk 2 is er een verschil tussen het ontwikkelen van mobiele applicaties en embedded applicaties. Vele van deze verschillen komen overeen met de verschillen van het ontwikkelen van websites, internet applicaties en pc applicaties. Dit moet in rekening gebracht worden bij het monitoren van mobiele applicaties. De verschillen in het monitoren van mobiele applicaties met het monitoren van websites, internet applicaties en pc applicaties wordt besproken in de volgende alinea's.

Indien de applicaties native worden ontwikkeld, dan moet de applicatie een keer voor Android en een keer voor iOS ontwikkeld worden. Dit zorgt ervoor dat er verschillende bugs en/of bottlenecks in de verschillende applicaties kunnen zitten. Het zou verkeerd zijn om gecollecteerde data van de twee applicaties samen te verwerken. Een desktop computer of laptop heeft in de meeste gevallen ofwel een WiFi verbinding of een ethernet verbinding met het internet. De internetverbinding van een smartphone varieert tussen WiFi en een mobiel netwerk (4G, 3G, Edge, ...). Deze connecties hebben een verschillende snelheid en moeten dus anders geïnterpreteerd worden.

Bij het monitoren van mobiele applicaties is het noodzakelijk dat een monitoring library zo weinig mogelijk resources (CPU, batterij, netwerk) gebruikt. Zoals eerder al vermeld is dit één van de meest voorkomende klachten bij gebruikers van een mobiele applicatie. Bij pc applicaties wordt hier zelden rekening mee gehouden, omdat de prestaties van de resources hier significant hoger zijn dan bij smartphones. Bij internet applicaties is dit niet van toepassing, omdat dit heel erg af hangt van de implementatie van de browser.

Het monitoren van mobiele applicaties is op veel vlakken een handige feature. Enerzijds om als ondersteuning te dienen voor de eigenaars om de gebruikers van de applicatie tevreden te stellen en de applicatie te kunnen verbeteren/veranderen in functie van de gebruiker en hoe de applicatie gebruikt wordt. Anderzijds dient het monitoren van de applicatie als een hulpmiddel voor de developers om te kunnen ontdekken of er bugs in de applicatie zitten en ook de plaats van voorkomen te

identificeren zodat deze opgelost kunnen geraken in een volgende versie. Er moet een keuze gemaakt worden tussen het gebruik van een bestaand monitoringsysteem of het zelf ontwikkelen van zo'n systeem met alle voordelen en nadelen in rekening gebracht.

2.2 Development Scenarios

In deze sectie worden er enkele scenario's uitgelegd die aantonen dat het monitoren van een mobiele applicatie belangrijk is. Deze sectie wordt opgedeeld in twee verschillende onderdelen die gemonitord kunnen worden, namelijk: *prestaties en gebruik*.

2.2.1 Prestaties

Zoals eerder al vermeldt zijn de prestaties van een mobiele applicatie één van de belangrijkste klachten van gebruikers van mobiele applicaties. Developers moeten er dus voor zorgen dat de prestaties van de applicatie zo hoog mogelijk zijn en dat er geen performance bugs in de applicatie voorkomen.

Om performance problemen te ontdekken moet er een manier zijn om deze problemen te kunnen ontdekken. Hier kan een monitoring systeem een oplossing bieden. Dit kan een monitoring systeem zijn dat geïmplementeerd is in het besturingssysteem (*Built-in OS support*), een third party monitoring library of een dedicated test environment.

De informatie die de developer kan terugkrijgen hangt af van welk systeem de developer gekozen heeft.

Er zijn een aantal zaken die de applicatie traag kunnen maken, namelijk: CPU-intensieve code, veel RAM geheugen, een trage server die ervoor zorgt dat content traag opgehaald wordt, ...

CPU-intensieve code gebruikt veel CPU tijd en kan de applicatie traag doen lijken. Dit kan ontdekt worden met een monitoringssysteem en afhankelijk van welke keuze van systeem kan de developer snel vinden waar deze code zich bevindt. De developer zou dan best deze code, indien mogelijk, herschrijven om de applicatie sneller te doen lopen.

Indien de applicatie **veel RAM geheugen** gebruikt, dan kan dit de applicatie traag doen lopen, omdat het RAM geheugen naar disk moet geswapt worden en deze acties de applicatie vertragen. De developer moet er dan voor zorgen dat de applicatie minder RAM geheugen gebruikt of de bug, waar veel RAM geheugen gebruikt wordt (bv. een oneindige while loop die in de achtergrond loopt), oplossen.

Mobiele applicaties halen vaak data van het internet of versturen data naar het internet. Indien deze acties veel tijd kosten hebben deze een impact op de snelheid van de applicatie. Dit soort problemen kunnen voorkomen indien het aantal gebruikers van de applicatie snel omhoog gaat en de servers de bijkomende trafiek niet aankunnen. Het is belangrijk voor de developer om te ontdekken dat deze problemen zich voordoen. Smartphones hebben een WiFi verbinding of een mobiele dataverbinding (4G, 3G, Edge,...). Deze karakteristiek moet mee in rekening genomen worden bij het beoordelen of er een probleem is en waar het probleem zich voordoet.

Buiten deze voorbeelden zijn er nog andere kenmerken die belangrijk zijn voor een mobiele applicatie, maar deze voorbeelden schetsen het beeld waarom het monitoren van mobiele applicaties belangrijk is.

2.2.2 Gebruik

Er bestaat een verschil in de gedachte van de ontwikkelaars van een applicatie over hoe de mobiele applicatie gebruikt wordt en hoe de gebruikers de applicatie werkelijk gebruiken. Deze mismatch zorgt ervoor dat ontwikkelaars verkeerde beslissingen kunnen nemen in verband met de mobiele applicatie. Ze kunnen bijvoorbeeld een veelgebruikte feature uit de applicatie halen, omdat ze niet weten dat deze feature vaak gebruikt wordt. Dit probleem komt voor in de lijst met meest voorkomende klachten over mobiele applicaties [26].

Om te weten te komen wat er gebruikt wordt in de applicatie kan er bijvoorbeeld geteld worden hoe vaak er op welke *button* gedrukt wordt in de applicatie. Zo kan men zien welke *buttons* er het meeste gebruikt worden. Men kan bijvoorbeeld ook gaan tellen hoe vaak een bepaald scherm bezocht wordt. Met deze informatie kan een ontwikkelaar ervoor kiezen om een bepaald scherm eruit te halen of weten dat ze een bepaald scherm zeker niet uit de applicatie mogen verwijderen.

Dit zijn enkele voorbeelden van hoe een ontwikkelaar een voordeel kan halen door een monitoringsysteem in zijn applicatie te verwerken. Er zijn natuurlijk nog andere scenario's waarin een monitoringsysteem belangrijk kan zijn in het ontwikkelen van mobiele applicaties en zo de klachten van gebruikers tot een minimum te houden.

Hoofdstuk 3

Doelstelling

Het onderzoek van deze thesis baseert zich op twee invalshoeken, enerzijds hoe de developers en eigenaars van een mobiele applicatie geholpen kunnen worden bij het ontwikkelen en verbeteren van die applicatie en anderzijds wat de trade-off is tussen enerzijds hoeveel de applicatie gemonitord wordt en anderzijds de impact op de performance van de applicatie, de developer effort, etc.

De eerste invalshoek gaat over hoe we developers kunnen helpen bij het ontwikkelen van een mobiele applicatie door de developers problemen te laten identificeren (zoals trage code sequenties) en de eigenaars te laten ontdekken welke componenten van de applicatie het meest gebruikt worden en welke het minste. Met deze informatie weten de developers waar ze verbeteringen kunnen aanbrengen en weten eigenaren in welke functies ze wel of niet moeten investeren. Langs de andere kant is het belangrijk om te weten wat de impact is van het verkrijgen van de informatie op de prestaties van de applicatie en hoeveel moeite en tijd er nodig is van de developer om deze informatie te kunnen vergaren.

Om dit onderzoek te doen slagen wordt er een library gebouwd die developers in hun applicaties kunnen inbouwen om hun applicatie te monitoren. In de volgende sectie wordt uitgelegd wat het doel is van deze library. In het verdere verloop van deze thesis wordt uitgelegd hoe de architectuur achter de gebouwde library in elkaar zit en waarom de keuzes zijn gemaakt voor die structuur. Nadat deze architectuur beschreven is wordt er een uitleg gegeven over de specifieke implementatie van de library. Ten slotte wordt de library geëvalueerd om te kijken of de kwaliteit van de library voldoende is om gebruikt te worden in de applicaties.

3.1 Library

Het doel van de library is om developers deze library te laten inbouwen in hun applicatie en met behulp van de methodes aangeboden door deze library de applicatie te kunnen monitoren. Deze methodes moeten voldoende zijn om alle aspecten van

de applicatie te kunnen monitoren.

Om een kwaliteitsvolle library te bouwen zijn er een aantal aspecten die belangrijk zijn om in het achterhoofd te houden:

- performance impact
- schaalbaarheid
- bruikbaarheid
- beschikbaarheid

Deze aspecten worden besproken in het komende deel van dit hoofdstuk.

3.1.1 Performance impact

Het is van cruciaal belang dat een applicatie niet traag is en responsief is naar de gebruiker toe. De verwerkingskracht (CPU) van een mobiel apparaat is relatief beperkt. Deze twee eigenschappen gecombineerd zorgt ervoor dat de library een zo beperkt mogelijke impact mag hebben op deze verwerkingskracht. Indien deze library relatief veel CPU tijd inneemt, dan blijft er minder CPU tijd over voor de applicatie, wat ervoor zorgt dat de applicatie trager wordt indien deze CPU-intensief is. Het is dus van cruciaal belang dat de library geoptimaliseerd wordt om zo weinig mogelijk CPU tijd te verbruiken. Een bijkomend voordeel dat verschijnt indien de library zo minimaal mogelijk CPU tijd verbruikt is dat de impact van de applicatie op het batterijverbruik verminderd wordt.

Naast het CPU verbruik is het ook van belang dat het netwerkgebruik zo efficiënt mogelijk gebruikt wordt. In een mobiel apparaat wordt de netwerkinterface in slaapstand gehouden tot deze gebruikt moet worden en op dat moment wordt deze interface geactiveerd. Dit wordt gedaan omdat de netwerkinterface relatief veel energie verbruikt indien deze aanstaat. Indien de netwerkinterface zo efficiënt mogelijk gebruikt wordt kan deze vaker in slaapstand gezet worden en is de impact van de library op het batterijgebruik minder dan indien deze inefficiënt gebruikt wordt. Deze efficiëntie zorgt er ook voor dat de applicatie de netwerkinterface zo vaak mogelijk kan gebruiken en hier zo weinig mogelijk vertraging opgelopen wordt als de applicatie het netwerk wil gebruiken.

Niet enkel de efficiëntie van het gebruik van de netwerkinterface is van belang, maar ook de snelheid waarmee de back end reageert op de request en deze een resultaat kan terug geven. Indien het relatief lang duurt eer de back end het resultaat terug geeft, is de optimalisatie van het gebruik van de netwerkinterface voor niets geweest. De snelheid van de back end is dus ook een belangrijk aspect.

De impact op de performance van de applicatie is het belangrijkste aspect in het bouwen van een monitoring library voor mobiele applicaties. Indien deze impact

relatief groot is, dan is het voor developers onmogelijk om deze library in de applicatie in te bouwen en nog steeds een degelijke responsiviteit en snelheid van deze applicatie te behalen.

3.1.2 Schaalbaarheid

Zoals hierboven vermeld is de snelheid van de back end een belangrijk aspect inzake performance van de netwerkinterface. Één aspect van deze snelheid is de implementatie van deze back end. De implementatie moet geoptimaliseerd worden zodat deze zo snel mogelijk een resultaat kan teruggeven aan de library. Het andere aspect in de snelheid van de back end is de load die de servers aankunnen om al de requests te verwerken. De mate van schaalbaarheid is hier een belangrijke factor in.

Een applicatie wordt simultaan gebruikt door gemiddeld N gebruikers waarin de library aanwezig is. Deze applicatie verstuurt per keer gemiddeld X metingen. De library wordt gebruikt door M verschillende applicaties. De applicaties zijn geconfigureerd met een gemiddeld synchronisatie-interval T . Dit geeft dat het gemiddeld aantal requests per tijdseenheid neerkomt op: $RpT = \frac{M*N*X}{T}$. Hoe hoger dit getal, hoe hoger het aantal requests de servers moeten beantwoorden en hoe meer load er per server komt. Elke server S heeft een limiet kwa aantal requests die simultaan kunnen beantwoord worden R . De volgende vergelijking geeft de relatie aan tussen het gemiddeld aantal requests en het aantal requests dat beantwoord kan worden: $S * R > RpT$. Dit wil zeggen dat er (altijd) meer requests moeten kunnen beantwoord worden dan dat er gemiddeld per tijdseenheid aan komen.

3.1.3 Bruikbaarheid

Het primaire doel van een library is om de taken uit te voeren die worden gegeven aan de library. Naast dit primaire doel is het ook de bedoeling dat developers de library kunnen gebruiken zonder veel effort. De bruikbaarheid van de library moet zo hoog mogelijk zijn zodat developers zonder veel tijd en moeite deze kunnen integreren in een applicatie. De volgende punten zijn hierin belangrijk:

- Concreet willen we maximaal vermijden dat de library de developer hindert bij het ontwikkelen van de applicatie.
- Concreet hebben we als doelstelling het vereiste aantal lijnen code en configuratie zo klein mogelijk te houden.
- Concreet willen we dat de developer een zo minimaal mogelijke tijd moet spenderen bij het implementeren van de library.

3.1.4 Beschikbaarheid

Met beschikbaarheid wordt bedoeld dat alle gecollecteerde data opgeslagen wordt op de servers. De data die gecollecteerd wordt is waardevol. Het is de bedoeling dat er

zo'n goed mogelijk beeld geschetst wordt voor de developer. Indien een deel van de data niet opgeslagen wordt in de database kan het zijn dat mogelijke uitzonderlijke data niet opgeslagen wordt, terwijl deze uitzonderlijke data juist de belangrijkste is. Er moet dus voor gezorgd worden dat 100% van de gecollecteerde data gesynchroniseerd wordt naar de server om een zo goed mogelijk beeld te vormen voor de developers en de eigenaars van de applicatie.

Deze aspecten moeten in rekening genomen worden bij het ontwerpen en ontwikkelen van een mobiele monitoring library. De belangrijkste aspecten zijn de performance aspecten, omdat in mobiele applicaties de verwerkingskracht en batterij de bottlenecks zijn.

3.2 Data Visualisatie

Data visualisatie is een belangrijk aspect in de library, omdat de developer uit deze visualisaties conclusies moet kunnen trekken over de prestaties en het gebruik van de applicatie. De verzamelde data van de gebruikers is samengenomen in één centraal punt. Het doel is om deze data samen te voegen en hieruit de nodige informatie te halen en weer te geven aan de developer. Het is dus de zaak om uit te zoeken voor elk soort meting welke informatie belangrijk is. Deze informatie moet worden gevisualiseerd om de developer een compacter overzicht te geven van de resultaten van de metingen. Door de data te groeperen op eigenschappen kan de developer dieper ingaan op de details van de metingen. Indien er niet aan data visualisatie gedaan zou worden, zou de developer zelf de verschillende parameters moeten berekenen uit de ruwe data.

Het uiteindelijke doel is dus om per type meting de parameters te gaan identificeren die nuttig kunnen zijn voor de developer om een conclusie uit dat type meting te kunnen trekken. Daarnaast is het de bedoeling om een software pakket te maken die deze parameters gaat weergeven in een overzicht voor de developer.

Hoofdstuk 4

Architectuur

De oplossing die voorgesteld wordt in deze thesis is een monitoring library voor mobiele applicaties, Tracklytics genaamd. In deze sectie worden de architecturale beslissingen uitgelegd.

Tracklytics stelt een gehele infrastructuur voor die ontworpen is om een mobiele applicatie te monitoren. In de volgende figuur 4.1 worden de belangrijkste componenten beschreven. De hoofdcomponenten zijn: *TracklyticsCore*, *TracklyticsBackend* en het *Dashboard*. Deze vormen het aangeboden Tracklytics systeem. De *UserApplication* stelt een mobiele applicatie voor die het Tracklytics systeem gebruikt.

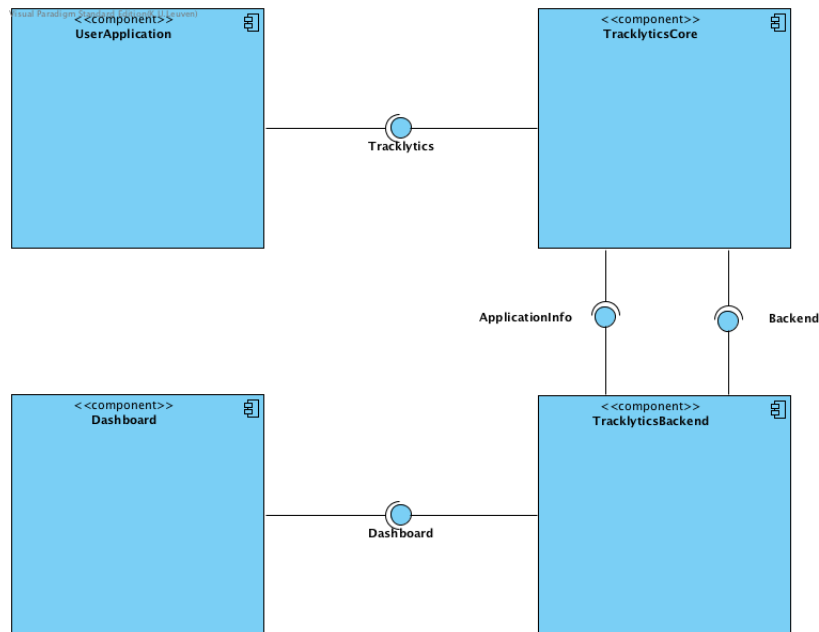
Het Tracklytics systeem bestaat uit:

- een reusable/easy-to-use library voor mobiele applicaties (*TracklyticsCore*)
- een back end service om de data uit de library te verwerken (*TracklyticsBackend*)
- een dashboard dat de data uit de back end weergeeft aan de developer (*Dashboard*)

Dit systeem kan gebruikt worden door mobiele applicaties om de applicatie te monitoren. De componenten van dit systeem zijn ontwikkeld om samen te werken en worden besproken in het volgende gedeelte van dit hoofdstuk.

4.1 Component Diagram

Het component diagram 4.1 duidt aan welke componenten een rol spelen in het bouwen van de Tracklytics library. In de volgende secties worden de verschillende componenten beschreven en hun rol wordt uitgelegd.



FIGUUR 4.1: Component diagram Tracklytics library.

4.1.1 UserApplication

De UserApplication component representeert een applicatie ontwikkeld door developers. De UserApplication gebruikt de Tracklytics interface om metingen uit te voeren en de applicatie te monitoren. Om de developers te helpen bij het monitoren van de applicatie willen we mechanismen aanbieden om het design en de performance van de applicatie te monitoren. Om het design te monitoren willen we het monitoren van volgende elementen ondersteunen:

- Op welke buttons/switches/entry in een tabel gebruikers drukken
- Welke schermen de gebruikers bezoeken
- Het gemiddeld aantal zoekresultaten per zoekopdracht
- Het gemiddelde of de verdeling van het getal dat een gebruiker in een bepaald veld invoert
- Hoe lang de applicatie gemiddeld gebruikt wordt

Om de performance van de applicatie op een hoog niveau te kunnen monitoren willen we volgende elementen kunnen monitoren:

- Hoe lang een stuk code over het uitvoeren ervan doet om na te gaan of deze code niet te traag is.

- De tijd die een request over het internet nodig heeft om te voltooien om te kijken of er hier een vertraging opgelopen wordt.
- Het gemiddeld aantal entries in een array of een NSDictionary (een Map in Java). Indien er geïtereerd wordt over deze datastructuren kan een groot aantal entries ervoor zorgen dat dat stuk code de applicatie vertraagt.
- Het gemiddeld aantal keer dat een bepaalde methode uitgevoerd wordt om te kijken

4.1.2 TracklyticsCore

De TracklyticsCore component is de belangrijkste component van de Tracklytics infrastructuur voor. Deze component bevat de monitoring library die ontwikkeld wordt in deze thesis. Deze component bevat de functionaliteit om de mobiele applicatie te monitoren. Voor elk soort element besproken in vorige sectie moet er een mechanisme ingebouwd zijn in deze component.

Om er zeker van te zijn dat alle data uiteindelijk in de back end geraakt via het netwerk, is er een optionele tijdelijke buffer/cache voorzien die de data tijdelijk naar de harde schijf van het toestel wegschrijft. Indien de data gesynchroniseerd is met de back end wordt deze data verwijderd van de harde schijf van het toestel. Dit mechanisme zorgt ervoor dat elke meting in de back end geraakt en er geen data verloren gaat. De extra indirectie is dat deze data op de harde schijf opgeslagen wordt.

In deze component bevindt zich een subcomponent die de communicatie met de back end verzorgt. Deze is verantwoordelijk voor het afleveren van de data van de metingen uit de library aan de back end. Het is de verantwoordelijkheid van deze subcomponent om alle meta-informatie over de applicatie op te halen uit de back end, zoals de voorkeuren die de developer heeft opgegeven in het dashboard. Deze subcomponent gebruikt een identifier voor de applicatie om deze correct te synchroniseren met de back end zodat het onderscheid kan gemaakt worden tussen deze applicatie en andere applicaties.

Interface De TracklyticsCore component biedt een Tracklytics interface aan. De UserApplication kan deze interface gebruiken om de applicatie te monitoren. De TracklyticsCore stuurt deze data door naar de server.

De interface bestaat uit methodes om de applicatie te monitoren. Tracklytics biedt vijf elementen aan om de applicatie te kunnen monitoren, namelijk:

- Counter: Kan gebruikt worden om evenementen te tellen
- Meter: Kan gebruikt worden om periodiek waardes op te halen.

- Histogram: Kan gebruikt worden om een waarde te verzamelen dat in het dashboard in een histogram gegoten kan worden.
- Timer: Kan gebruikt worden om de lengte van een evenement te meten.
- Gauge: Kan gebruikt worden om een bepaalde waarde te meten.

4.1.3 Dashboard

De Dashboard component representeert een applicatie om de gemonitorde data van de Tracklytics library weer te kunnen geven aan de developers en de eigenaars. Met behulp van grafieken en verwerkte data kunnen de developers en de eigenaars beslissingen nemen omtrent de applicatie. In dit dashboard kunnen instellingen aangegeven worden door de developer voor de applicatie.

4.1.4 TracklyticsBackend

De TracklyticsBackend component representeert de server kant van de library. Hierop draait de code om de gemonitorde data op te slaan in de database. De TracklyticsBackend component bevat dus de database om alle informatie in op te slaan.

De TracklyticsBackend component biedt drie interfaces aan, namelijk: ApplicationInfo, Backend en Dashboard. Deze worden in de volgende sectie besproken.

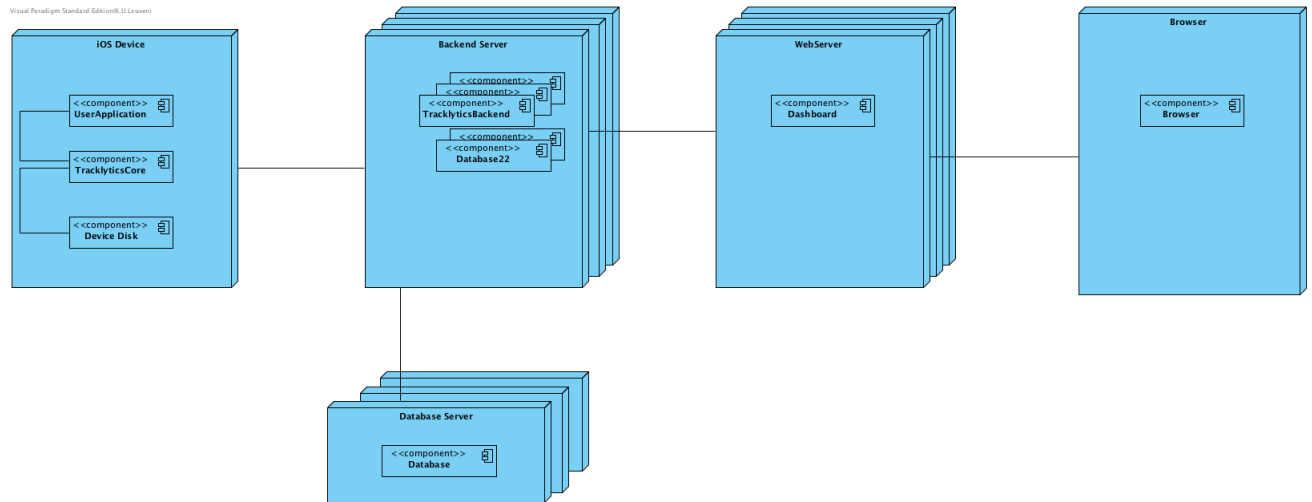
De TracklyticsBackend behandelt elke request die binnenkomt van het Dashboard of van de TracklyticsCore. Om de request van de TracklyticsCore te beantwoorden voegt hij alle informatie die in de request staat toe aan de database en zendt hij een bericht terug met de boodschap dat de request succesvol is of een boodschap met een foutmelding in, zodat de TracklyticsCore deze fout kan verwerken. Om een request van het Dashboard af te werken haalt de TracklyticsBackend de gewenste gegevens uit de database, verwerkt deze informatie en geeft de verwerkte informatie door aan de back end om weer te geven. Om deze functionaliteit aan te bieden moet de TracklyticsBackend een connectie hebben met de database.

Interfaces

ApplicationInfo Deze interface wordt gebruikt door de TracklyticsCore component om voorkeuren die de developer heeft gemaakt in het dashboard op te halen en toe te passen op de monitoring library.

Backend Deze interface wordt gebruikt om data uit te wisselen tussen de TracklyticsCore en de TracklyticsBackend. Dit is de gemonitorde data die de TracklyticsCore verzameld heeft. De TracklyticsBackend verwerkt deze informatie en slaat deze op in de database.

Dashboard Deze interface wordt gebruikt door het dashboard om de data uit de database op te halen om deze weer te kunnen geven aan de developers.



FIGUUR 4.2: Deployment diagram Tracklytics library.

4.2 Deployment Diagram

Het deployment diagram toont op welke nodes de componenten besproken in vorige sectie draaien. Zoals je kan zien draait de UserApplication en de TracklyticsCore op het mobiele toestel van de gebruiker zelf. De TracklyticsBackend draait op een backend server die gerepliceerd kan worden. Deze staat in verbinding met een Database server waar alle data wordt opgeslagen. Het Dashboard draait op een webserver. Deze staat in verbinding met de backend server om de data te kunnen ophalen.

4.3 Belangrijkste flows

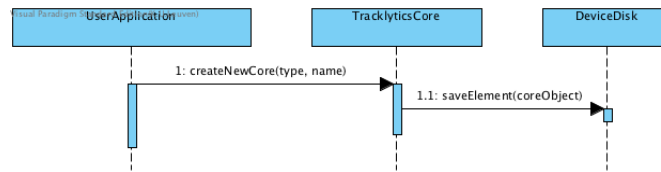
In deze sectie worden de belangrijkste flows in de Tracklytics library beschreven. Deze geven aan hoe de data flow binnen Tracklytics werkt.

De belangrijkste flows door Tracklytics zijn:

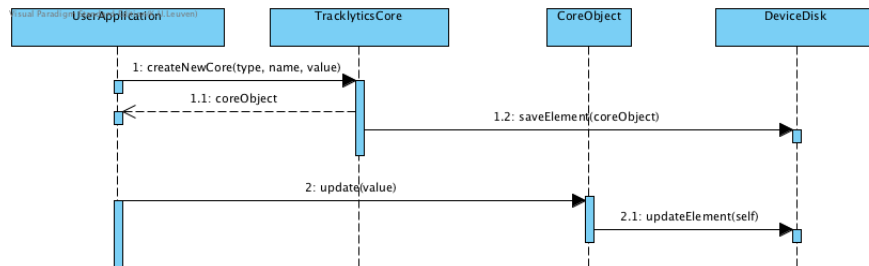
- Tracking en monitoring van gegevens
- Verzenden en opslaan van de gegevens

4.3.1 Tracking en monitoring van gegevens

Het monitoren van de gegevens van de applicatie kan opgesplitst worden in twee verschillende flows.



FIGUUR 4.3: Aanmaken van een core object van de Tracklytics library.



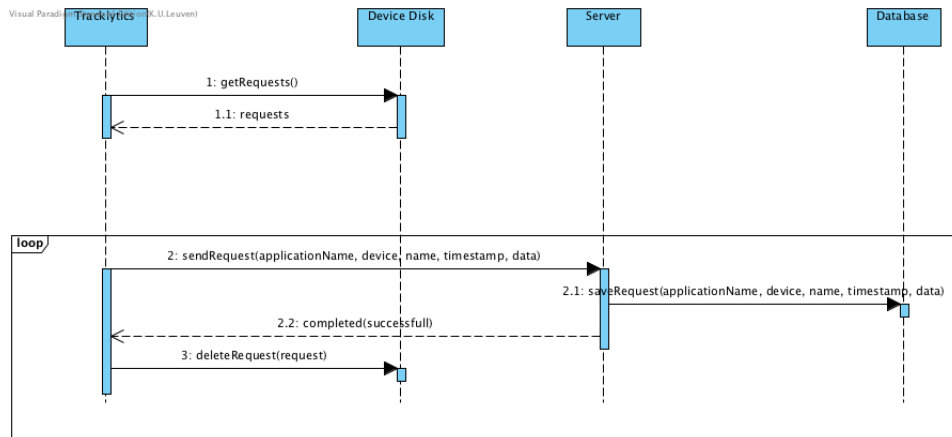
FIGUUR 4.4: Aanmaken en aanpassen van een core object van de Tracklytis library via de update call.

De eerste flow 4.3 toont de meest simpele flow. De developer geeft aan dat een bepaalde waarde gecollecteerd moet worden. De TracklyticsCore houdt deze waarde in de cache totdat deze gesynchroniseerd is naar de server, wat getoond wordt in 4.3.2. Indien de developer dit aangegeven heeft, slaat de TracklyticsCore deze waarde op op de harde schijf van het toestel totdat deze waarde gesynchroniseerd is met de back end. Deze situatie is geldig wanneer er geen object wordt teruggegeven aan de developer om hierop functies te kunnen aanroepen.

De tweede flow 4.4 toont een uitbreiding op de eerste flow 4.3. In deze situatie geeft de TracklyticsCore een object terug aan de developer. Op dit object kan de developer verschillende functies uitvoeren om de data te manipuleren via een update call. De TracklyticsCore past deze update toe op de gecachte waarde en de kopie die op de harde schijf van het toestel staat, indien deze bestaat. In plaats van dat een object uit de cache of van de harde schijf verwijderd wordt, houdt de TracklyticsCore deze bij tot de applicatie wordt afgesloten. Bij het heropstarten van de applicatie moeten deze objecten gesynchroniseerd worden met de server en verwijderd worden van de harde schijf. De objecten zijn al verwijderd uit de cache door het afsluiten van de applicatie.

4.3.2 Verzenden en opslaan van de gegevens

In het volgende diagram 4.5 wordt getoond hoe het verzenden van de gegevens van het toestel naar de server in zijn werk gaat en hoe de gegevens worden opgeslagen in de database.



FIGUUR 4.5: Gegevens verzenden vanuit de library naar de back end

Om de data naar de TracklyticsBackend te verzenden haalt de TracklyticsCore de gegevens die op de harde schijf staan op en voegt deze samen met de gegevens die in zijn cache staan. De TracklyticsCore itereert over deze samengevoegde gegevens en verzendt deze naar de server. De server verwerkt de data die doorgestuurd is en slaat deze verwerkte data op in de database. De server stuurt een antwoord terug naar de TracklyticsCore met daarin ofwel een succes boodschap en een identifier van het object in de database, ofwel met een error message. De TracklyticsCore verwerkt deze boodschap: ofwel is het object een object van het type dat beschreven is in 4.3, dan verwijdert de component het object uit zijn cache en van de harde schijf van het toestel. Ofwel is het een object van het type beschreven in 4.4, dan moet hij de identifier dat mee is gestuurd in het bericht toewijzen aan dat object, zodat bij een update het object wordt aangepast in de database in plaats van dat er een tweede object in de database geplaatst wordt. Ofwel is het een error message, dan moet de TracklyticsCore ofwel opnieuw proberen te synchroniseren, ofwel wachten tot de volgende synchronisatie om het object naar de server door te sturen.

4.4 Uitbreidingen

In deze sectie worden een aantal uitbreidingen besproken die niet standaard in een monitoring library zitten, maar zeer handig kunnen zijn om developers te helpen bij het monitoren van de applicatie.

Data availability Smartphones vallen weleens zonder internet connectiviteit door het wegvallen van een WiFi verbinding of het wegvallen van de mobiele verbinding. De Tracklytics library zendt de gemonitorde data over het internet naar de backend om deze op te slaan in een database. Als de internetverbinding zou wegvallen, dan zou er mogelijk belangrijke data verloren gaan. Om dit scenario te voorkomen

slaan we de data tijdelijk op op de harde schijf van de mobiele telefoon. Indien de smartphone dan terug verbinding met het internet heeft gemaakt kan de data alsnog gesynchroniseerd worden met de backend. De data wordt tijdelijk opgeslagen, dus nadat de data gesynchroniseerd is wordt deze van de smartphone verwijderd. Zo wordt ervoor gezorgd dat de data in ieder geval in de database terecht komt.

On Demand Het is de bedoeling de developer zoveel mogelijk vrijheid te geven in het monitoren van de applicatie. Om deze vrijheid te verhogen is het On Demand aan- of uitzetten van het monitoren aan Tracklytics toegevoegd. Een developer kan vanuit het dashboard aangeven of de applicatie op dit moment gemonitord moet worden of niet. Dit kan gebruikt worden om op de piekdagen of piekmomenten het monitoren aan te zetten. Zo kan uitgezocht worden of er een bottleneck bestaat als het gebruik van de applicatie piekt.

Indien een developer ontdekt dat er geen bottleneck of problemen zijn met de app en deze niet verder gemonitord moet worden, dan is het handig om de monitoring uit te kunnen zetten. Het uitzetten van het monitoren zorgt ervoor dat de prestatiekost van de library geëlimineerd wordt in de applicatie.

Aggregatie De data die doorgestuurd wordt naar de server heeft enkel betrekking op de specifieke gebruiker van de applicatie op zijn telefoon. Om een overzicht te geven van wat de algemene trend is van deze metingen, is het nodig deze data samen te voegen met de data van alle andere gebruikers en deze te aggregeren om een goede analyse hiervan te maken. Er zijn twee plaatsen waar deze aggregatie uitgevoerd kan worden, namelijk op de smartphone zelf in de Tracklytics library of in de backend.

Om bandbreedte te besparen en om verwerkingstijd in de back end te besparen is er de mogelijkheid om de data op het toestel van de gebruiker te aggregeren. Enkel de geaggregeerde data wordt doorgestuurd naar de back end die deze dan in geaggregeerde vorm in de database opslaat. Indien het dashboard deze data dan opvraagt moet de back end deze data nog samen voegen, wat minder tijd en bandbreedte vraagt van de database, omdat de data kleiner is. De back end kan deze data sneller verwerken, omdat hij enkel de geaggregeerde data moet verwerken en niet alle data die de library heeft vergaard. Het nadeel hieraan is dat sommige statistieken (zoals bijvoorbeeld de mediaan) een benadering van de werkelijke mediaan wordt.

De reden om de aggregatie op de back end uit te voeren is dat de back end een grotere verwerkingskracht heeft dan het toestel van de gebruiker en de back end heeft als enige taak data te verwerken en op te slaan in de database en data uit de database te halen, terwijl het toestel naast het monitoren van de applicatie de taak heeft om de applicatie uit te voeren. De back end moet in deze situatie alle data ophalen en verwerken, wat veel bandbreedte en verwerkingskracht kost indien het totaal aantal meetpunten groot is.

Een derde mogelijkheid is om de gebruikers op te delen in twee verschillende groepen, waarbij de ene groep de data aggregaat op het toestel en de andere de aggregatie laat doen in de back end. De back end moet de geaggregeerde en de niet-geaggregeerde data samenbrengen en verwerken indien het dashboard deze opvraagt. Een typische eigenschap is om de groep op te delen op type toestel. Zo kunnen toestellen met een tragere CPU de aggregatie laten gebeuren in de back end en de toestellen met een snellere CPU de aggregatie uitvoeren op het toestel zelf.

Een andere mogelijkheid is om een deel van de data te aggregeren op het toestel en een deel van de data in de back end te laten aggregeren. Zo wordt er toch bandbreedte bespaard en is de benadering van bijvoorbeeld de mediaan juist dan dat alles op het toestel van de gebruiker geaggregeerd wordt. De developer moet de mogelijkheid gegeven worden om het percentage van de data dat geaggregeerd moet worden op het toestel aan te geven. Zo kan er de developer het percentage zoeken dat het best bij zijn applicatie past.

Een functie in het dashboard zou ervoor kunnen zorgen dat de developer de optie krijgt om één van deze mogelijkheden te kiezen die op dit moment actief zou moeten zijn. Zo wordt de developer de vrijheid gegeven om de library te tweakken naar zijn voorkeuren.

AB Testing AB testing is een mechanisme dat grote bedrijven, zoals facebook, gebruiken om nieuwe features uit te rollen naar de gebruikers. Het mechanisme werkt als volgt: er bestaat een versie A en een versie B van de software met B de nieuwere versie. AB testing zorgt ervoor dat de uitrol van versie B geleidelijk verloopt. De gebruikers van de software worden dus in twee (of meerdere) groepen opgedeeld door een bepaalde eigenschap. Deze eigenschap kan eender welke vorm aannemen, bijvoorbeeld: geografische locatie, ingestelde taal, de gebruikte browser, het type toestel, enz. Indien er dan een probleem met versie B is, dan bestaat dit enkel bij de groep die versie B al verkregen is en dus niet bij alle gebruikers. Hierdoor merkt enkel die groep dat er een probleem is en kan versie B aangepast worden om dit probleem op te lossen of eventueel kan ervoor gezorgd worden dat iedereen terug versie A gebruikt tot het probleem in versie B opgelost is.

In mobiele applicaties is het moeilijker om echt aan AB testing te doen, omdat deze applicaties meestal statisch zijn, er moet een update in de app store komen om een nieuwe versie met nieuwe features en code uit te rollen. Dit staat pal tegenover websites die hun webpaginas en code rechtstreeks van een server halen en waar de aanpassingen aan de versies meteen zichtbaar zijn bij de gebruikers. Een workaround van dit probleem is dat de mobiele applicatie de code van de nieuwe versie (B) al kan bevatten en ook nog de code van de oude versie (A) erin heeft staan. Tracklytics kan dan een functie in het dashboard aanbieden om de groepen op te delen in twee (of meerdere) groepen op basis van eigenschappen die de developer kan kiezen. In de mobiele applicatie moeten we dus dat onderscheid kunnen maken welke code

aangeropen moet worden. Het gemakkelijkste is om een codenaam aan de versie toe te voegen, zodat er meerdere versies van de code aanwezig kan zijn. Het nadeel van dit mechanisme is dat de applicatie groter is dan hij zou zijn moest de applicatie enkel de code bevatten die voor die bepaalde groep nodig is.

Security & Privacy Een belangrijk aspect is de privacy van de gebruiker. Zoals eerder al aangehaald is NewRelic een closed source library, wat wil zeggen dat developers niet weten welke informatie er doorgestuurd wordt naar de backend. Dit gegeven zorgt ervoor dat privacygevoelige applicaties deze library niet zouden mogen gebruiken, omdat er gebruikersinformatie gelekt zou kunnen worden naar de backend van NewRelic zoals bijvoorbeeld email-adressen, creditcard gegevens, etc. Om deze reden is ervoor gekozen dat Tracklytics open source is. Zo kunnen developers zeker zijn welke informatie er wordt doorgestuurd naar de backend en is de privacy van de gebruiker wel gegarandeerd. Tracklytics verzamelt enkel metadata van de telefoon en de applicatie. De metadata die gecollecteerd wordt door Tracklytics bestaat uit: het type toestel, de versie, de naam van de applicatie, een identifier van de applicatie, een identifier van het toestel om deze van elkaar te kunnen onderscheiden, de datum en het type connectie waarop het toestel zich bevindt. Dit gegeven zorgt ervoor dat Tracklytics de privacy van de gebruiker garandeert..

Naast privacy is security ook een belangrijk aspect. Er bestaat namelijk een verbinding tussen de mobiele applicatie en de backend over een onveilig netwerk. Het zou dus niet mogen dat de doorgezonden data door een onrechtmatig iemand gecollecteerd wordt of dat hiermee geknoeid wordt. Om deze situaties te voorkomen is ervoor gekozen om via een HTTPS verbinding te werken. Deze verbinding zorgt uit zichzelf voor een veilige verbinding tussen begin- en eindpunt. Als extra veiligheid werkt Tracklytics met HTTP Post in plaats van HTTP Get, zodat de doorgegeven data niet zichtbaar is in de URL naar de backend. Deze combinatie zorgt ervoor dat de gegevens niet onderschept kunnen worden door onrechtmatige personen.

Hoofdstuk 5

Implementatie

In het vorige hoofdstuk werd er beschreven hoe de architectuur van een monitoring library eruit kan zien. In dit hoofdstuk wordt er gekeken naar de implementatie van de Tracklytics library [20]. Er wordt gekeken naar welke technologieën er gebruikt zijn bij het ontwikkelen van de library. Daarnaast wordt er uitgelegd hoe developers de library kunnen gebruiken in hun applicaties. Er wordt dieper ingegaan op de communicatie tussen de Tracklytics library en de server.

In het eerste deel van dit hoofdstuk wordt er gekeken naar de details van de implementatie van de monitoring library voor iOS. Nadien wordt de implementatie van de back end uit de doeken gedaan waarmee de library in contact staat. De implementatie van het dashboard wordt samen met hoe de data gevisualiseerd wordt uitgelegd. Later in dit hoofdstuk wordt de aggregatiestrategie uitgelegd gebruikt in de Tracklytics library. Tenslotte wordt er naar een uitbreiding gekeken die belangrijk kan zijn om developers te helpen met het onderhouden van de library.

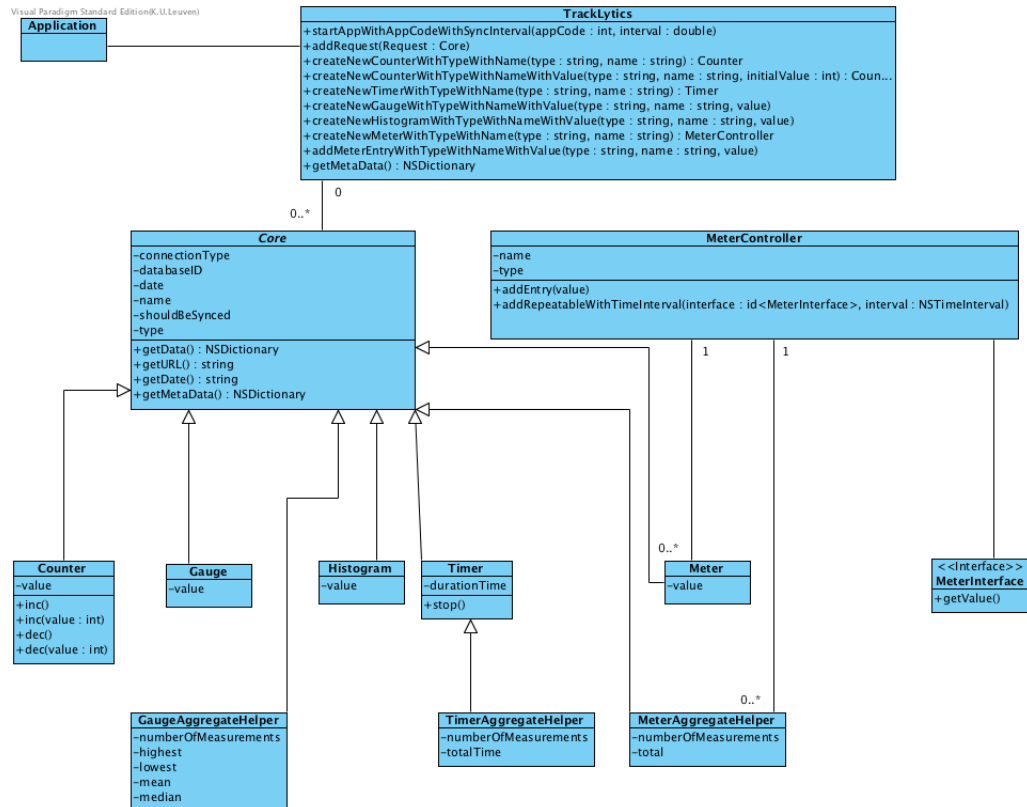
5.1 Details Implementatie

In deze sectie wordt er besproken welke technologieën er gebruikt zijn bij het ontwikkelen van de Tracklytics library.

5.1.1 iOS Library

De Tracklytics library beschreven in deze thesis is ontwikkeld voor het iOS besturingssysteem. Voor het iOS besturingssysteem bestaan er twee programmeertalen om een applicatie, of in dit geval een library, te ontwikkelen, namelijk: Swift[11] en Objective-C [36]. Swift is een redelijk recente taal, op het moment van schrijven is deze nog geen twee jaar oud. De beide talen hebben evenveel functionaliteit, enkel de syntax verschilt hierbij en de prestaties van de taal. De mobile developer community biedt de meeste ondersteuning in Objective-C [27]. Dit is de hoofdreden dat er voor Objective-C gekozen is bij het ontwikkelen van de Tracklytics library.

5. IMPLEMENTATIE



FIGUUR 5.1: Klassediagram Tracklytics library.

5.1.2 Klassediagram

Het klassediagram duidt de klassen aan waaruit de Tracklytics library bestaat. De verantwoordelijkheden van de verschillende klassen worden uitgelegd in de volgende secties.

TrackLytics

De Tracklytics klasse is de klasse die de synchronisatie met de server verzorgt. De klasse is ook verantwoordelijk voor het creëren en het eventueel opslaan van de meetobjecten. De methodes van de Tracklytics library die gebruikt kunnen worden zijn statische methodes. De keuze hiervoor is gebaseerd op twee redenen.

De Tracklytics library klasse die de methodes aanbiedt is geen objectgerichte klasse. Indien er een object van deze klasse zou gemaakt worden, zou deze elke keer na creatie bijna onmiddellijk niet meer gebruikt worden. Deze creatie van het object zorgt voor een overhead, de welke is weggewerkt door het statisch maken van alle methodes die bruikbaar zijn door de developers.

Een tweede reden is de gebruiksvriendelijkheid. Men hoeft niet steeds opnieuw een Tracklytics object aan te maken om te kunnen monitoren. Het creëren van een

meetpunt kan hierdoor op één lijn code gebeuren.

De Tracklytics klasse haalt de informatie over de applicatie uit de back end op aan de hand van de appCode die meegegeven moet worden bij het opstarten van de applicatie.

Core

Om de gemeenschappelijke elementen (de metadata) die er zijn per meting te combineren is ervoor gekozen om die gemeenschappelijke elementen in een abstracte superklasse te steken. De gemeenschappelijke methodes worden ook in de Core klasse gestoken, zodat de subklassen indien nodig dit kunnen overriden.

Counter

De counter klasse stelt een telobject voor. Je kan bij een counter een getal optellen en aftrekken. De `inc()` functie verhoogt de waarde van de counter met 1 terwijl `inc(x)` de waarde van de counter verhoogt met `x`. Hetzelfde geldt voor `dec` dat de waarde van de counter gaat verlagen.

Gauge

De gauge klasse stelt een object met één enkele waarde voor. Dit object wordt aangemaakt en eventueel opgeslagen met die waarde en gesynchroniseerd naar de server.

Histogram

De gauge klasse en het histogram zijn maar op 1 punt verschillend en dat is in de back-end. De waarde van het histogram wordt ergens anders opgeslagen dan de waarde van de gauge. In de Tracklytics library hebben ze dezelfde functionaliteit, enkel de URL naar waar de data verstuurd wordt verschilt. Dit zorgt ervoor dat in de back end de data op een andere plaats opgeslagen wordt en het dashboard zo deze data kan onderscheiden.

Timer

De timer klasse kan gebruikt worden om de lengte (in tijd) van een gebeurtenis te meten. Bij het aanmaken van de timer wordt de huidige timestamp bijgehouden. De `stop()` methode neemt de huidige timestamp en vergelijkt die met degene die werd bijgehouden. Dit resultaat is de duur van het evenement. Indien de programmeur de methode `stop()` nooit aanroept wordt de timer niet gesynchroniseerd naar de server zodat er geen foute data tussen de data in de database staat.

Meter

Een meter is bedoeld om een reeks van waarden te kunnen meten. Om dit voor de developer makkelijk te maken zijn er 3 componenten uitgedacht: de Meter, de MeterController en de MeterInterface. De meter component is een uitzondering als er gekeken wordt naar het flow diagram uit de vorige sectie 4.4. In plaats van dat een Meter object wordt terug gegeven, wordt er een MeterController object terug gegeven. Een MeterController beheert het collecteren van de data.

Meter Het meter object is het object dat opgeslagen wordt en dat gesynchroniseerd wordt naar de server. Het object houdt de waarde bij en ook het tijdstip van aanmaken. Per meting dat de meter in de applicatie doet moet er een Meter object aangemaakt worden. Dit is een van de redenen dat de MeterController bestaat.

MeterController De metercontroller zorgt ervoor dat nieuwe meters aangemaakt kunnen worden. De metercontroller bevat de naam en het type van de meter. De keuze om de meters op deze manier aan te maken berust zich op het feit dat zo niet telkens opnieuw het type van de meter moet meegegeven worden, omdat de metercontroller dit al doet en ook om de data periodiek op te kunnen halen. De metercontroller biedt een mogelijkheid aan om via de MeterInterface 5.1.2 periodiek een waarde op te halen. Deze functionaliteit zorgt ervoor dat de developer niet telkens na een bepaald interval zelf de metercontroller aan moet roepen, maar dat dit automatisch gebeurt.

MeterInterface De meterinterface wordt gebruikt om automatisch een waarde op te halen. De metercontroller roept de enige methode die deze interface aanbiedt (getValue()) aan elke keer een gegeven tijdsinterval voorbij is. De developer moet deze methode implementeren in de klasse waar deze data gecollecteerd moet worden.

Aggregatie

Zoals besproken in de architectuur 4.4 is één van de uitbreidingen van de monitoring library de developer de keuze te geven tussen het aggregeren van de data in de back end of op het toestel van de gebruiker zelf. Om deze uitbreiding te ondersteunen is ervoor gekozen om voor drie types meetobjecten de mogelijkheid tot aggregatie toe te voegen aan de Tracklytics library, namelijk: *Gauge*, *Meter* en *Timer*. De *Counter* en *Histogram* worden niet geaggregeerd, omdat er normaal gezien maar per type één counter bestaat en omdat een histogram alle waardes nodig heeft om een model op te stellen. Er is geen verschil tussen de code die de developer moet implementeren in de applicatie om van deze functionaliteit gebruik te kunnen maken.

De GaugeAggregateHelper stelt de klasse voor die de waarden van de Gauges verwerkt op het toestel. In plaats van dat er telkens een nieuwe Gauge aangemaakt

wordt, worden de volgende waarden berekend: *hoogste waarde*, *laagste waarde*, *gemiddelde en mediaan*. Het totaal aantal metingen wordt telkens met één verhoogd. Deze functionaliteit is geïmplementeerd in de Tracklytics klasse bij de *createNewGauge* methode.

De MeterAggregateHelper collecteert de waardes die in een Meter object zouden geplaatst worden. In plaats van het aanmaken van een nieuwe Meter wordt het totaal verhoogd met de gemeten waarde en wordt het totaal aantal metingen met één verhoogd. Dit heeft zeker geen invloed op de code die de developer moet schrijven, omdat de MeterController het aanmaken van de Meters verzorgt.

De TimerAggregateHelper klasse is iets complexer. De developer moet een Timer object terug krijgen om de *stop* methode te kunnen aanroepen. Dit is de reden dat de TimerAggregateHelper als superklasse de Timer heeft. Zo moet de developer zijn code niet aanpassen om deze functionaliteit te gebruiken. De Tracklytics klasse zorgt bij het aanroepen van de *createTimer* methode dat de huidige tijd wordt gezet in de TimerAggregateHelper. Indien de *stop* methode wordt aangeroepen, dan berekent de TimerAggregateHelper de verstreken tijd, telt deze op bij de totale tijd en verhoogt het totaal aantal metingen met één.

5.1.3 Back end

De data die de Tracklytics library doorstuurt vanaf het toestel van de gebruiker moet opgeslagen worden in een database. Zo kan deze data later verwerkt worden en worden weergegeven in het dashboard. Er moet een keuze gemaakt worden over het besturingssysteem, de database en de programmeertaal van de back end.

De server draait in OpenStack, een cloud platform. Deze is gedeployed als infrastructure-as-a-service (IaaS). Dit wil zeggen dat dit meerdere virtuele servers kan aanbieden (zelfs meerdere virtuele servers als fysieke servers). Dit biedt een abstractie en schermt de virtuele server af van andere virtuele servers. Een gebruiker kan zelf nieuwe virtuele servers aanmaken. Elke virtuele server heeft zijn eigen besturingssysteem, te kiezen uit een lijst van images aangeboden door OpenStack[5].

In de Tracklytics architectuur is gekozen voor een Linux distributie (in dit geval Ubuntu [7]). Deze keuze is gemaakt op basis van de gebruiksvriendelijkheid van dit besturingssysteem. Zo is het gemakkelijk om een webserver op te zetten en een database te installeren. Er is voor Ubuntu gekozen, omdat dit de meest gekende en meest gebruikte linux distributie is.

In de database wordt alle data opgeslagen, dat wil zeggen dat dit een van de meest belangrijke onderdelen van de Tracklytics architectuur vormt. De metadata die gecollecteerd wordt per applicatie is in vele gevallen hetzelfde. De parameters die verschillen zijn: het type toestel, het type internet connectie en de versie van de applicatie. Dit gegeven zorgt ervoor dat de metadata vaak hetzelfde is. Indien de metadata uit de data komende van de Tracklytics library wordt uitgehaald kan er

relatief veel opslagruimte gespaard worden, omdat deze metadata niet in elke entry in de database aanwezig moet zijn, enkel een verwijzing naar waar die metadata staat. Er is voor gekozen om een relationele database te kiezen, namelijk MySQL [30]. De gebruikte tabellen zijn door SQL queries gecombineerd worden in views om de data overzichtelijk te maken.

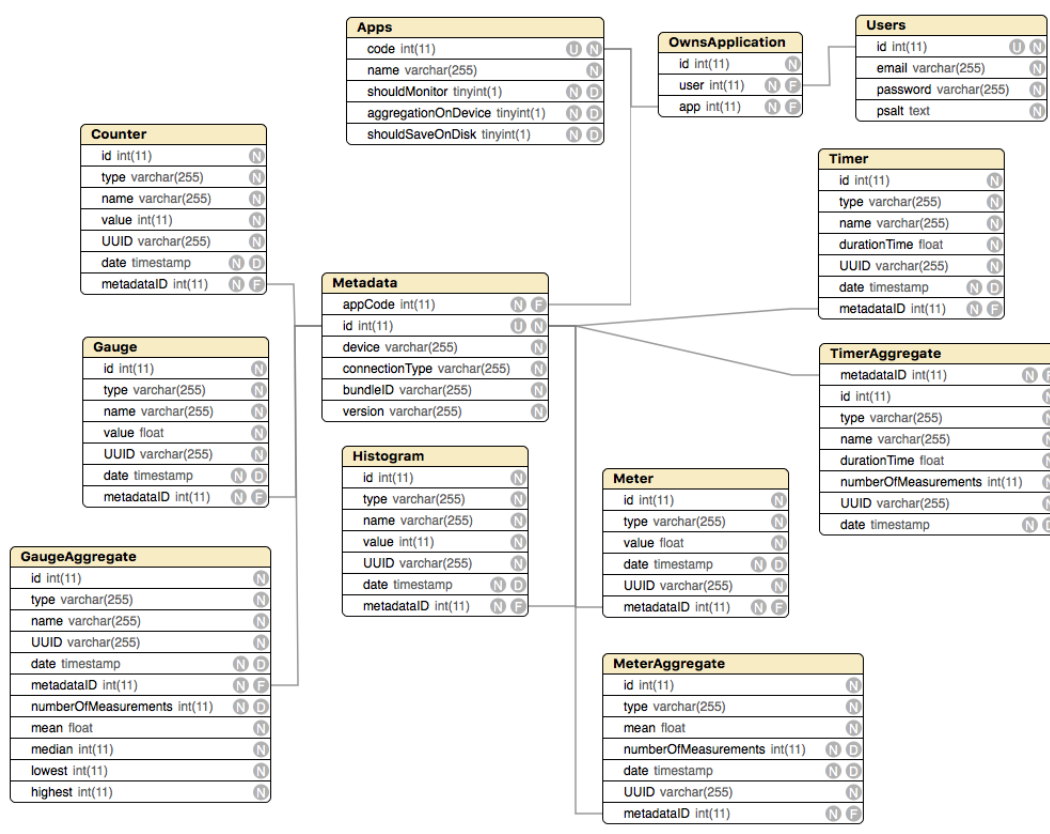
De back end heeft in de Tracklytics library twee functies, namelijk: het verwerken en opslaan van de data in de database en het opvragen van data uit de database om het dashboard van de data te voorzien. De keuze is gevallen op PHP [6] als programmeertaal. Met PHP is het eenvoudig om een database connectie op te zetten en via SQL queries deze data in of uit de database te krijgen. Een tweede voordeel van PHP is dat deze met POST data om kan. Op deze manier kan de data in de request van de Tracklytics library verborgen worden in de request en moet deze niet rechtstreeks doorgegeven worden in bijvoorbeeld de URL zelf. Zo is er meer veiligheid en privacy van de data.

5.1.4 Structuur database

In bovenstaande figuur 5.2 is de structuur van de Tracklytics library getoond. Voor elk type meetobject is er een tabel aangemaakt waarin de gegevens opgeslagen worden. Voor elk type geaggregeerd object is er een aparte tabel aangemaakt omdat deze een andere structuur nodig hebben dan de originele objecten. De metadata wordt opgeslagen in een aparte tabel en vanuit elk type meetobject wordt hier naar gelinkt met behulp van de metadataID. Elke applicatie die de Tracklytics library gebruikt heeft een entry in de Apps tabel om deze applicatie te identificeren. De OwnsApplication en User tabel worden gebruikt door het dashboard om de developers die het dashboard gebruiken te linken aan de applicaties die ze geregistreerd hebben.

De Metadata tabel bevat per applicatie de metadata die gelinkt wordt aan de entries in een tabel van de meetobjecten. Hierin worden de volgende eigenschappen gebundeld: een link naar de applicatie waar deze entry bij hoort, het type toestel, welke soort netwerkverbinding de gebruiker had, de versie van de applicatie en het bundleID van de applicatie, wat een identificatie van die bepaalde applicatie voorstelt vanuit het iOS systeem. Deze tabel verkleint de tabellen van de meetobjecten, omdat er enkel een link naar een entry in deze tabel moet staan en niet alle metadata bij in de tabel van het meetobject moet staan.

De tabellen van de meetobjecten bevatten de data die door de library gecollecteerd zijn. Zoals hierboven vermeld hebben de entries in de tabel een link naar een entry in de metadata tabel. De meetobjecten en de metadata worden gecombineerd in een view per type meetobject om zo per meting te kunnen zien welke



FIGUUR 5.2: Structuur van de Tracklytics database.

metadata de meting heeft. Dit is eenvoudiger voor het ophalen van de data voor het dashboard.

5.1.5 Structuur back end

De back end is opgedeeld in twee delen, namelijk een deel waarmee de mobiele library in contact staat en een deel waarmee het dashboard in contact staat. Het doel van het eerste deel is om de data die van de mobiele library komt te verwerken en op te slaan in de database. Het doel van het tweede deel is om de gegevens uit de database op te halen en door te geven aan het dashboard. Om de data die van de mobiele library komt op te slaan in de database is ervoor gekozen om per type meetobject een apart PHP bestand aan te maken die de data van dat type meting te verwerken en op te slaan in de database. Het dashboard haalt de data die hij nodig heeft op via specifieke PHP bestanden ontwikkeld om het dashboard van data te voorzien, zoals bijvoorbeeld het totaal aantal counts.

5.1.6 Dashboard

Het Tracklytics dashboard is ontworpen om de gegevens van de applicatie in grafieken en details weer te geven om hieruit een conclusie te kunnen trekken over het functioneren van de applicatie. Het dashboard is ontworpen als webapplicatie in plaats van een desktop applicatie. De voordelen hiervan zijn: software updates worden automatisch doorgevoerd zonder dat er tussenkomst van de gebruiker nodig is, het is cross platform, omdat het in de browser draait en er is geen installatie vereist.

Om de webapplicatie te ontwikkelen werd er gebruik gemaakt van AngularJS [21]. De reden hiervoor is dat dit een zeer handig framework is voor het ontwikkelen van dynamische websites. Dit framework bindt stukken HTML code aan JavaScript code, wat ervoor zorgt dat het DOM gemakkelijk manipuleerbaar is door JavaScript. Met AngularJS is het eenvoudig om grafieken en gegevens die getoond moeten worden meermaals te herhalen met andere gegevens in. Zo kunnen de verschillende grafieken onder elkaar geplaatst worden zonder telkens de HTML code in JavaScript aan te moeten passen. Een bijkomstig voordeel is dat de data in de verschillende tabbladen maar eenmalig ingeladen moet worden, omdat AngularJS ervoor zorgt dat als je op een tab drukt niet de hele webpagina opnieuw wordt ingeladen, maar enkel de view die verandert ingeladen wordt.

Om de grafieken weer te kunnen geven, is ervoor gekozen om Chart.js[2] (voor AngularJS) te gebruiken. Dit framework neemt data die in AngularJS variabelen gezet worden en geeft deze weer in een gekozen grafiek (bv. lijn-grafiek of staafdiagram). Chart.js is een simpele manier om snel een grafiek weer te kunnen geven, het werkt out-of-the-box en er zijn een aantal zeer handige opties die kunnen aangepast worden.

Om de histogrammen en de meters bruikbaar te maken werden hier sliders (5.5, 5.6) bij toegevoegd om bij de histogrammen het interval te veranderen en zo een kleinere dataset te gebruiken. Bij de meters kan de dataset verkleind worden door het tijdsinterval te wijzigen. Een slider boven de grafieken zorgt ervoor dat de dataset verkleind kan worden. Deze slider is geïmplementeerd met behulp van de AngularJS-slider [1].

De grafieken worden per soort meetobject weergegeven, er bestaat dus een aparte pagina voor de counters, de gauges, de timers, de meters en de histogrammen. Zo heeft een developer een overzicht per type meting. De grafieken die weergegeven worden zijn basisgrafieken, deze zijn enkel opgedeeld door de namen die aan de metingen gegeven zijn. De developer kan hier enkel basisinformatie uithalen. Om een dieper detailniveau te verkrijgen is ervoor gekozen om per grafiek een gedetailleerdere pagina weer te geven. De grafiek wordt dan opgesplitst door verschillende eigenschappen te gebruiken. De volgende eigenschappen worden gebruikt om de grafieken op te delen: het type toestel, de versie van de applicatie en het type connectie. Deze worden onderling nog gecombineerd om nog extra in detail de gemeten data te kunnen bekijken. Aan de hand hiervan kunnen developers een concreet overzicht krijgen over de prestaties en het gebruik van de applicatie.

De counters 5.3 worden weergegeven in een staafdiagram, zodat er snel gezien kan worden welke counter het meeste tellingen heeft. In de detailpagina wordt er opgesplitst per toestel en per versie en deze worden met elkaar gecombineerd. Zo kan de developer een beter overzicht krijgen naar de tellingen.

De timers 5.7 worden als gemiddelde weergegeven, niet in een grafiek maar als tekst. Zo kan de developer snel kijken wat de gemiddelde duur is. In de detailpagina worden deze metingen opgesplitst op welke internetverbinding het toestel had, welk toestel het was waarop gemeten werd en welke versie. Deze worden onderling gecombineerd om een gedetailleerder beeld te geven aan de developers.

De gauges 5.4 worden weergegeven als de gemiddelde waarde die gemeten is. Zo wordt er een snel overzicht gegeven aan de developers. Developers kunnen deze data dieper bekijken op de detailpagina. Hier worden de volgende overzichten aangeboden: het gemiddelde, de mediaan, de laagste en de hoogste waarde.

De histogrammen 5.5 worden weergegeven als histogrammen, in een staafdiagram met daaronder de gegevens die belangrijk zijn over het histogram, namelijk: de mediaan, het gemiddelde, standaarddeviatie. Er zijn nog enkele andere gegevens bijgevoegd die belangrijk kunnen zijn, namelijk: het verschil tussen de grootste en de kleinste waarde, het percentage lager dan het gemiddelde en het percentage hoger dan het gemiddelde. In de detailpagina wordt het histogram opgedeeld op basis van het toestel en/of de versie om een gedetailleerder beeld te krijgen. Er is voor gekozen om een slider toe te voegen aan de histogrammen om het bereik te kunnen verkleinen. Er is de mogelijkheid om de maximum- en minimumwaarde aan te geven met behulp van deze slider.

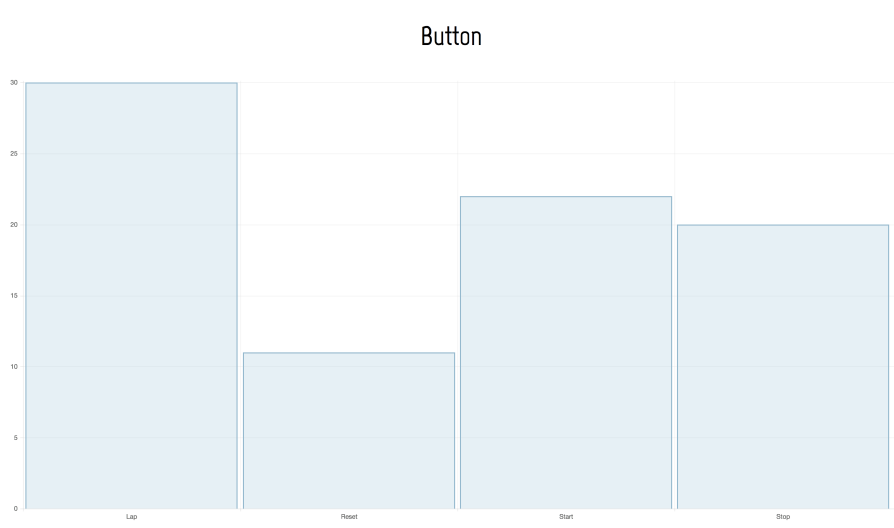
De meters 5.6 worden weergegeven als een lijngrafiek met daarop het gemiddelde per dag. Er is een slider voorzien om de developer de kans te geven de dataset te verkleinen door het bereik aan te passen en zo aan te geven van welke dagen hij het overzicht wil zien. Indien de dataset minder dan drie data bevat is er voor gekozen om de data per drie uren weer te geven, zodat er meer op detail in kan gegaan worden. In de detailpagina wordt deze data opgesplitst op basis van het type toestel en/of de versie waarop gemeten is.

In het dashboard is ruimte voorzien om instellingen van de applicatie op afstand te wijzigen 5.9. De developer kan kiezen of de applicatie gemonitord moet worden, of dat de aggregatie op het toestel moet gebeuren of in de backend en hij kan kiezen

of de data tijdelijk op de harde schijf opgeslagen moet worden of niet. Deze remote functies zorgen ervoor dat de developer niet telkens een nieuwe versie moet uitrollen om deze instellingen te wijzigen.

De data die nodig is om de titels en de gegevens voor de grafieken in te laden moet uit de backend gehaald worden. Deze is zoals in vorige sectie aangehaald geschreven in PHP.

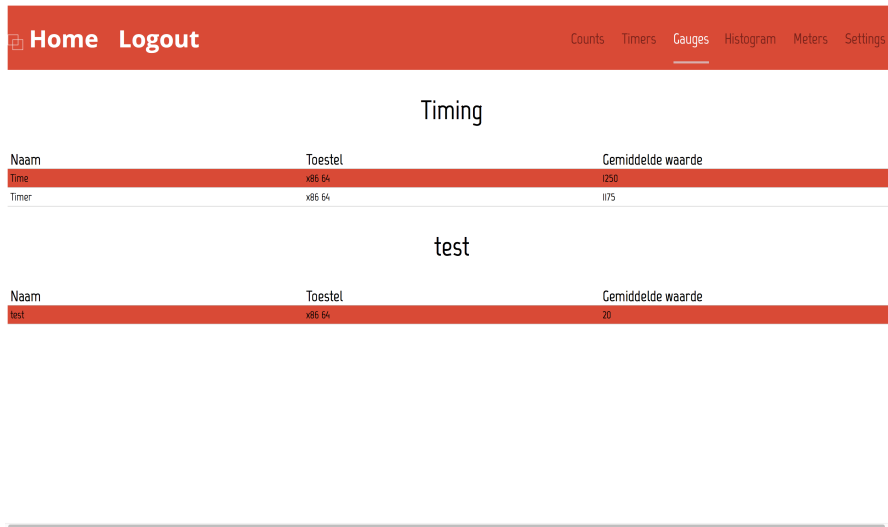
Er is enorm veel werk gekropen in het ontwikkelen van dit dashboard. Dit omdat het dashboard from scratch ontwikkeld is. De reden hiervoor is dat de bestaande dashboards die vrij te gebruiken zijn geen oplossing gaven voor de visualisatie die ik wou geven. Het ontwikkelen van het dashboard (inclusief het opzetten van alle achterliggende systemen zoals de database en dergelijke) in combinatie met het ontwikkelen en het tweaken van de library heeft meer dan 200 uur tijd in beslag genomen.



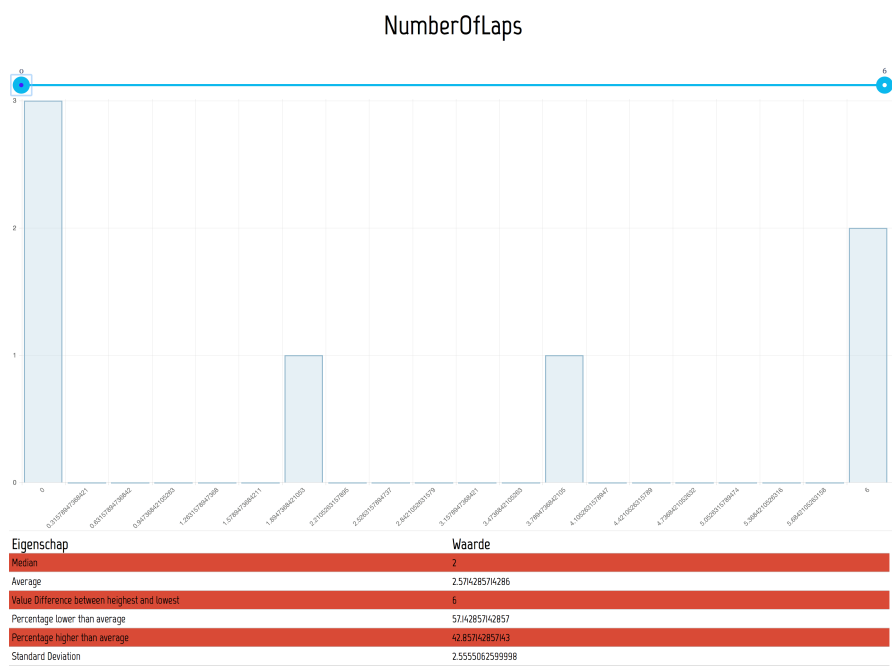
FIGUUR 5.3: Dashboard: Counter overzicht.

5.1.7 Connectie tussen Front end en Back end

Zoals eerder aangegeven stuurt de front end (de Tracklytics iOS library) data naar de back end over een internetverbinding. Om ervoor te zorgen dat deze data veilig overgedragen wordt is ervoor gekomen om een HTTPS verbinding te gebruiken. Dit zorgt er automatisch voor dat er een veilige verbinding tussen client en server is. Een alternatief is de data zelf encoderen aan de client zijde en decoderen aan de server zijde. Omdat HTTPS ervoor zorgt dat de data veilig wordt overgedragen is ervoor gekozen om encoderen en decoderen niet te gebruiken. Een andere beveiligingskeuze die er is gemaakt is om HTTP POST te gebruiken in plaats van HTTP GET. HTTP POST verbergt de data in de request, terwijl HTTP GET de data in de URL van de

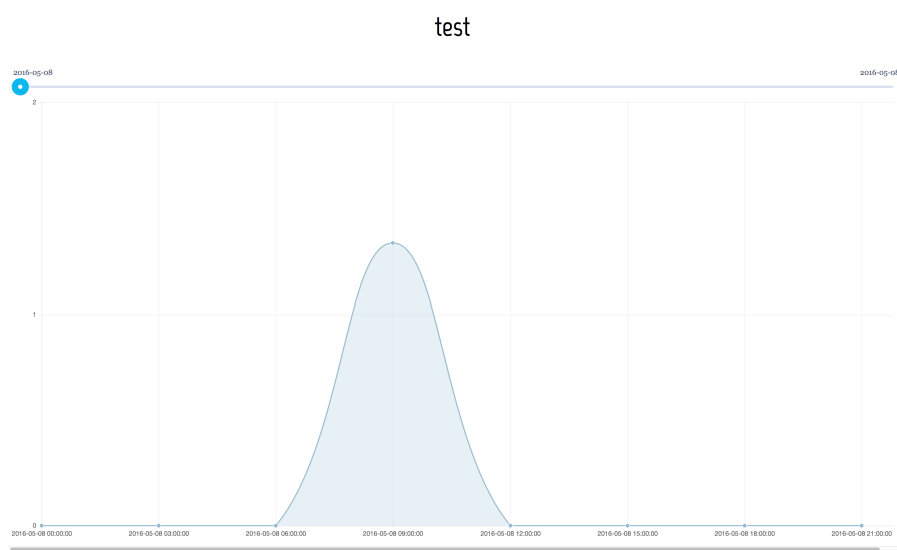


FIGUUR 5.4: Dashboard: Gauge overzicht.

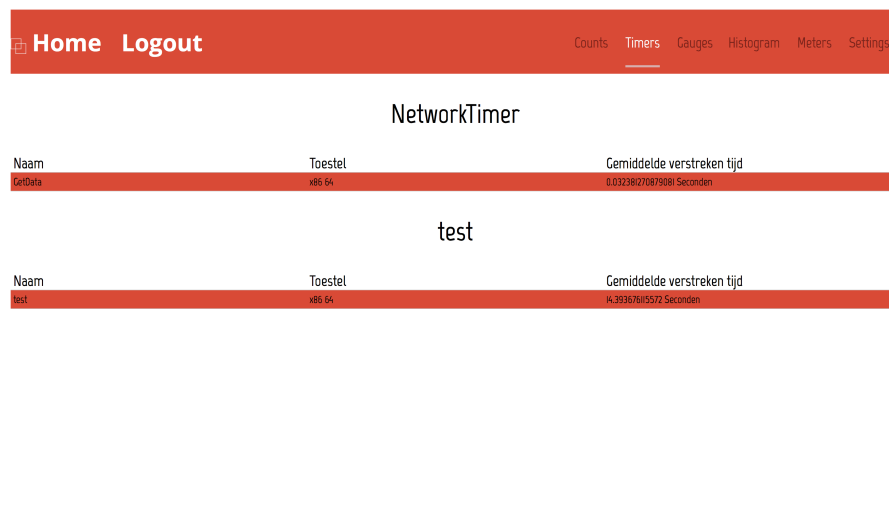


FIGUUR 5.5: Dashboard: Histogram overzicht.

request zet. De consequenties hiervan zijn dat de data in HTTP POST requests in geen enkele log of geschiedenis voorkomen, wat wel kan gebeuren met HTTP GET requests. Het is ook moeilijker de data in de HTTP POST requests aan te passen dan die van de HTTP GET requests, omdat het gemakkelijker is om een URL aan te passen dan een request zelf.



FIGUUR 5.6: Dashboard: Meter overzicht.



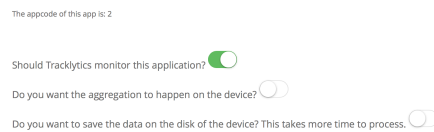
FIGUUR 5.7: Dashboard: Timer overzicht.

5.2 Aggregatie

De data die de Tracklytics library opgehaald heeft bij het monitoren van de applicatie moet geaggregeerd worden om weer te kunnen geven in het dashboard. De logische plaats om deze aggregatie uit te voeren is de back end, omdat deze elk meetpunt in de database heeft staan. Het voordeel hieraan is dat de data opgehaald kan



FIGUUR 5.8: Dashboard: Timer overzicht per versie.



FIGUUR 5.9: Dashboard: Instellingen overzicht.

worden en enkel verwerkt wordt wanneer deze nodig is. Een nadeel hieraan is dat dit zeer veel CPU kan kosten en zo het dashboard enorm traag maakt. Dit nadeel is gelinkt aan de hoeveelheid gebruikers van de applicatie en het aantal meetpunten in de applicatie. Groeit een van beide of beiden, dan treedt er een extra vertraging op in het aggregeren van de gegevens. Dit probleem kan opgelost worden door op voorhand te aggregeren, dit kan op twee manieren: **op het toestel zelf** en **op periodieke intervallen**. Deze sectie berust zich op het feit dat I/O operaties veel trager zijn dan verwerkingsoperaties.

De eerste oplossing is om de aggregatie uit te voeren op het toestel zelf. De Tracklytics library kan aangepast worden om in plaats van de data van de meetpunten zelf door te sturen, eerst deze te verwerken op het toestel zelf en die verwerkte data door te sturen naar de server. De back end zal ook moeten aangepast worden om deze data op te slaan zodat deze herbruikt kan worden. Het voordeel hieraan is dat als het dashboard de data opvraagt van de back end, de back end dit sneller zal kunnen doen, omdat er minder data uit de database opgehaald en verwerkt moet worden. Zo is het dashboard bruikbaar bij grotere schaal. Een nadeel hieraan is dat de gegevens niet meer optimaal zijn. Bijvoorbeeld: de mediaan kan enkel nog benaderd worden, omdat niet alle waarden aanwezig zijn in de database. Door afrondingsfouten kunnen de gemiddeldes verschillen van de werkelijke gemiddeldes.

Een tweede oplossing is om periodiek (bv. één keer per dag) per applicatie de nieuwe data te gaan aggregeren en samenvoegen met de al geaggregeerde data van de voorbije tijdstippen. Deze functionaliteit moet op de servers geïmplementeerd

worden. Het voordeel hieraan is dat indien het dashboard de data opvraagt, enkel de nieuwe data nog moet samengevoegd worden met de al geaggregeerde data. Dit zorgt voor een enorme performance boost van het dashboard. Het nadeel hieraan is dat ofwel er een overzicht moet bijgehouden worden welke data al geaggregeerd is en welke niet, ofwel de verwerkte data telkens uit de database verwijderd moet worden om delen van de dataset niet opnieuw te verwerken. Dit vraagt dus een extra inspanning om te implementeren.

Een derde oplossing is om de twee te combineren. Op het toestel wordt een deel van de aggregatie al gedaan en deze geaggregeerde data wordt dan verwerkt op de servers zoals in het vorige deel. Dit geeft een extra prestatiewinst, de verwerking van de geaggregeerde data op de servers gaat sneller, omdat deze al geaggregeerd is. Er is dus minder CPU-tijd nodig om de opdracht af te werken. Indien het dashboard de data opvraagt aan de back end gaat dit ook veel sneller. De combinatie van de nieuwe data en geaggregeerde data gaat veel sneller, omdat er minder data aanwezig is.

Een laatste oplossing is om de data te aggregeren wanneer het echt nodig is en deze dan op te slaan. Als de developer of eigenaar van de applicatie het dashboard voor de eerste keer opent, dan wordt deze data geaggregeerd en opgeslagen zoals bij de tweede oplossing. Als de developer of eigenaar het dashboard de volgende keer opent moet enkel de nieuwe data geaggregeerd worden en gecombineerd met de al geaggregeerde data. Deze combinatie wordt dan opnieuw opgeslagen. Dit proces wordt herhaald telkens dat het dashboard van die applicatie geopend wordt. Het voordeel hieraan is dat dit sneller is dan telkens opnieuw alle data te aggregeren. Het nadeel hieraan is dat er moet bijgehouden worden welke data verwerkt is en welke niet.

In de Tracklytics library is ervoor gekozen om de developer de keuze aan te bieden tussen het aggregeren van de data in de back end en het aggregeren van de data op het toestel. De Tracklytics library aggregiert data van de volgende meetobjecten op het toestel: **Gauge, Meter, Timer**. De developer kan deze keuze maken in de instellingen die het dashboard aanbiedt.

Het Gauge meetobject wordt geaggregeerd op het toestel door tijdens het aggregeren *het gemiddelde, de mediaan, de laagste en de hoogste waarde* te berekenen op basis van de waarden die gemeten worden. In plaats van telkens een nieuwe Gauge waarde aan te maken wordt deze toegevoegd aan een helper object die deze waarden bijhoudt om de mediaan te berekenen en bij het toevoegen van een waarde ineens het gemiddelde berekent en bekijkt of deze waarde de hoogste of laagste waarde is. Het grootste nadeel hieraan is dat de mediaan in de back end enkel nog benaderd kan worden omdat niet alle waarden nog in de back end aanwezig zijn.

Het Meter meetobject wordt als volgt geaggregeerd: in plaats van elke waarde apart op te slaan wordt het aantal metingen bijgehouden en wordt er een gemiddelde bij gehouden. Deze waardes kunnen in de back end gecombineerd worden om een globaal overzicht te krijgen. Het nadeel hieraan is dat er afrondingsfouten kunnen gebeuren bij het berekenen van het gemiddelde en er zo een verkeerde observatie gemaakt kan worden. Ook wordt er niet per meting bijgehouden op welk tijdstip deze voorvalt.

Het Timer meetobject wordt geaggregeerd door de gemiddelde tijd te berekenen bij het stoppen van een timer en bij te houden hoeveel metingen er voorgekomen zijn. Het nadeel is dat er per timer maar één instantie tegelijkertijd kan lopen omdat Tracklytics dezelfde methodes aanbiedt voor een geaggregeerd object en een niet-geaggregeerd object. Net zoals bij het *Meter meetobject* kunnen er hier afrondingsfouten voorkomen.

5.3 Openstaande uitdagingen

De implementatie gegeven in deze thesis dekt niet de volledige architectuur uit de vorige sectie. Er blijven dus nog steeds uitdagingen om de library uit te breiden zodat deze dichter aanleunt bij de architectuur. Er worden ook andere uitbreidingen beschreven die niet in de architectuur beschreven worden, maar van even groot belang zijn dan degene die wel in de architectuur beschreven wordt.

5.3.1 AB Testing

AB testing wordt gebruikt om geleidelijk aan een nieuwe versie van een applicatie of website uit te rollen, zoals beschreven in de architectuur sectie. Het is een uitdaging om dit concept te combineren met de Tracklytics library. Er moet onderzocht worden wat de efficiëntste manier is om AB testing in mobiele applicaties mogelijk te maken. Een manier om AB testing te gebruiken is om eerst het versienummer voor die gebruiker op te halen in de database. Een niet efficiënte manier zou dan zijn om per versie een if statement te gebruiken om zo de verschillende karakteristieken van de versie te bepalen. Deze manier is op drie manieren inefficiënt, namelijk: de code van de applicatie (en dus ook de grootte op schijf) wordt groter naarmate er meer versies worden gebruikt, door de if statements wordt de applicatie trager en als er een aanpassing moet gebeuren aan een versie of er wordt een versie toegevoegd moet deze nog steeds eerst via de app store als update uitgevoerd worden.

5.3.2 Operating systems

De Tracklytics library is ontwikkeld voor iOS, het besturingssysteem van Apple. Er bestaat buiten iOS nog een besturingssysteem dat de moeite waard is om te bekijken, namelijk Android. Op het moment van schrijven is Android de marktleider

met een marktaandeel van 59.65% in de mobiele markt, gevolgd door iOS (32.28%) [28]. De andere mobiele besturingssystemen zijn niet de moeite waard om te bekijken, omdat deze een bijna verwaarloosbare marktaandeel hebben. De toestellen die Android gebruiken zijn verdeeld over vele versies. Op moment van schrijven draait ongeveer een derde van de toestellen op een twee jaar oud besturingssysteem (KitKat 4.4) [23]. In contrast, 84% van de toestellen die op iOS draaien hebben de laatste versie geïnstalleerd (op moment van schrijven) [9]. Elke nieuwe versie brengt veranderingen en vernieuwingen met zich mee, wat er voor zorgt dat sommige taken sneller/trager uitgevoerd worden in de ene versie dan in de andere versie. Dit heeft mede de keuze voor iOS gemaakt.

De uitdaging die zich hier voordoet is het schrijven van een library voor Android. Dezelfde architectuur kan worden gebruikt voor de library. Er moeten enkele aanpassingen gedaan worden om dit te doen werken. Allereerst moet er een onderscheid gemaakt kunnen worden tussen iOS en Android data. Door telkens het type besturingssysteem door te sturen als metadata is dit mogelijk om het zo in de database te stoppen. Anderzijds is het noodzakelijk in het Android geval om de versie van het besturingssysteem mee door te sturen, zodat er in het dashboard een duidelijke opdeling hiertussen kan gemaakt worden. Dit zorgt er dan weer voor dat het dashboard complexer wordt om te implementeren.

5.3.3 Alarmen

Om het concept en nut van alarmen uit te leggen is het handig om met een voorbeeld te beginnen.

Een eigenaar heeft een applicatie ontwikkeld die communiceert met de back end. De eigenaar heeft ervoor gekozen om servers te huren in plaats van zijn eigen servers te bouwen en onderhouden. De eigenaar wil dat er zo weinig mogelijk servers gehuurd moeten worden en dat de gebruiker van de applicatie geen zichtbare vertraging heeft (er toch genoeg servers zijn om de requests te behandelen). Zo kunnen de kosten geminimaliseerd worden en dus de winst gemaximaliseerd. Developers moeten dus servers toevoegen als de servers de requests niet meer aan kunnen en een server verwijderen indien die server overbodig wordt om het werk gedaan te krijgen.

Om dit mogelijk te maken moet er een trigger of alarm komen die de developers waarschuwt indien de huidige servers het werk niet meer snel genoeg kunnen doen. Dit alarm kan geïmplementeerd worden in de servers of in de Tracklytics library.

Er bestaan al meerdere oplossingen die deze functionaliteit aanbieden op het vlak van servers. Hier wordt niet verder op ingekeken omdat dit niet in de context van deze thesis past.

In de Tracklytics library kan er een alarm functionaliteit aangeboden worden. Een developer moet dan in het dashboard aangeven welke parameter welke waarde maximum mag krijgen. Het is ook handig dat de developer een maximum aantal schendingen van deze parameter kan aangeven en dat hij enkele types van metadata

kan uitsluiten. (bv. de tijd dat het kost om data van de back end op te halen mag maximum bij 5 verschillende gebruikers boven de 10 seconden liggen op WiFi en 4G). De developer moet ook de tijdspanne aanduiden waarin deze parameters niet overschreden mogen worden. Indien deze drempel overschreden wordt, moet er een waarschuwing gestuurd worden naar de developer. Hierdoor moeten de drie componenten samenwerken (Dashboard, Back end en mobiele library). Het dashboard moet de gegeven paramaters doorgeven aan de back end zodat deze opgeslagen kunnen worden in de database. De mobiele library moet, via de back end, de parameters ophalen. Indien er dan een match is wanneer er nieuwe gegevens worden doorgegeven via de applicatie, dan moet de library nagaan of deze de drempel niet overschrijden. Indien de drempel wordt overschreden, dan moet dit gemeld worden aan de back end. De Tracklytics library moet dan een call doen naar de back end. De back end slaat deze melding op in de database en gaat na of er een alarm verstuurd moet worden. De back end kijkt dus of er in de gegeven tijdspanne meer schendingen geweest zijn dan dat de developer als maximum aangaf. In plaats van elke keer dat er een schending is na te gaan of er een alarm verstuurd moet worden, kan de back end periodiek nagaan of de parameters overschreden worden. Het voordeel hieraan is dat de back end sneller is, omdat de controle niet telkens moet uitgevoerd worden. Het nadeel is dat het alarm bijna een volledige periode later uitgezonden wordt indien de schending van de parameters aan het begin van de periode plaatsvindt.

De alarm functionaliteit kan ervoor zorgen dat er zo weinig mogelijk servers gebruikt worden en het geen impact heeft op de kwaliteit van de applicatie en niet zichtbaar is voor de gebruikers. Zo kunnen de kosten geminimaliseerd worden en de winst gemaximaliseerd.

Hoofdstuk 6

Tutorial

In deze sectie wordt er uit de doeken gedaan hoe je als developer de Tracklytics library in je applicatie kunt inbouwen.

6.1 Installatie

In deze sectie wordt besproken hoe de installatie van de library in de applicatie in zijn werk gaat.

Om de library snel en gemakkelijk in een applicatie in te bouwen is ervoor gekozen om CocoaPods te gebruiken [16]. Dit is de standaard methode om externe libraries te gebruiken in iOS. De volgende stappen moeten gebeuren om de Tracklytics library in te bouwen in de applicatie.

Allereerst moet er in de root folder van het project (via een terminal) van de applicatie *pod init* uitgevoerd worden. Dit maakt een PodFile bestand aan. Dit bestand moet aangepast worden, het platform moet aangegeven worden als *ios*, *7.0* en tussen *target 'AppName' do* en *end* moet de volgende tekst komen: *pod "Tracklytics"*. Cocoapods selecteert automatisch de laatste versie van de library.

Om de library dan te installeren moet in de root folder het volgende commando (via een terminal) uitgevoerd worden: *pod install*. Cocoapods installeert nu de library in de applicatie en zorgt dat alle dependencies geïnstalleerd worden. Dit commando genereert een nieuw project dat voortaan gebruikt moet worden in plaats van het andere project.

6.2 Initialisatie

De Tracklytics library moet bij het opstarten van de applicatie gestart worden om het synchronisatieproces te starten. De aanbevolen manier is om dit in de `AppDelegate` klasse te doen in de `(BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions` methode. Deze

methode wordt automatisch uitgevoerd nadat de applicatie opgestart is, in welke applicatie dan ook. In deze methode is het dus belangrijk om volgende code uit te voeren: `[Tracklytics startTrackerWithAppCode:appCode withSyncInterval:interval];`. Deze code start het synchronisatieproces van de Tracklytics library. De parameter `appCode` stelt de unieke code voor die Tracklytics gebruikt voor het identificeren van de applicatie. De interval parameter geeft de tijd (in seconden) tussen twee synchronisatiecycli weer.

6.3 Counter

Een counter kan aangemaakt worden met behulp van volgende call naar de Tracklytics library: `[Tracklytics createNewCounterWithType:type withName:name withValue:value]`, de `withValue:value` is optioneel indien er een initiële waarde in de counter moet staan. Deze methode geeft een `CounterObject` terug, wat besproken wordt in volgende sectie.

Parameters

- `type`: Het type evenement (bv. Button als er op een button geklikt wordt, vrij te kiezen door de developer). Deze parameter zorgt ervoor dat de data verdeeld wordt over de verschillende grafieken in het dashboard.
- `name`: De naam van het evenement (bv. de naam die de button heeft). Deze parameter zorgt ervoor dat de data in de grafieken verder opgedeeld worden per naam.
- `value`: De initiële waarde die de counter moet hebben wanneer die gecreëerd wordt. Deze waarde is optioneel en is standaard 0 bij creatie.

6.3.1 CounterObject

Bij de creatie van een Counter wordt er een `CounterObject` terug gegeven. Dit object kan gebruikt worden om evenementen te tellen. Een counter biedt een methode aan om de counter te verhogen: `inc` met een optionele waarde. Het object biedt ook een methode aan om de counter te verlagen: `dec`, ook met een optionele waarde. De Tracklytics library zorgt ervoor dat het `CounterObject` in sync blijft met de backend.

6.4 Gauge

Aangezien een gauge waarde niet verandert in de tijd, is het niet nodig om hier, zoals bij de counter, een object terug te geven. De waarde moet maar een keer eventueel opgeslagen en doorgezonden worden naar de server. Om de waarde te verzamelen moet men de volgende methode uitvoeren: `[Tracklytics createNewGaugeWithType:type withName:name withValue:value]`.

Parameters

- **type:** Het type evenement (bv. Search Result bij een zoekopdracht). Deze parameter zorgt ervoor dat de data verdeeld wordt in verschillende secties in het dashboard.
- **name:** De naam van het evenement (bv. het ingevulde woord/zin in de zoekopdracht). Deze parameter zorgt ervoor dat de data in de verschillende secties verder opgedeeld worden per naam.
- **value:** De waarde die gesynchroniseerd moet worden naar de server (bv. het aantal resultaten van de zoekopdracht).

6.5 Histogram

Een histogram heeft dezelfde eigenschappen als een gauge in termen van collectie van data in de Tracklytics library. Een histogram waarde verandert niet in de tijd en het is dus niet nodig om een object hiervan terug te geven. De methode die aangeroepen wordt verschilt enkel in de naam van de methode: `createNewHistogramWithType:type withName:name withValue:value`.

Parameters

- **type:** Het type evenement. Deze parameter zorgt ervoor dat de data verdeeld wordt over de verschillende grafieken in het dashboard.
- **name:** De naam van het evenement. Deze parameter zorgt ervoor dat de data in de grafieken verder opgedeeld worden per naam.
- **value:** De waarde die gesynchroniseerd moet worden naar de server.

6.6 Timer

Een timer meet de tijd dat een evenement nodig heeft. Om een timer aan te maken moet de volgende call gemaakt worden naar de Tracklytics library: `[Tracklytics createNewTimerWithType:type withName:name]`.

Parameters

- **type:** Het type evenement. Deze parameter zorgt ervoor dat de data verdeeld wordt in verschillende secties in het dashboard.
- **name:** De naam van het evenement. Deze parameter zorgt ervoor dat de data in de secties verder opgedeeld worden per naam.

Deze methode geeft een Timer object terug. Vanaf dat deze methode uitgevoerd is begint de timer te lopen. Deze blijft lopen tot de `stop` methode aangeroepen wordt op het Timer object. De timer heeft dan een exacte waarde hoeveel tijd het evenement in beslag nam. Tracklytics houdt deze waarde bij en synchroniseert deze naar de server.

6.7 Meter

Een meter is complexer dan alle andere meetobjecten in de Tracklytics library. Dit is omdat ervoor gekozen is om de mogelijkheid aan te bieden om data automatisch op een bepaald interval te collecteren. Zo moet de developer hier niet meer naar omkijken. Om dit te kunnen verwezenlijken is er een controller gemaakt, de `MeterController`, die de mogelijkheid biedt voor de automatische collectie van data en van de manuele invoer van data. Een `MeterController` can gecreëerd worden door de volgende call naar de Tracklytics library: `[Tracklytics createNewMeter:type]`.

Parameters

- `type`: Het type evenement. Deze parameter zorgt ervoor dat de data verdeeld wordt over de verschillende grafieken in het dashboard.

6.7.1 MeterController

De `metercontroller` zorgt voor het managen van de meter values. Het is mogelijk om manueel data toe te voegen of automatisch de data te laten controlleren.

Om data manueel toe te voegen moet de volgende methode uitgevoerd worden: `addEntry: (float) value`. De `value` parameter is de waarde die moet toegevoegd worden aan de meter.

Om de data automatisch te laten collecteren moet de methode `addRepeatable:(id<MeterInterface>) interface withTimeInterval:(NSTimeInterval) interval` uitgevoerd worden. Deze neemt een object dat de `MeterInterface` implementeerd als parameter en een interval.

Het `MeterInterface` object implementeerd de `(float) getValue` methode die een float waarde terug geeft die de `MeterController` kan collecteren door die methode. Het interval geeft aan (in seconden) hoeveel tijd er tussen twee cyclussen zit waarin de waarde opgehaald wordt.

Hoofdstuk 7

Evaluatie

In deze sectie wordt de ontwikkelde library geëvalueerd. Allereerst wordt er gekeken naar wat de impact van de library op de performance van de applicatie is. Verder in dit hoofdstuk wordt de schaalbaarheid van de library bekeken. Tot slot wordt er nog gekeken hoeveel effort er van de developer nodig is om de library te integreren in een applicatie. Uit deze evaluaties wordt een conclusie getrokken over de Tracklytics library.

7.1 Performance

In het hoofdstuk Doelstellingen is aangehaald dat impact van de Tracklytics library op de performance aanvaardbaar moet zijn om de applicatie niet significant te doen vertragen. In deze sectie wordt deze impact onderzocht en geëvalueerd.

Om de performance te meten zijn er twee verschillende applicaties gekozen waar de library in ingebouwd is, namelijk een stopwatch applicatie en een angry birds kloon [14]. De stopwatch wordt gebruikt omdat dit een niet CPU intensieve applicatie is en zo de resultaten niet vertekend zijn. De angry birds applicatie is CPU intensiever en vereist een bepaalde framerate, aangezien dit een spel is. Deze applicatie wordt gebruikt om de impact van de Tracklytics library op de framerate te onderzoeken.

De evaluaties zijn uitgevoerd op een iPhone 6 [10] met als besturingssysteem: iOS versie 9.3.1. Het toestel was ten allen tijde voorzien van een WiFi verbinding met het internet. Om de resultaten op te halen van het toestel is de ontwikkelaarsomgeving Xcode [12] versie 7.3.1 gebruikt.

In de volgende secties worden de metingen uitgelegd en uitgevoerd. Er gaat gemeten worden hoe lang de gemiddelde uitvoeringstijd bedraagt per methode die gebruikt kan worden van de library. Nadien wordt er gekeken naar de impact van de library op de framerate van de angry birds kloon.

7.1.1 Call duration

In deze sectie wordt uitgezocht hoelang het gemiddeld duurt voor elke methode om volledig uitgevoerd te worden. Zo kan er een beeld gevormd worden van hoe sterk de library de applicatie zou vertragen. Om te testen hoe lang een methode gemiddeld duurt om uitgevoerd te worden, is de volgende testopstelling uitgedacht. De geteste methode wordt X aantal keer uitgevoerd om hieruit een gemiddelde tijd te kunnen berekenen. deze X wordt stelselmatig verhoogd om te kijken of de grootte van deze X een invloed heeft op de gemiddelde vertraging van de geteste methode. Deze testopstelling wordt gebruikt om alle beschikbare methodes in de Tracklytics library te evalueren. Het synchronisatieinterval is 60 seconden.

In de vorige secties is beschreven dat de developer de keuze gegeven is tussen het tijdelijk opslaan van de metingen op de harde schijf van het toestel en het niet opslaan van de metingen op de harde schijf van het toestel. Dit heeft een invloed op de prestaties van de library, omdat er een significant prestatieverschil zit in disk I/O en verwerkingskracht [25]. Om dit prestatieverschil in kaart te brengen is deze evaluatie uitgevoerd eenmaal met het opslaan op disk en eenmaal zonder het opslaan op disk (aangegeven in de resultaten als ZOD).

Er is in de Tracklytics library gekozen om de developer de vrijheid te geven om te beslissen dat de aggregatie van de metingen deels op het toestel van de gebruiker uitgevoerd wordt of dat deze aggregatie volledig uitgevoerd wordt in de back end. Omdat er een verschil is in de implementatie van de geaggregeerde meetobjecten en de standaard meetobjecten zijn de testen opgedeeld in testen voor de standaard meetobjecten en testen voor de geaggregeerde meetobjecten (in het geval van de Gauge, de Meter en de Timer). Zo kunnen er de juiste conclusies getrokken worden uit de testen die uitgevoerd zijn.

De resultaten van het testen van de standaard meetobjecten kunnen gevonden worden in volgende tabellen: 7.1, 7.7, 7.4, 7.5, 7.2. De resultaten van de testen op de geaggregeerde meetobjecten zijn opgelijst in volgende tabellen: 7.8, 7.6, 7.3. De conclusies die uit deze resultaten getrokken kunnen worden zijn beschreven in de conclusie sectie 7.4.

Om een vergelijking te geven van het verschil in duratie tijd van het opslaan op harde schijf en het niet opslaan op harde schijf zijn deze in grafieken samengevat: 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 7.10, 7.11, 7.12. Met behulp van deze grafieken is de evolutie van de duratie tijd zichtbaar.

7.1.2 Invloed op FPS

De gemiddelde tijd om een methode uit te voeren zegt iets over de vertraging die de library extra invoert in de applicatie. Het zegt echter niets over de impact op de framerate van de applicatie. Om dit te kunnen testen is er gekozen om de impact

van de library op een game te testen. Zoals eerder vermeldt is deze game een angry birds kloon.

Om deze test uit te voeren is de volgende testopstelling gebruikt. Het gemiddeld aantal frames per second (FPS) wordt berekend uit 3000 gemeten waarden. Een FPS waarde wordt elke 0.2 seconde gemeten. Om de impact van de library op deze FPS te testen is ervoor gekozen om elke seconde X aantal keer een library call uit te voeren en deze X telkens te verhogen. Er is gekozen om de worst case test uit te voeren gebaseerd op de test van de call duration. In de resultaten is te zien dat het aanmaken van het Counter object met het opslaan op disk het hoogste gemiddelde heeft. Het aanmaken van een Counter met het opslaan van de data op de harde schijf wordt gebruikt om de invloed van de library op de FPS te onderzoeken. Om de invloed van de library op de FPS van de applicatie te onderzoeken zonder dat de metingen op de harde schijf opgeslagen worden, wordt het stoppen van een Timer gebruikt, omdat deze de grootste call duratie heeft van de niet-geaggregeerde objecten.

Een andere factor die invloed kan hebben op de impact die de library heeft op de applicatie is het synchronisatieinterval. Dit interval is een indicatie van hoeveel tijd er tussen twee synchronisatiecycli bevindt. Door dit synchronisatieinterval systematisch te verlagen kan de invloed van de Tracklytics library op de FPS van de applicatie gemeten worden.

De resultaten van deze testen kunnen gevonden worden in de volgende tabellen: [7.9](#), [7.10](#). De linkerkolom geeft het aantal simultane metingen weer. Het synchronisatieinterval wordt systematisch verlaagd van 60 seconden naar 5 seconden. De conclusies die uit deze resultaten getrokken kunnen worden zijn beschreven in de conclusie sectie [7.4](#).

7.2 Schaalbaarheid

Het is belangrijk dat de Tracklytics back end een schaalbare back end is. Indien de load op de servers te groot wordt, dan zou dit merkbaar zijn in de applicatie. De Tracklytics library zou de netwerkkinterface constant gebruiken om de metingen te synchroniseren naar de back end. De applicatie waarin de library is ingebouwd deelt deze netwerkkinterface met de library. Als gevolg hiervan wordt de applicatie trager, omdat de netwerkkinterface constant gebruikt wordt door de library. Dit scenario moet voorkomen worden om de bruikbaarheid van de Tracklytics library optimaal te houden.

Een oplossing voor dit probleem is de back end uit te schalen over meerdere servers en servers bij te voegen indien de load op de servers te groot wordt. Om de back end schaalbaar te maken moet zowel de PHP code die op de servers draait schaalbaar zijn, alsook de database die in de back end draait.

De PHP code zet verwerkt de metingen komende van de Tracklytics library en zet deze resultaten in de database. Voor deze code is het voldoende dat deze een connectie tot een database hebben en dit maakt de PHP code dus zeer schaalbaar.

De database gebruikt in de Tracklytics library is een MySQL database. Dit is een implementatie van een Relational database management system (RDBMS). Dit soort database is niet ontwikkeld om schaalbaar te zijn [8]. Dit soort systeem is ontwikkeld om op één server te draaien om de integriteit van de tabel mappings te behouden en de problemen van gedistribueerde verwerking te vermijden. Indien dit systeem dan geschaald moet worden, dan moet er grotere en complexere hardware gekocht worden met meer verwerkingskracht, RAM geheugen en opslag. Een oplossing die hiervoor gevonden is, is een master-slave model waarbij de slaves de load van de master kunnen verlagen [32].

Een alternatief hiervoor zou zijn om een NoSQL database te gebruiken. Dit soort database is ontwikkeld om schaalbaar te zijn. Het nadeel hiervan is dat de ACID properties niet gegarandeerd kunnen worden, wat ervoor zorgt dat data inconsistent kan worden. Er bestaat ook nog geen standaard taal (zoals SQL voor RDBMS), zodat de overgang van een RDBMS (of van een andere NoSQL) naar een NoSQL database een probleem kan vormen.

Een alternatief is dat developers van een applicatie de Tracklytics back end hosten op hun eigen servers. Dit is mogelijk, omdat de library open source is. Enkel de connectie met de database moet aangepast worden en de links in de mobiele library moeten vervangen worden door een link naar de server van de developer. Dit geeft als voordeel dat de data zich bij de developer bevindt en de developer hier eventueel andere overzichten van kan genereren, aangepast aan zijn noden. In dit scenario is de developer niet afhankelijk van een gemeenschappelijke Tracklytics back end en heeft hij de performance van de Tracklytics back end in eigen handen. Het nadeel aan dit alternatief is dat de developer zelf serverruimte moet voorzien en onderhouden om de back end op te draaien.

7.3 Developer effort

Met DevOps willen developers en managers het ontwikkelingsproces van software versnellen. Indien het relatief veel tijd inneemt om de Tracklytics library in een applicatie in te bouwen is het developer aspect verdwenen uit deze thesis. Het is ongewenst dat het veel lijnen code inneemt om de library in de applicatie in te bouwen, omdat dit enerzijds meer tijd inneemt om de code te schrijven en anderzijds de code minder overzichtelijk maakt voor de developers.

Om deze eigenschappen na te gaan is ervoor gekozen om volgende zaken na te gaan: hoeveel tijd neemt het in om de library in te bouwen, het aantal lijnen code nodig om deze in de applicatie in te bouwen, wat er gemonitord wordt en hoeveel

metingen er maximaal simultaan voorkomen. Met "hoeveel tijd neemt het in om de library in te bouwen" wordt de gehele flow bedoeld: het registreren van de applicatie in de back end, het installeren van de library via CocoaPods en het implementeren van de code in de applicatie.

Om deze zaken na te gaan zijn er twee applicaties gebruikt, namelijk de stopwatch applicatie die eerder vermeldt is en een applicatie ontwikkeld in mijn stage. De stopwatch heeft volgende eigenschappen:

- 250 lijnen code
- 13 uren ontwikkelingstijd (inclusief design)

De tweede applicatie heeft volgende eigenschappen:

- \pm 10000 lijnen code
- 4 maanden ontwikkelingstijd (inclusief design)

7.3.1 Resultaten

In deze sectie worden de resultaten besproken van de evaluatie van de developer effort. In de volgende sectie worden deze resultaten besproken [7.4](#).

StopWatch applicatie

- Duratie: Minder dan 10 minuten
- #Lijnen code:
 - 9 meetobjecten in totaal
 - gemiddeld 1.0 lijnen per meetobject (zonder imports)
 - in totaal 9 lijnen extra code
- Tracken van: alle buttons, tijd wanneer op stop gedrukt, aantal rondes alvorens reset, tijd om data van een server te halen.
- Maximaal 2 metingen tegelijk (Counter & Gauge)

Andere applicatie

- Duratie: \pm anderhalf uur
- #Lijnen code:
 - 68 meetobjecten in totaal
 - gemiddeld 1.4 lijnen per meetobject (zonder imports)
 - in totaal \pm 100 lijnen extra code

- Tracken van: Alle buttons waarop geklikt wordt, alle screen visits, #zoekresultaten, duratie van complexe code, duratie content van het netwerk halen
- Maximaal ± 10 metingen tegelijk (Counters, Timers en Gauges)

Het is nuttig om dit soort informatie te meten, omdat met behulp van dit soort metingen kan ondervonden worden hoe de gebruiker de applicatie gebruikt. Er kan zo een overzicht gevormd worden welke knoppen hoe vaak gebruikt worden en welke schermen dat de gebruiker het vaakst bezoekt. Anderzijds is het monitoren van de duratie van complexe code en van het ophalen van informatie van het netwerk een manier om bottlenecks te kunnen ontdekken in de applicatie.

7.4 Conclusie en bespreking resultaten

In het hoofdstuk Doelstellingen is er besproken dat de impact van de ontwikkelde Tracklytics library op de performance van de applicatie zo minimaal mogelijk moet zijn. Indien we de resultaten van de verschillende meetobjecten bekijken zien we dat de library de grootste impact heeft op de applicatie indien de library de metingen op de harde schijf opslaat. Indien we gaan kijken naar de Counter tabel zien we dat het aanmaken van een Counter gemiddeld ± 2000 keer sneller is indien het object niet op de harde schijf opgeslagen wordt. Dit resultaat geldt voor het aanmaken van elk niet-geaggregeerd meetobject. De addEntry methode bij de Meter maakt een Meter object aan en hoort dus bij het aanmaken van een niet-geaggregeerd meetobject. Indien we gaan kijken naar het snelheidsverschil bij het uitvoeren van een operatie op een niet-geaggregeerd meetobject, kunnen we vaststellen dat het verhogen en verlagen van een Counter ± 1000 maal sneller is indien de data niet op harde schijf wordt opgeslagen. Het stoppen van een Timer is ± 500 maal sneller indien deze Timer niet op de harde schijf van het toestel opgeslagen wordt. Uit deze observatie kunnen we concluderen dat de performance impact van de Tracklytics library groter is indien de metingen tijdelijk op de harde schijf opgeslagen worden.

Door naar de grafieken te kijken van de niet-geaggregeerde meetobjecten kunnen we zien dat er zich telkens dezelfde trend voordoet: de gemiddelde tijd die het inneemt om een meetobject aan te maken neemt toe naarmate er meer objecten aangemaakt worden en de hoeveelheid uitvoeringen van een methode op een meetobject heeft een beperkte invloed op de gemiddelde uitvoeringstijd van de methode. Deze observatie geldt zowel voor de metingen die opgeslagen worden op harde schijf als de metingen die niet opgeslagen worden op de harde schijf. Combineren we de grafieken met de resultaten in de tabellen kunnen we concluderen dat het aanmaken van Counters de zwaarste performance impact heeft.

Zoals eerder besproken in deze thesis geeft de Tracklytics library de keuze aan de developer om de data op het toestel of in de back end te aggregeren. We hebben de metingen vergeleken van de geaggregeerde en de niet-geaggregeerde objecten in de volgende grafieken: [7.14](#), [7.13](#), [7.16](#), [7.15](#), [7.20](#), [7.19](#), [7.18](#), [7.17](#). Indien we deze

grafieken bekijken, dan kunnen we deze opdelen in twee verschillende secties, namelijk langs de ene kant de grafieken die te maken hebben met de Gauge en langs de andere kant de grafieken die te maken hebben met de Meter en de Timer.

De tijd om een GaugeAggregate aan te maken daalt eerst tot en met er ongeveer 100 metingen simultaan gebeuren en stijgt dan later vanaf ongeveer 10000 simultane metingen opnieuw. De reden dat de tijd om een GaugeAggregate aan te maken deze karakteristiek heeft, is dat de Tracklytics library een lijst van alle waarden bijhoudt in de GaugeAggregate klasse om de exacte mediaan te kunnen berekenen.

In de grafieken die de call duratie van de Meter en de Timer vergelijken met de geaggregeerde objecten is er een trend te zien, namelijk: eerst daalt de grafiek en daarna blijft de grafiek ongeveer horizontaal. We kunnen zien dat indien er veel metingen gebeuren, de tijd die de geaggregeerde functies innemen kleiner of ongeveer gelijk zijn aan de niet-geaggregeerde functies.

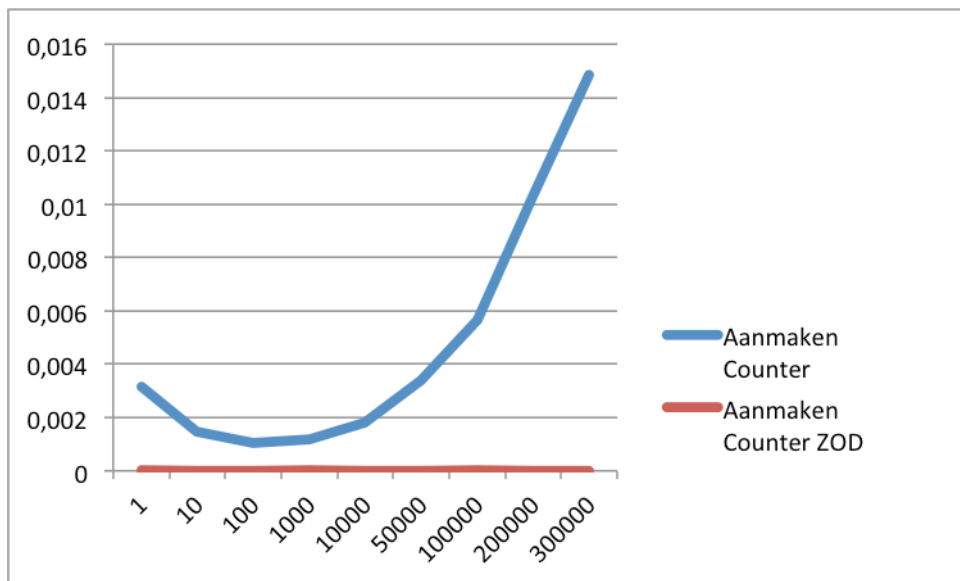
Bekijken we de resultaten van hoeveel effort een developer in een applicatie moet stoppen (7.3), concluderen we dat bij een grote applicatie er ongeveer 10 metingen tegelijk kunnen gebeuren. Indien we dit resultaat combineren met de resultaten van de grafieken en cijfers van de metingen van de geaggregeerde objecten, zien we dat, met uitzondering tot het stoppen van een Timer, het steeds het geval is dat het niet aggregeren van de metingen op het toestel sneller of ongeveer even snel is als het aggregeren van de metingen op het toestel tot en met 100 simultane metingen. Uit deze observatie kunnen we afleiden dat het aggregeren van de metingen op het toestel van de gebruiker een grotere performance impact heeft dan het aggregeren van de metingen in de back end.

Naast het evalueren van de call duratie van de Tracklytics library hebben we de impact op de FPS van de Tracklytics library geëvalueerd. Indien we deze resultaten bekijken, bekommen we dat tot en met 20 simultane metingen, de impact op de applicatie van de library minimaal is indien de gegevens op de harde schijf opgeslagen wordt. Het aantal FPS daalt pas significant indien het aantal simultane metingen richting 50 gaat. Indien de gegevens niet op de harde schijf opgeslagen worden, heeft de library een minimale invloed op de applicatie. Deze observatie zorgt ervoor dat, in het geval dat FPS de limiterende factor is, de library in elke applicatie ingebouwd kan worden indien het maximum aantal simultane metingen in de buurt van 20 ligt indien de developer wil dat alle gegevens tijdelijk op de harde schijf van de applicatie opgeslagen wordt.

Een gebruiker merkt een vertraging pas indien de vertraging groter is dan 0.1 seconde [29]. De Tracklytics library zou dit getal niet mogen overstijgen om de impact op de performance aanvaardbaar te houden. In de resultaten van de call duratie zien we dat bij elk meetobject 100 simultane uitvoeringen gedaan moeten worden om de 0.1 seconde vertraging te overstijgen indien de metingen op de harde schijf van de telefoon opgeslagen worden.

Uit deze observaties kunnen we concluderen dat de library bruikbaar is op het vlak van performance bij het ontwikkelen van een applicatie, zelfs indien de data op de harde schijf opgeslagen wordt. De developers moeten er rekening mee houden dat er maximaal rond de **20 simultane metingen** in de applicatie mogen gebeuren om de vertraging van de applicatie binnen de perken te houden indien de data tijdelijk op de harde schijf opgeslagen wordt. Indien de metingen niet op de harde schijf opgeslagen worden alvorens te synchroniseren naar de back end, dan staat er geen échte limiet op het aantal simultane metingen. Meer dan 1000 simultane metingen zijn mogelijk in deze situatie alvorens de library een significante impact heeft op de prestaties van de applicatie.

De effort van de developer moet met een korrel zout genomen worden, omdat de ontwikkelaar van de library deze testen zelf heeft uitgevoerd. Het is mogelijk dat developers een langere tijd nodig hebben om de Tracklytics library in te bouwen in een applicatie. Zelfs al verdubbel je de tijd die nodig was om de library te implementeren, is de effort die de developer moet steken in het inbouwen van de Tracklytics library laag. Hieruit concluderen we dat de library bruikbaar is op het vlak van developer effort.



FIGUUR 7.1: Grafiek vergelijking tijden aanmaken van een Counter.

# metingen	Aanmaken	Inc	Dec	Aanmaken ZOD	Inc ZOD	Dec ZOD
1	0,003168987	0,000516049	0,000513007	4,41015E-06	5,87324E-07	5,41994E-07
10	0,001451302	0,000512397	0,000505489	1,68934E-06	5,30546E-07	5,29546E-07
100	0,001045819	0,000505216	0,000506907	1,18792E-06	5,50558E-07	5,7864E-07
1000	0,001175395	0,000522625	0,000505655	2,7231E-06	5,35803E-07	5,27143E-07
10000	0,0017926232	0,000545799	0,000527276	2,05297E-06	5,54156E-07	5,10363E-07
50000	0,003394888	0,00053188	0,000539547	1,7636E-06	5,45784E-07	5,8755E-07
100000	0,005672893	0,000534656	0,0005382	2,30629E-06	5,26104E-07	4,93241E-07
200000	0,010294359	0,000530225	0,000537563	1,66409E-06	4,97316E-07	5,43947E-07
300000	0,014826151	0,000553307	0,000544871	1,76572E-06	5,45442E-07	5,08001E-07
Gemiddeld	0,004758046	0,000528017	0,000524279	2,17369E-06	5,41448E-07	5,35603E-07

TABEL 7.1: Resultaten call tijd Counter (in seconden)

# metingen	Aanmaken	Stop	Aanmaken ZOD	Stop ZOD
1	0,000342011	0,000580966	4,09013E-06	2,89679E-05
10	0,000284296	0,000492996	1,91528E-06	5,6982E-06
100	0,000298141	0,00049423	2,07685E-06	2,2608E-06
1000	0,000297281	0,000522041	2,03055E-06	2,18725E-06
10000	0,000740984	0,000527625	1,88986E-06	1,26034E-06
50000	0,002660037	0,000536449	2,03353E-06	1,17797E-06
100000	0,00515636	0,000544189	1,83201E-06	1,17261E-06
200000	0,009800656	0,000575258	1,86301E-06	1,16086E-06
300000	0,014302578	0,000522853	1,75735E-06	1,11202E-06
Gemiddeld	0,003764705	0,000532956	2,1654E-06	4,99977E-06

TABEL 7.2: Resultaten call tijd Timer (in seconden)

# metingen	Aanmaken	Stop	Aanmaken ZOD	Stop
1	0,002120972	0,002666599	4,36699E-05	4,00181E-06
10	0,000227898	0,002475894	3,0541E-05	1,69697E-06
100	2,12032E-05	0,000657661	2,74795E-05	1,77817E-06
1000	3,52955E-06	0,000614075	3,24672E-06	1,51634E-06
10000	1,26566E-06	0,000587031	1,25878E-06	1,43751E-06
50000	1,0892E-06	0,000584363	1,12291E-06	1,36487E-06
100000	1,02262E-06	0,000574846	1,04058E-06	1,45262E-06
200000	1,03471E-06	0,000565405	1,02583E-06	1,36E-06
300000	1,0063E-06	0,000574488	1,01804E-06	1,4087E-06
Gemiddeld	0,000264336	0,001033374	1,2267E-05	1,77967E-06

TABEL 7.3: Resultaten call tijd Timer met Aggregatie (in seconden)

# metingen	Aanmaken	Aanmaken ZOD
1	0,002752006	4,21825E-06
10	0,001020319	1,94199E-06
100	0,001067632	2,11683E-06
1000	0,001118987	2,89034E-06
10000	0,001677553	2,00928E-06
50000	0,003420756	1,86953E-06
100000	0,005801659	1,86129E-06
200000	0,010283894	1,85536E-06
300000	0,014828387	1,8752E-06
Gemiddeld	0,004663466	2,29312E-06

TABEL 7.4: Resultaten call tijd Histogram (in seconden)

# metingen	addEntry	addEntry ZOD
1	0,002198004	3,99419E-06
10	0,001131004	2,17136E-06
100	0,001070941	2,1365E-06
1000	0,001186648	2,36198E-06
10000	0,001511716	2,28242E-06
50000	0,003305941	2,04427E-06
100000	0,005528791	2,23297E-06
200000	0,010175614	1,88071E-06
300000	0,015041323	1,89719E-06
Gemiddeld	0,00457222	2,33351E-06

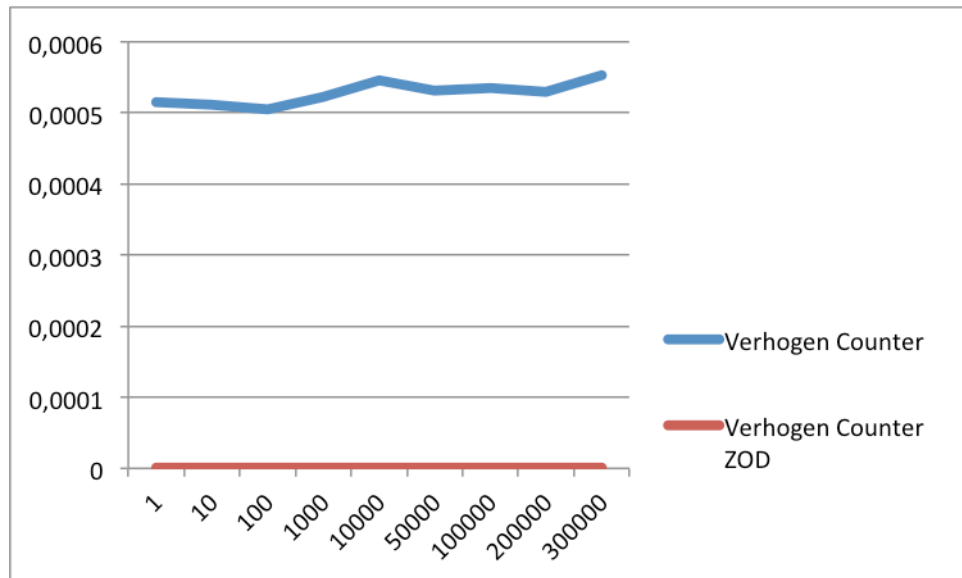
TABEL 7.5: Resultaten call tijd Meter (in seconden)

# metingen	addEntry	addEntry ZOD
1	0,025438964	4,48893E-05
10	0,003257507	2,90151E-05
100	0,000962322	6,39166E-06
1000	0,000811738	1,56662E-06
10000	0,000793704	1,18229E-06
50000	0,000813867	1,15234E-06
100000	0,000769992	1,0986E-06
200000	0,000618082	1,15231E-06
300000	0,000561398	1,10977E-06
Gemiddeld	0,003780842	9,72867E-06

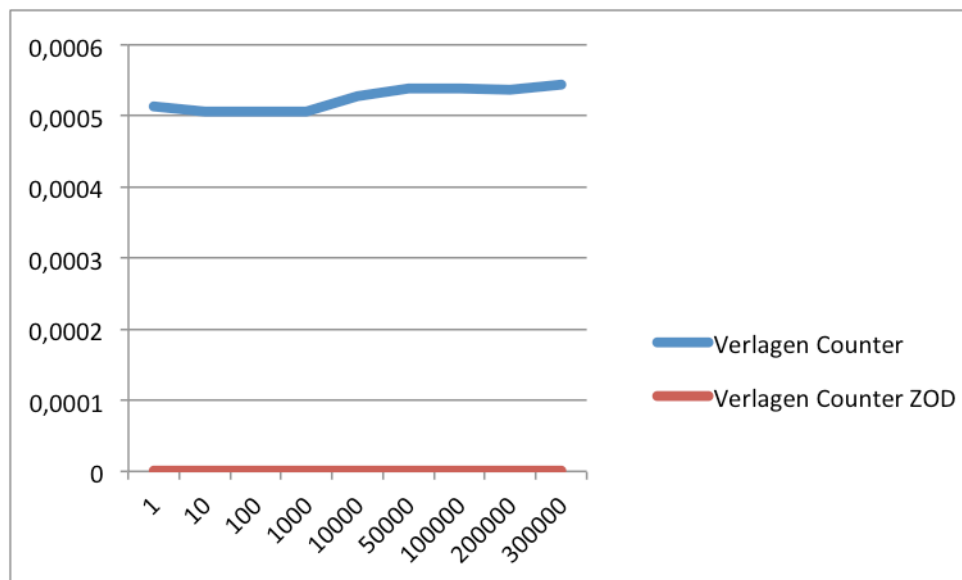
TABEL 7.6: Resultaten call tijd Meter met Aggregatie (in seconden)

# metingen	Aanmaken	Aanmaken ZOD
1	0,002469957	4,12756E-06
10	0,001076302	1,95558E-06
100	0,001117599	2,09727E-06
1000	0,001157752	2,47002E-06
10000	0,001552656	2,00483E-06
50000	0,003327206	1,81246E-06
100000	0,005833407	1,91939E-06
200000	0,010283247	1,82617E-06
300000	0,015243511	1,90657E-06
Gemiddeld	0,004673515	2,23554E-06

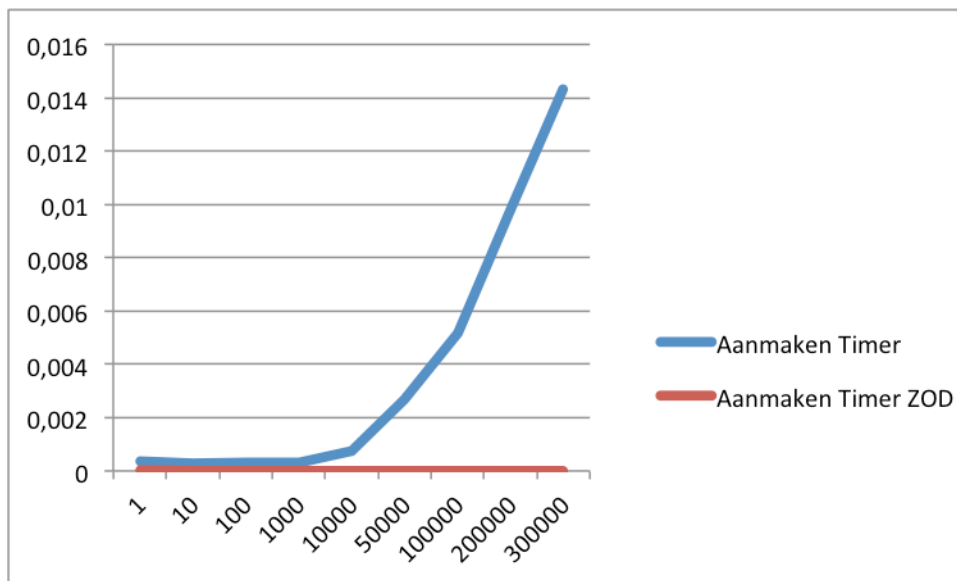
TABEL 7.7: Resultaten call tijd Gauge (in seconden)



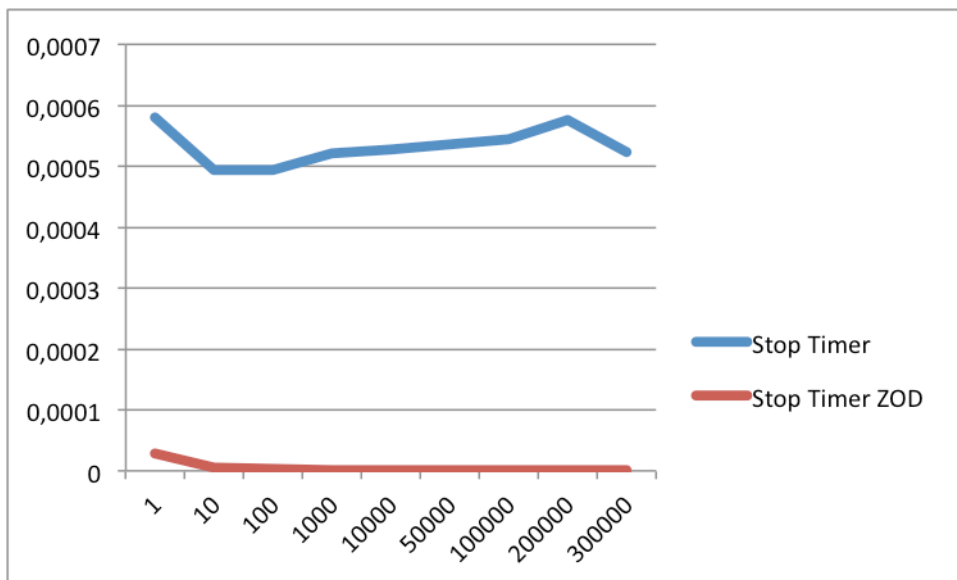
FIGUUR 7.2: Grafiek vergelijking tijden verhogen van een Counter.



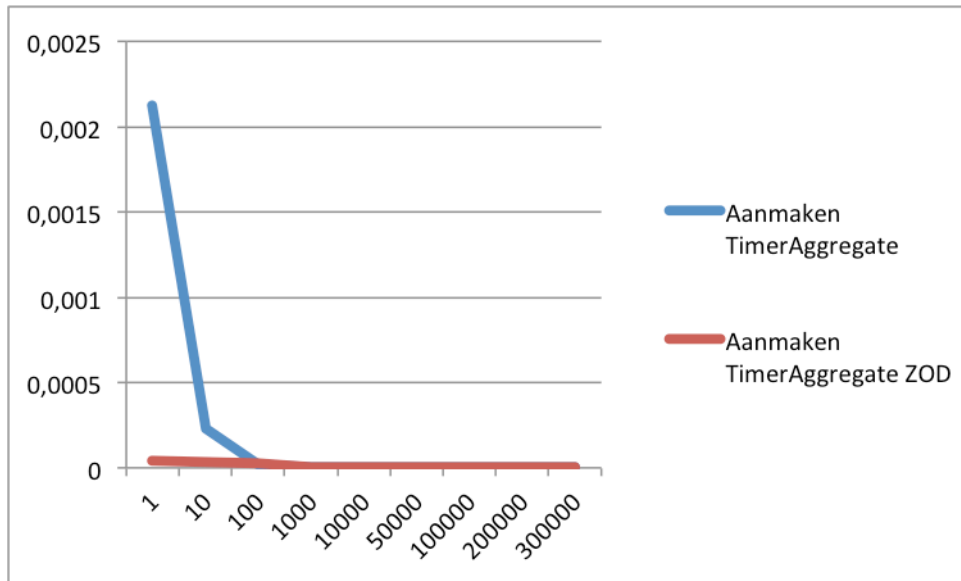
FIGUUR 7.3: Grafiek vergelijking tijden verlagen van een Counter.



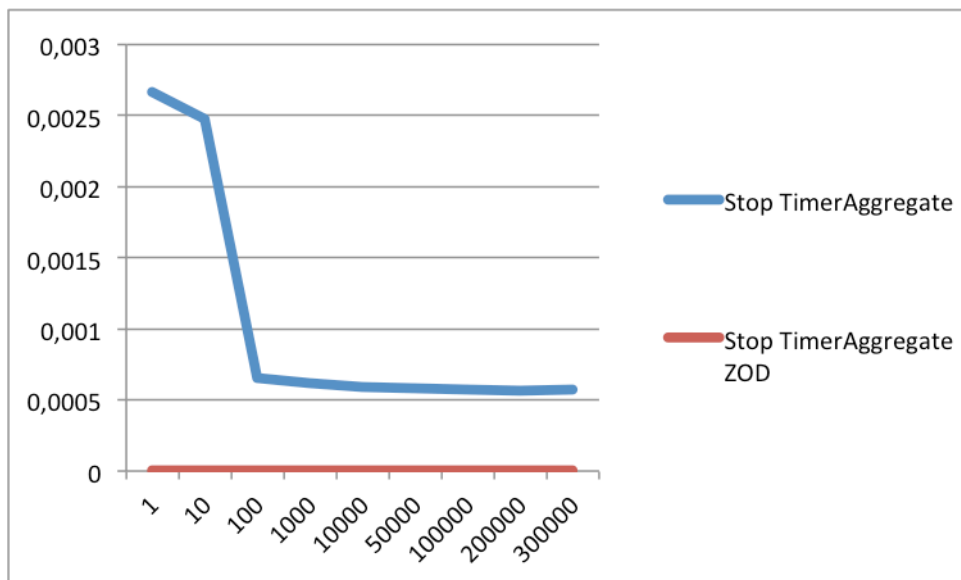
FIGUUR 7.4: Grafiek vergelijking tijden aanmaken van een Timer.



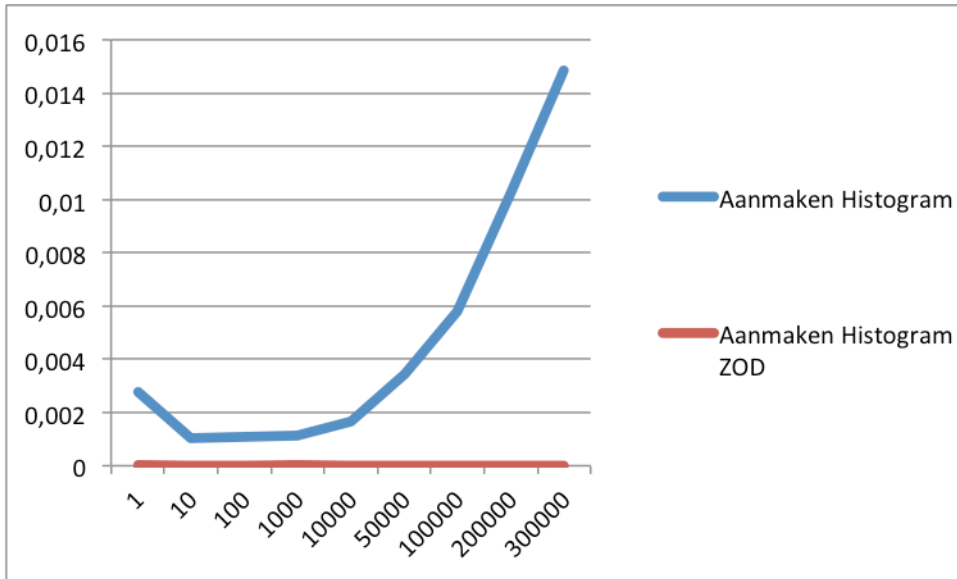
FIGUUR 7.5: Grafiek vergelijking tijden stoppen van een Timer.



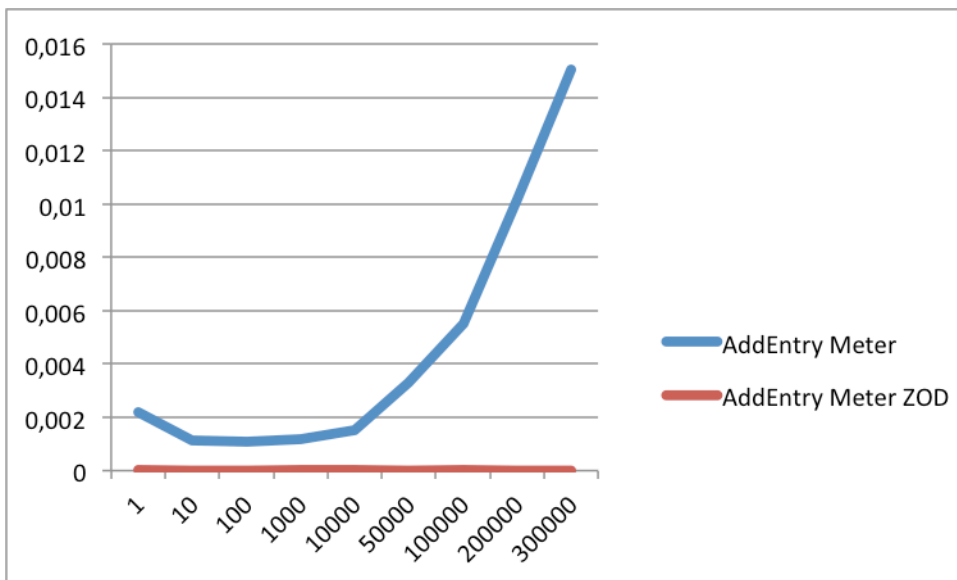
FIGUUR 7.6: Grafiek vergelijking tijden aanmaken van een TimerAggregate object.



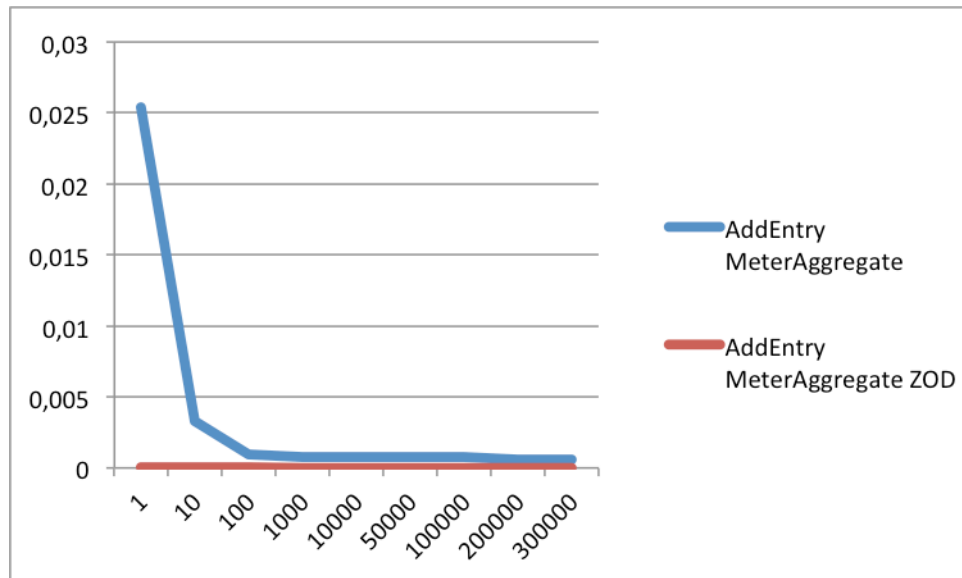
FIGUUR 7.7: Grafiek vergelijking tijden stoppen van een TimerAggregate object.



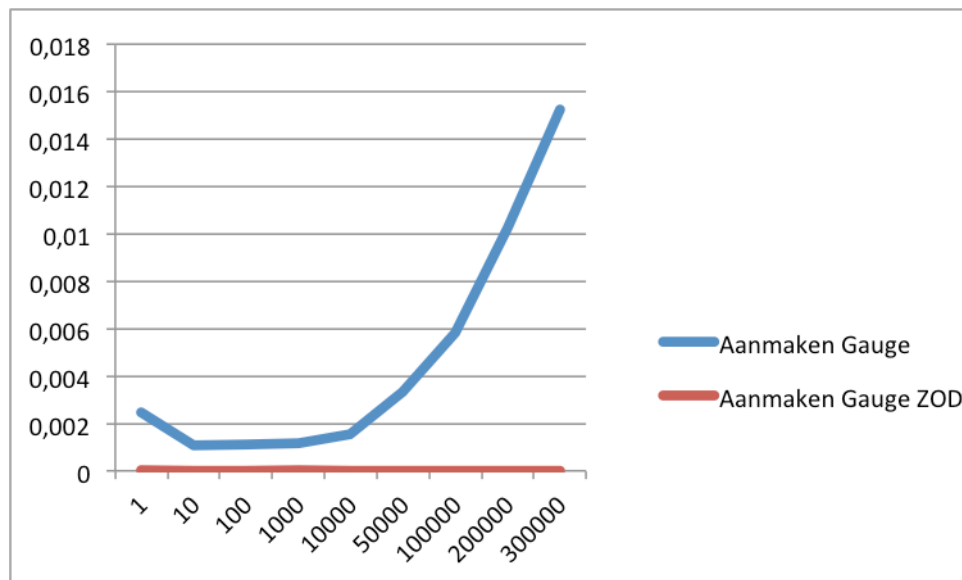
FIGUUR 7.8: Grafiek vergelijking tijden aanmaken van een Histogram.



FIGUUR 7.9: Grafiek vergelijking tijden addEntry van een Meter.



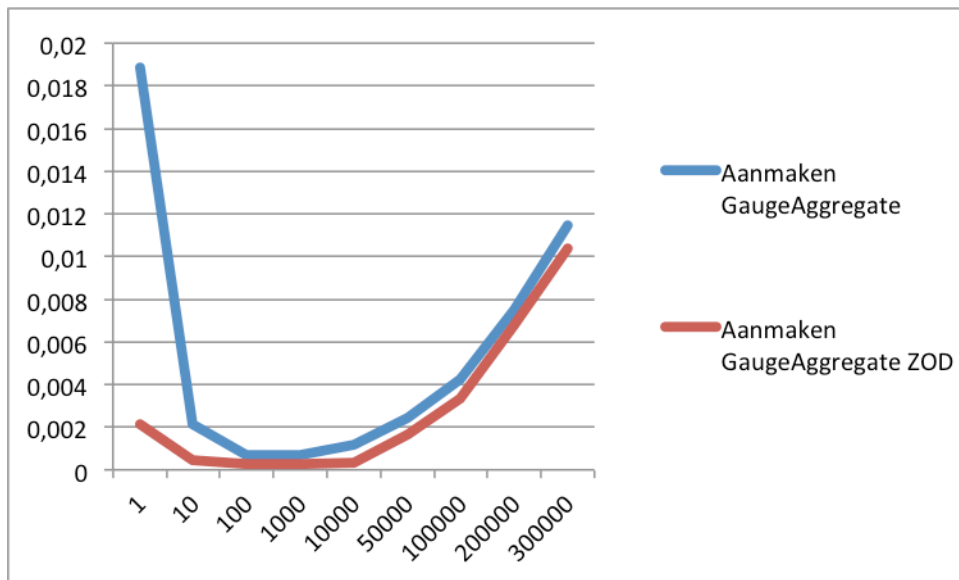
FIGUUR 7.10: Grafiek vergelijking tijden addEntry van een MeterAggregate object.



FIGUUR 7.11: Grafiek vergelijking tijden aanmaken van een Gauge.

# metingen	Aanmaken	Aanmaken ZOD
1	0,018877983	0,002133965
10	0,002148187	0,0004381
100	0,00066519	0,000245582
1000	0,000668403	0,000252874
10000	0,001156017	0,000333561
50000	0,002451678	0,001636235
100000	0,004218748	0,003317311
200000	0,007491983	0,006745731
300000	0,011473117	0,01040852
Gemiddeld	0,005461256	0,002834653

TABEL 7.8: Resultaten call tijd Gauge met Aggregatie (in seconden)



FIGUUR 7.12: Grafiek vergelijking tijden aanmaken van een GaugeAggregate.

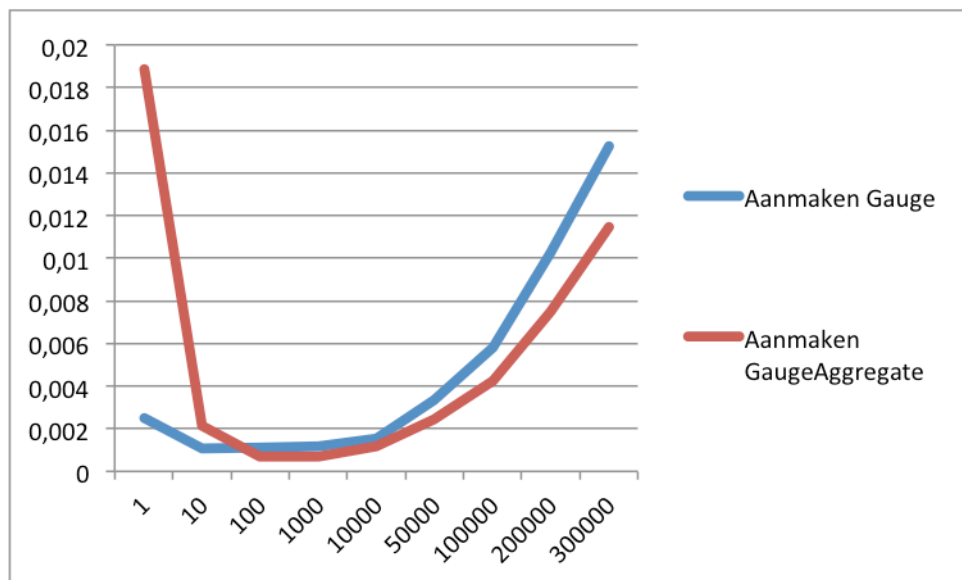
7. EVALUATIE

#metingen/sec	60 sec sync	30 sec sync	15 sec sync	10 sec syn	5 sec sync
0	31,5632141	31,6737281	31,5281931	31,5672819	31,0382918
1	31,5482895	31,5798341	31,5577123	31,5036609	31,2145193
10	31,4920183	31,5852394	31,1482867	31,2389401	31,2015632
20	30,9340218	30,9576821	30,8974831	30,91473821	30,7183928
50	29,8320184	29,3209473	29,3382103	29,1371842	27,0873822
100	20,8572947				
1000	9,3930281				

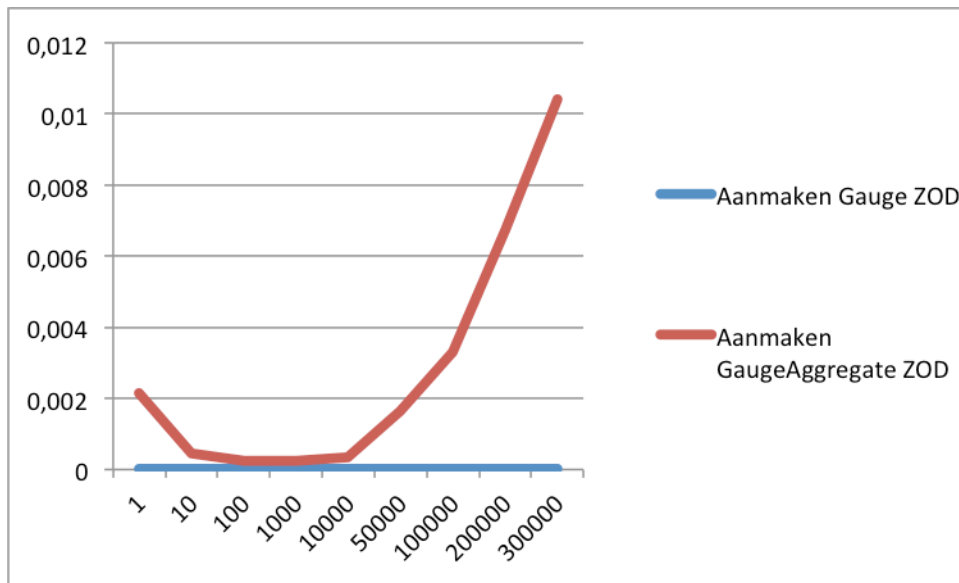
TABEL 7.9: Resultaten metingen van het aanmaken van een Counter (in FPS)

#metingen/sec	60 sec sync	30 sec sync	15 sec sync	10 sec syn	5 sec sync
0	31,1820647	31,8803964	31,3142858	31,6800421	31,22479
1	31,0432074	31,0152085	31,2826062	31,6564528	31,6203637
10	31,0542531	31,1544302	31,6991781	31,5175955	31,7503975
20	31,3124629	31,3124629	31,3124629	31,3124629	31,3124629
50	31,3699147	31,0174307	31,1349254	31,1842869	31,41264
100	31,2135833	31,222065	31,1640447	31,1833848	31,1436048
1000	30,2901981	30,2194739	30,2063641	30,1752	30,1750657

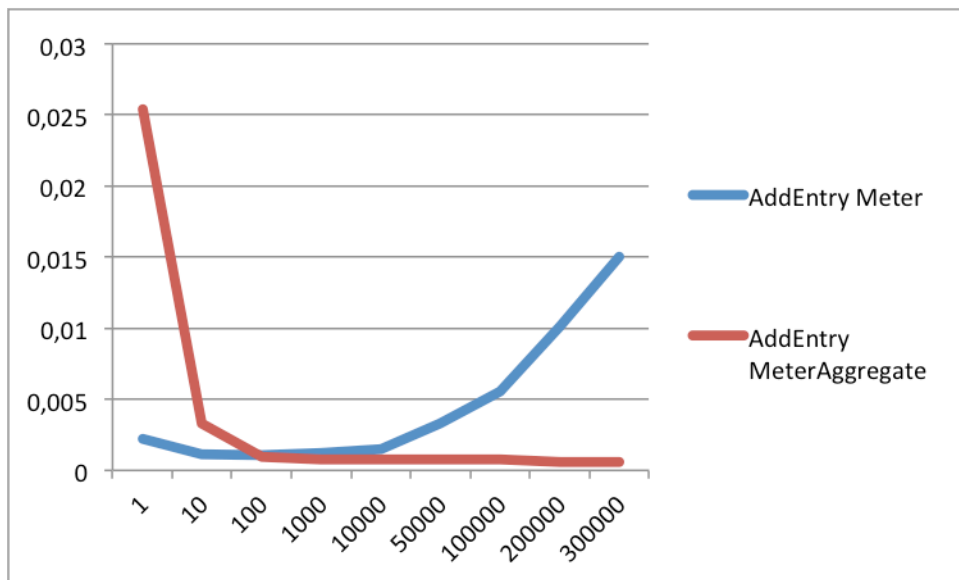
TABEL 7.10: Resultaten metingen van het stoppen van een Timer zonder opslaan op harde schijf (in FPS)



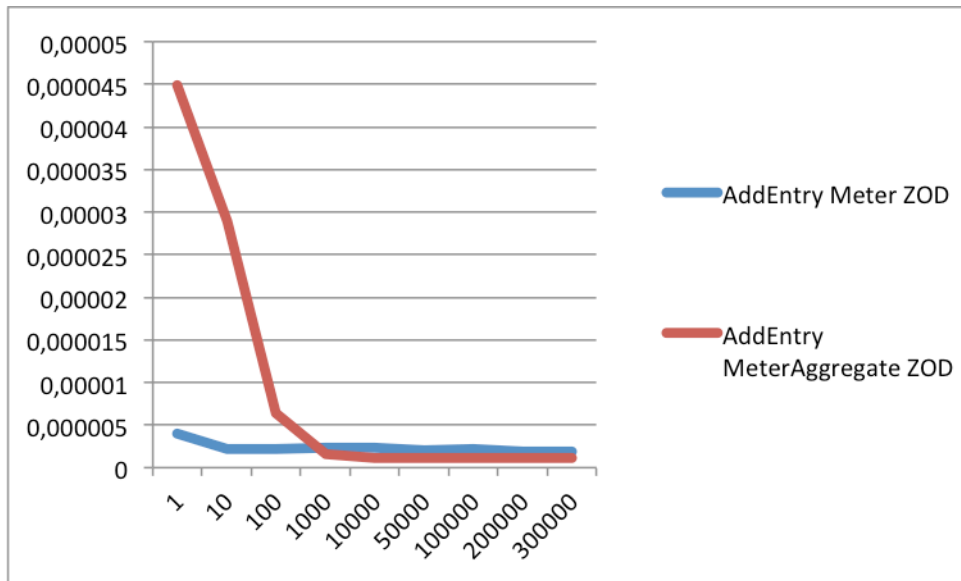
FIGUUR 7.13: Grafiek vergelijking tussen het aanmaken van een Gauge en een GaugeAggregate object.



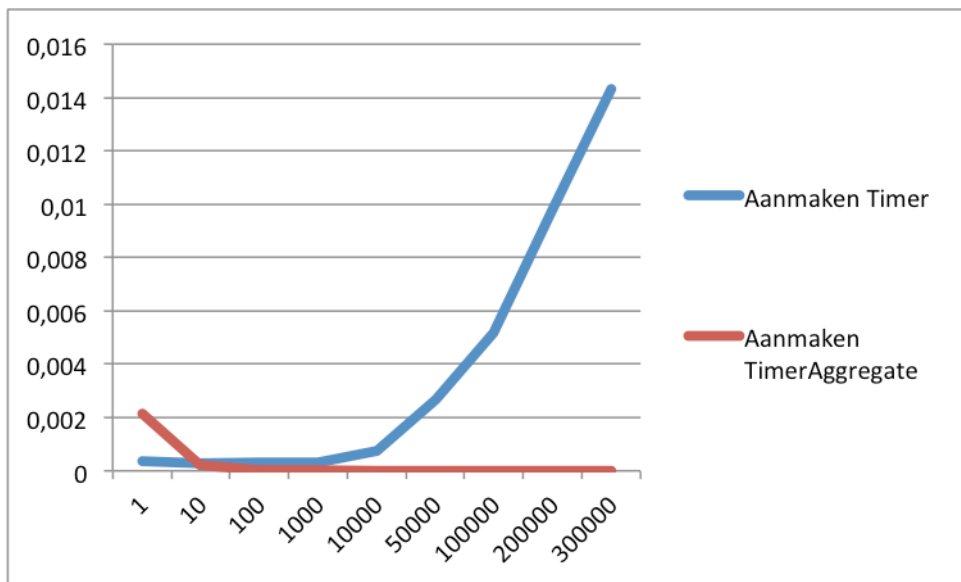
FIGUUR 7.14: Grafiek vergelijking tussen het aanmaken van een Gauge en een GaugeAggregate object zonder opslaan op de harde schijf.



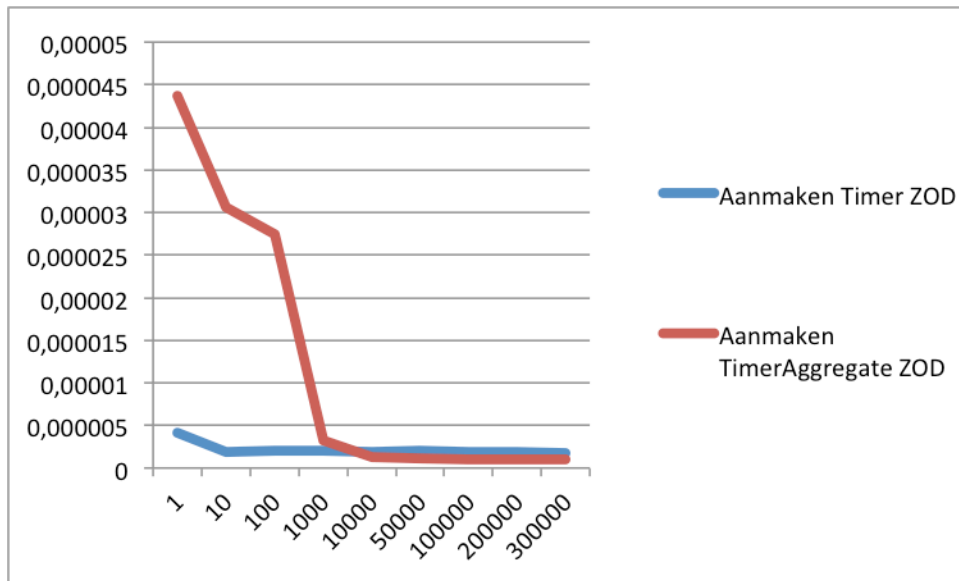
FIGUUR 7.15: Grafiek vergelijking tussen het aanmaken van een Meter en een MeterAggregate object.



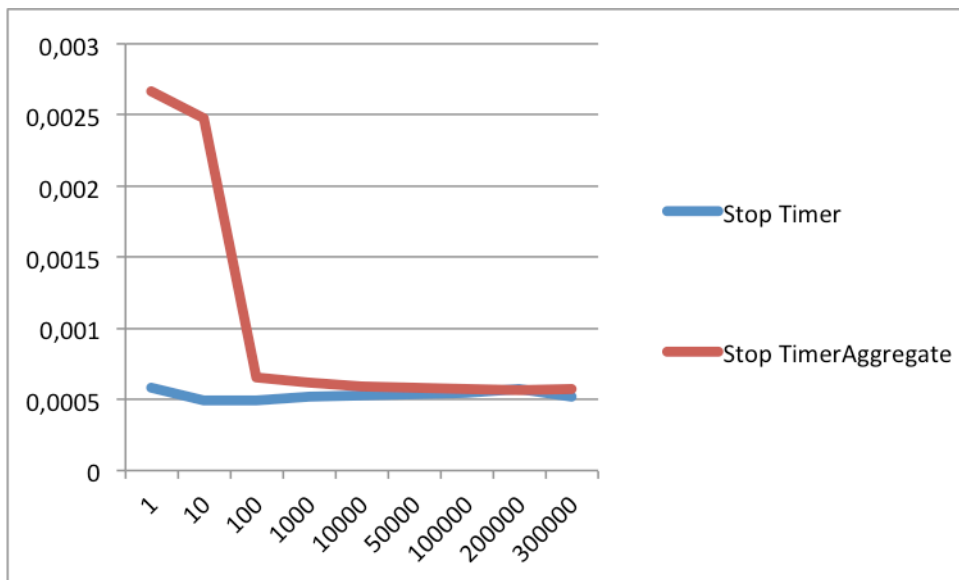
FIGUUR 7.16: Grafiek vergelijking tussen het aanmaken van een Meter en een MeterAggregate object zonder opslaan op de harde schijf.



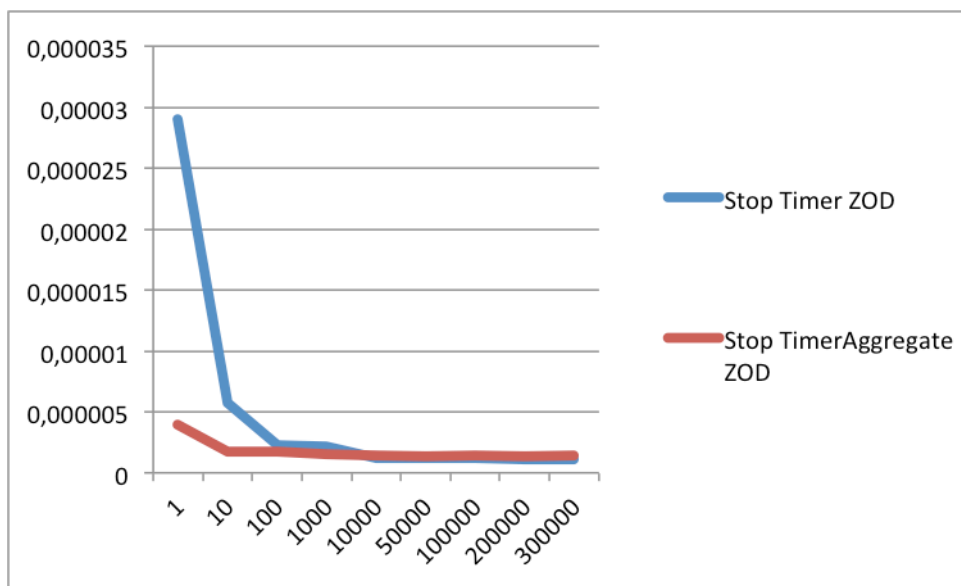
FIGUUR 7.17: Grafiek vergelijking tussen het aanmaken van een Timer en een TimerAggregate object.



FIGUUR 7.18: Grafiek vergelijking tussen het aanmaken van een Timer en een TimerAggregate object zonder opslaan op de harde schijf.



FIGUUR 7.19: Grafiek vergelijking tussen het aanroepen van de stop methode op een Timer en het aanroepen van de stop methode op een TimerAggregate object.



FIGUUR 7.20: Grafiek vergelijking tussen het aanroepen van de stop methode op een Timer en het aanroepen van de stop methode op een TimerAggregate object zonder opslaan op de harde schijf..

Hoofdstuk 8

Besluit

In deze thesis is er op zoek gegaan naar een oplossing om het ontwikkelen van mobiele applicaties en de concepten van DevOps te combineren. Er is gekozen om te kijken naar het monitoring aspect van DevOps. Er zijn enkele oplossingen voorgesteld om mobiele applicaties te monitoren. Uit deze voorstellen is één voorstel gekozen, namelijk een library om in de mobiele applicatie in te bouwen die de applicatie monitort. Voor deze library zijn er enkele doelstellingen opgesteld waaraan moet voldaan worden om een succesvolle monitoring library te bouwen. De vormgeving en architectuur van de monitoring library wordt voorgesteld in het architectuur hoofdstuk samen met enkele uitbreidingen op de library. Deze architectuur is deels uitgewerkt in een implementatie. De library is ontwikkeld voor iOS, het besturingssysteem voor mobiele toestellen van Apple. Deze implementatie is ten slotte geëvalueerd om te kijken of deze implementatie kan werken in de realiteit.

In het hoofdstuk doelstellingen [3](#) zijn een aantal doelstellingen opgesomd voor de ontwikkelde library. Deze werden opgedeeld in volgende categorieën:

- performance impact
- schaalbaarheid
- bruikbaarheid
- beschikbaarheid

De performance impact werd gezien als het belangrijkste aspect in het ontwikkelen van een monitoring library. Deze impact is onderzocht en getest in het evaluatie hoofdstuk [7](#). Uit deze evaluatie kan men afleiden dat de impact van de library op een mobiele applicatie klein genoeg is om de applicatie niet significant te vertragen.

De schaalbaarheid van de Tracklytics library is besproken in het evaluatie hoofdstuk [7](#). De back end is het deel van de library dat schaalbaar moet zijn om de

requests die van de mobiele library komen te verwerken. De bottleneck in de back end is de relationele database. Deze moet zo goed mogelijk uitgeschaald worden om aan de requests te kunnen voldoen. Een alternatief is om de relationele database te veranderen naar een NoSQL of andere schaalbare database. Het is ook mogelijk voor de developer om de back end op de eigen servers te draaien om zo alle data zelf te hebben en niet afhankelijk te zijn van de Tracklytics infrastructuur.

De bruikbaarheid van de library wordt gedefiniëerd als hoeveel moeite een developer nodig heeft om de library in te bouwen in de mobiele applicatie. In het evaluatie hoofdstuk 7 is nagegaan hoeveel moeite een developer nodig heeft om de library in de applicatie in te bouwen. In deze evaluatie zijn de volgende zaken opgenomen: de totale tijd om de library in te bouwen, het aantal lijnen code en wat er gemonitord wordt door de library. Uit de gegevens die hieruit zijn gekomen kan geconcludeerd worden dat de developer effort minimaal is en de library dus in dit aspect bruikbaar is.

Om de beschikbaarheid van de data maximaal te houden moet elke meting naar de back end verstuurd worden. De Tracklytics library laat de keuze of de data tijdelijk op de harde schijf van het toestel opgeslagen moet worden over aan de developer zelf. De developer moet deze keuze aangeven in het dashboard.

De voorgaande paragrafen tonen aan dat de doelstellingen vooropgesteld in deze thesis voldaan zijn in de implementatie van de library. Naast deze doelstellingen werd in het hoofdstuk over architectuur 4 nog enkele vereisten gegeven waaraan de library moet voldoen om de developer te helpen bij het ontwikkelen van een mobiele applicatie, namelijk:

- Op welke buttons/switches/entry in een tabel gebruikers drukken
- Welke schermen de gebruikers bezoeken
- Het gemiddeld aantal zoekresultaten per zoekopdracht
- Het gemiddelde of de verdeling van het getal dat een gebruiker in een bepaald veld invoert
- Hoe lang de applicatie gemiddeld gebruikt wordt
- Hoe lang een stuk code over het uitvoeren ervan doet om na te gaan of deze code niet te traag is.
- De tijd die een request over het internet nodig heeft om te voltooien om te kijken of er hier een vertraging opgelopen wordt.
- Het gemiddeld aantal entries in een array of een NSDictionary (een Map in Java). Indien er geïtereerd wordt over deze datastructuren kan een groot aantal entries ervoor zorgen dat dat stuk code de applicatie vertraagt.

-
- Het gemiddeld aantal keer dat een bepaalde methode uitgevoerd wordt om te kijken

Met behulp van de Tracklytics library kan aan deze vereisten voldaan worden. De eerste twee vereisten kunnen ingelost worden door counters in te bouwen in de applicatie. De tweede, derde, voorlaatste en laatste vereiste kunnen opgelost worden door ofwel een gauge te gebruiken ofwel een histogram te gebruiken. Aan de vierde, de vijfde en de zesde vereiste kan voldaan worden door een timer in te bouwen in de applicatie. Dit zorgt ervoor dat aan deze doelstelling voldaan is.

Tijdens het oplossen van deze doelstellingen en het ontwikkelen van de Tracklytics library zijn er enkele problemen opgetreden. Allereerst was het moeilijk om uit te denken wat er gemonitord zou moeten worden en welke types van meetobjecten er in de library geïmplementeerd moesten worden. Het was daarnaast moeilijk om deze types meetobjecten om te zetten in een performante mobiele library die bruikbaar kon zijn in de realiteit. Het opstellen van nuttige testopstellingen was een moeilijkheid, omdat we de impact van de library zo goed mogelijk wouden testen.

Naast het ontwikkelen van de mobiele applicatie was het ontwikkelen van het dashboard de grote moeilijkheid. Allereerst is er uitgezocht hoe we de verschillende meetobjecten het beste kunnen weergeven. Deze weergave is getweakt tot we dachten dat het de beste weergave was die we de developers konden geven. Daarnaast heeft het ontwikkelen van dit dashboard veel tijd gekost. Enerzijds omdat web development relatief nieuw voor mij was en anderzijds omdat er complexe geneste lussen zitten in het ophalen van de data bij de detail pagina's.

Een monitoring library heeft zijn voor- en nadelen. De gebruikers kunnen de applicatie anders gebruiken dan dat deze getest is in een test opstelling. Een monitoring library kan deze informatie ophalen, omdat deze in de applicatie, die in productie is, ingebouwd is. Het nadeel aan een monitoring library is dat deze een extra performance impact heeft op de applicatie, maar zoals eerder besproken is deze impact klein genoeg. Het dashboard zou nog wat meer functionaliteit kunnen hebben door nog gedetailleerdere overzichten te geven of extra mogelijkheden te geven tot het verkleinen van de dataset.

Indien ik nog meer tijd zou hebben gehad in het ontwikkelen van deze thesis zou ik graag nog wat uitbreidingen toegevoegd hebben aan de monitoring library. Als eerste zou ik graag het tracken van features hebben toegevoegd aan de applicatie. Features zijn een samenhang van componenten die een functie uitvoeren. Naar dit soort tracking van applicaties is momenteel veel onderzoek naar en ik zou dit onderzoek willen combineren met mobiele applicaties.

Daarnaast zou ik graag AB testing in de applicatie inbouwen. AB testing is het concept dat een selecte groep van de gebruikers een nieuwere versie te zien krijgen dan een andere groep. Zo kan de uitrol van een nieuwe versie geleidelijk aan gebeuren en indien er een bug zit in de software is deze enkel zichtbaar voor een klein deel van

de gebruikers. In mobiele applicaties is dit een uitdaging, omdat dynamisch code laden niet mogelijk is. Ik zou willen onderzoeken wat de mogelijkheden hierin zijn en welke alternatieven er zijn om toch aan AB testing te kunnen doen en deze dan combineren met de monitoring library.

Ten slotte zou ik willen onderzoeken of het mogelijk is om een plugin te bouwen voor Xcode, de ontwikkelingsomgeving voor het iOS besturingssysteem. Deze plugin zou de developer kunnen helpen met het ontwikkelen van de library door bijvoorbeeld automatisch aan te geven welke stukken code het traagste zijn en hoe traag.

Bijlagen

Bijlage A

IEEE Artikel

The importance of tracking mobile applications

Stef Van Gils

Monitoring of mobile applications is a relatively new concept, there is not much research done to explore this kind of monitoring. However, the data collected from the monitoring can be used to improve the application.

In this article our vision on monitoring mobile applications is presented. The article explains how the design of a monitoring library could look. This design is based upon the Metrics library ([12]). Our design gets compared to libraries that exist at the moment and an overview of the differences is made.

I. INTRODUCTION

WITH approximately 1 out of 5 people using a smartphone ([4]), the mobile phone has experienced a revolution. Smartphone applications are used daily and a key property of them is that they have to be responsive almost all the time. This is not easy to accomplish for developers, some delays are only visible when the application is in production and not in the testing phase. At this moment there is no straightforward way to discover this problem. A possible solution is presented in this paper with the help of a monitoring library.

The article starts with giving a global overview of mobile applications and smartphone usage. With this view it is possible to talk about the importance of the monitoring mobile applications, which is best to be kept in mind in the following sections.

The key part of this article is which parts of the application should be tracked. To make the image a bit clearer, this is split into two sections: the monitoring of performance and the monitoring of the usage of the application.

At the end the monitoring library gets evaluated with the lessons that are learned from the previous sections.

II. THE IMPACT OF MOBILE APPLICATIONS

In 2015, the number of smartphone users has grown to 1.859 billion users. ([4]). That means that approximately 1 out of 5 people uses a smartphone. Every smartphone consists of a number of mobile applications (referenced to as apps further). The Apple App Store consists of approximately 1.5 million of them, which means that there are a lot of apps already in the open ([5]). This number also means that there exists more than one app for one purpose.

The impact of the smartphone can be seen as enormous. Adults spent more time on their smartphone than on any other digital device ([6]). The smartphone now accounts for one third of the internet usage ([7]). This is very important to keep in mind when reading the rest of the article.

All these properties lead to a couple of questions:

- As developer, how do you distinguish yourself from other similar apps?
- What is the impact of app performance?
- Does a user use my app as intended?

The answer to all these questions can be found in the next sections.

III. THE IMPORTANCE OF MONITORING

As developer it is desirable that the software that you write is not slow and works as it should work. To be sure that this is the case, a developer writes tests and concludes that his code works or that he should alter his code. However there are many cases that testing alone is not sufficient. One case is that if data gets sent or received from a server and the server is overloaded, the app appears slow, but the malefactor is the server. This is one example showing that sometimes it is not possible to foresee the bottlenecks of an application. To be able to find out where the issues lie, it is necessary to monitor the application. If the application is monitored correctly, the developer should be able to identify the bottlenecks in the application and solve them in a following version.

Not only is it important to monitor the performance of an application. In many cases a developer or owner of an application wants to know which parts of the application gets used mostly by the users so they have an idea where to improve or alter the application.

A. Performance

As discussed before, performance is a key issue in developing mobile applications. Three out of the top ten of most mentioned complaints for mobile applications are performance complaints, namely: *Resource-heavy*, *Slow or lagging*, *Frequent crashing* ([8]). These complaints are the problem of the developer and should be solved to improve the application. These problems could be discovered by monitoring the application. Without monitoring the application for performance issues, the developer has to read the reviews by users to identify the problems, which is in most cases almost impossible to do because of the lack of info that is in the review. If the application is monitored as complete as possible, the developer is able to identify the correct spot in the code where the performance issue lies and solve them in the following version.

B. Usage monitoring

There are two reasons for monitoring the usage of a mobile application, namely: to discover which parts of the application gets used and in combination with performance monitoring to find bottlenecks in the application.

The group of persons that wants to discover which parts of the applications the users use are the owners of the application. The main reason they want to know this information is money. If a part of the application isn't used frequently, it is a waste of

money to solve bugs in it or invest money to improve this part. The money should be invested in parts that get used frequently to improve the user experience or not even at all. If we look back at the list of most mentioned complaints ([8]), we can see that there is a complaint *Feature removal*. This means that the owners of the application decided the feature gets used rarely. If it was possible to monitor the usage of this feature, the owners wouldn't make that mistake of getting that feature removed. This leads to less disappointment from the users.

Developers could use this kind of information too. The situation gets a lot clearer with an example. Take the situation where the app gets slow when the usage passes n users simultaneously, because the load on the server is too high. If this happens frequently, the developer should increase the server capacity, but if this happens almost never, a solution could be to rent a server when the number of users gets to a critical number. These decisions could save a lot of money.

These situations indicate that there is a need for monitoring mobile applications. A monitoring library could improve the application and the business owning the application in many ways.

IV. WHAT TO MONITOR

Now that there is a clear picture why monitoring is important, it is necessary to check what parts could be monitored. Like the previous section there is a separation of performance monitoring and usage monitoring. This section covers the most important scenarios, of course there are other scenarios possible.

A. Performance Monitoring

The importance of the monitoring of the performance of an application is already clear. In this section it is explained which properties of performance could be monitored to discover problems.

1) CPU Usage

CPU usage on a mobile device is a complicated property. On a mobile device there are a lot of background tasks that are competing with your application for CPU time. It would be wrong to take the percentage of CPU usage, because it works with peaks and differs from device to device. The solution is to use the time an operation needs to complete. The variability on this is less than it on the percentage of CPU usage. It should be somewhat the same over different devices of the same type. CPU usage is a useful parameter to measure, because it measures how much CPU the application uses at a time. If the CPU usage of the device approaches 100%, the device loses its responsiveness and the user notices this, because there would be a significant delay between an action performed by the user and the response of the application. So it is important to keep the CPU usage down to keep the application and the device running smooth.

2) Network Speed

The network speed is different from the internet connection speed. The network speed depends on two different subsystems, namely: the speed of the back end and the speed

to retrieve data from the back end. It is important to monitor the network speed, because almost every application uses the internet to send data to a server or retrieve data from the internet. The only thing a developer can't manage is the speed of the internet connection of the device, but the processing speed of the back end is an important thing to manage.

a) Speed of back end: The speed of the back end is the time the back end application needs to process the request. If this process takes a significant time, this delay flows through to the mobile application and gets noticed by the user of the application. It is important to keep this time as low as possible. It is useful to monitor this speed at the back end itself and discover possible bottlenecks or speed issues.

b) Speed to retrieve from back end: There are other issues that can slow the performance of the back end, for example: the server's application server is configured badly, the server can't handle all the requests, etc. The tracking of this happens in the application. Note that this only detects a problem in the server, not where the problem exists. This is how we detect a problem, a system administrator should look further into this at the server side.

The tracking of the network speed consists of tracking the time it needs to retrieve or send the content to the server. It is important to know the type of connection the user has, because there is a speed difference between the different connections (4G, 3G, WiFi, ...). Combining this delay with the delay measured at the back end (discussed in previous paragraph), it is possible to detect a problem at the server side.

3) Battery consumption

If an application uses a lot of battery, the user of the application will notice this. So it is important to monitor the battery consumption, however this is almost impossible. The battery consumption consists of a lot of different, sometimes hard to track, variables: CPU usage, network usage, other applications running, room temperature, etc. It is possible to track the CPU usage and eventually the network usage by measuring how many times there is a network activity in the application and by exploring how long the network interface is used on average with one network activity. It is, on mobile platforms, not possible to discover how many other applications that are running and their impact on the battery of the device and the influence on the application. A battery from a mobile phone loses its capacity faster when the temperature is higher ([13]), so this also has an impact on the battery consumption. However it is currently not possible to measure the room temperature with current smartphones.

B. Usage Monitoring

Besides monitoring the performance of an application, it is possible to monitor how the application is used by the users of the application. It is important for the owners of the application can have a wrong picture of how the users use the app. This could lead to wrong decisions and make the app worse from the users' side.

There are a lot of ways to track the user's behaviour. Here are some examples that are useful to monitor:

- The buttons the user clicks on and how many times
- The screens the user visits and how many times
- The number of search results
- etc.

These are just a couple of examples of what is possible to monitor, but it makes the picture clear.

V. CHARACTERISTICS OF A MONITORING LIBRARY

The challenge is to merge all these requirements into one library. *What is needed to be able to monitor the properties mentioned above?* There is no clear answer for this, every monitoring library has a different architecture. The monitoring library described below is based upon a Java monitoring library, namely metrics ([12]). The library has five monitoring aids, namely: **Counters, Gauges, Timers, Histograms and Meters**. With these, it is possible to monitor most of the requirements of a developer.

A. Counter

A counter is a concept that lets the developer count some property. For example it can be used to log how many times a button is clicked. The counter object should offer the operations to increase and decrease the value of the counter to complete the concept.

B. Gauge

A gauge is a simple object that just consists of a value that is returned by some property. Each gauge can only contain one value to keep it simple. The gauge object can be used for example to log the amount of values returned by a search.

C. Timer

A timer is a concept to track the duration of an event. This can be any event that is available to track. The timer can be used for example to track the duration of a method or how long it takes to retrieve content from the internet, etc. A timer object saves the current time stamp on creation and compares it to the time stamp collected when the timer is stopped. The time between these two is the time the event needed to complete.

D. Histogram

The concept of a histogram is the same as in the world of statistics: to measure the distribution of values. The values are divided over a number of buckets and represented on a bar graph so the distribution of the graph can be seen immediately. The histogram object in the library should only collect one value at each interval to keep it simple.

E. Meter

A meter measures the speed at which an event occurs. An example of a meter is the Unix load calculator ([9]) which calculates the average load in three numbers: the one-, five- and fifteen-minute load average. The meter object can be used to collect such data.

This data should be collected where and when the developer chooses to collect it. It is necessary to collect other data as well, because with this data alone it is impossible to make a correct decision. There is a need to collect metadata as well to support the original data. The types of metadata that should be collected (at least) is: **the name of the application, the version of the application and the device used**. These are the necessary types of metadata that should be collected. There is another type of metadata that is valuable to collect, namely: **the connection type**.

Why are these types of metadata valuable?

A. Application Name

The name of the application is an absolute necessity to collect, because it distinguishes the tracked values from one application to another one (it might be useful to collect more properties like this to ensure no overlap between two applications). If the application name isn't collected, the dashboard doesn't know which values to collect for a particular application and it won't work as intended.

B. Version of the Application

The most valuable type of metadata to collect is the version number. It distinguishes the results from previous versions to results from current versions. The following example will make it clear why it is so important:

Imagine your application has a version A and version B and version B is higher than version A. With the library we discovered that there was a piece of code that slowed the application down in version A, but the developers tried to solve this in version B. After the release of version B you're not completely sure the solution solved the slowness of the code, so it would be helpful to see if version B solved the problem.

So it is important to distinguish the results from version to version to be able to make a correct conclusion.

C. User's Device

The user's device is another valuable type of metadata to collect. This type only has value in tracking performance and not in tracking the usage of the application. *Why?* The usage of a button or visiting a screen doesn't depend on the type of device the user is on, because the design of the application (mostly) isn't different from one device to another. The performance of the app changes in all aspects

from device to device. The explanation is that in newer devices the hardware is improved against the older devices (like cpu, network antennas, batteries, ...). This makes it necessary to collect the user's device in the metadata. It has to be relied upon to make a decision regarding the performance of an app.

D. Connection type

Smartphones have (until now) in most cases four types of different connection types, namely: WiFi, 4G, 3G and Edge. WiFi and 4G are sometimes equally fast, but 3G is slower and Edge even slower. The only use case in which the collection of this type of metadata is useful is when a developer wants to track the speed of the network. In that case, if we don't collect the connection type, the results aren't useful, because we merge all the connection types together in one value. When the values of the different connection types are separated, it is possible to get a better view of the speed of the network.

If we combine the data with the metadata, we get an overview of all the properties we wanted to track. This helps the owners of the app to make correct decisions in what to change in the app and what not to change.

VI. PROPERTIES OF THE LIBRARY

The properties of the previous section can now be combined in one library who collects all the information and sends it to a server to store it. The dashboard application can get the information of the server to generate charts and more technical information. The library should have at least the following properties: easy to use and no significant impact.

The library should have an easy to use interface and should be well documented, otherwise the developer who uses the library wouldn't be able to get the most out of it and the library wouldn't be used as intended.

The library should not have a significant impact on the performance of the application it is tracking. The impact the library could have on the performance is: slowing the app down, clogging the network interface and tremendous battery usage. When these factors have an impact, the library's usefulness decreases, because we want to track these factors with the library, not introduce them. So in that case the library would only be useful in a test environment. It is thus a necessity to evaluate the performance and the impact of the library on an application.

VII. ALTERNATIVE LIBRARIES

Tracking in mobile applications is a relatively new topic. At the moment there are only two mobile application trackers widely available that are worth mentioning: **NewRelic** and **Google Analytics**.

NewRelic ([10]) is a closed source mobile tracking library with almost the same characteristics as the library

described above. The biggest difference is that NewRelic doesn't include feature tracking. The biggest drawback of the NewRelic library is that it is closed source, which means that there is no indication which data gets sent to the server and which isn't what makes this library dangerous when dealing with privacy-sensitive data.

Google Analytics ([11]) is a tracking library that is used across multiple platforms (web, mobile, ...). The difference between Google Analytics and the library described in this article is that Google Analytics only tracks the usage of an app and not the performance of an app.

There is one library worth mentioning other than the two above, namely the **Metrics library** ([12]). Metrics is a tracking library for (non-mobile) Java applications. The reason this library is mentioned is because the core of the library described in this article is based on the Metrics library.

VIII. CONCLUSION

Mobile tracking is a relatively new topic, supported by the fact that only two decent libraries exist at the moment. This article showed that, despite the lack of choice in libraries, mobile tracking is an important research field.

The list of complaints we showed earlier shows that there is a need for tracking mobile applications. Without the ability to track the mobile application, there is no way to tell where the app goes wrong without guessing. An app owner should rely on exact data that is measured at the core of the problem to make a decision.

The smartphone market is still growing and will continue to grow. Every week an average of 4100 apps are introduced in app stores around the world (1.5 million applications in 7 years). This means that it becomes more and more important to distinguish your app from the similar apps that are available or become available. The only thing an app owner can do to keep their users and gather more users is to listen to the complaints of the users. As listed before, many of the most important complaints are possible to track with a decent library and can be solved afterwards.

Mobile tracking is a research field that should be explored further to be able to improve the mobile application field. Mobile applications, developed by third parties, only exist for about 7 years (https://en.wikipedia.org/wiki/App_store), what means that there is room for improvement. Tracking of (non-mobile) software is standard in developing software, so why shouldn't it be standard in developing mobile software.

REFERENCES


- [1] D. Van Landuyt, S. Walraven and W. Joosen, *Variability Middleware for Multi-tenant SaaS Applications: A Research Roadmap for Service Lines*, Proceedings of the 19th International Conference on Software Product

Line, pages 211-215

- [2] H. Moens and F. De Turck, *Feature-based application development and management of multi-tenant applications in clouds*, SPLC 14: 18th International Software Product Line Conference, pages 7281, 2014
- [3] F. Gey, D. Van Landuyt, S. Walraven, and W. Joosen, *Feature models at run time: Feature middleware for multi-tenant SaaS applications*, Models@run.time 14, pages 2130. CEUR-WS.org, 2014.
- [4] Number of smartphone users* worldwide from 2014 to 2019 (in millions). URL <http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>
- [5] Number of apps available in leading app stores as of July 2015. URL <http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>
- [6] Danyl Bosomworth. Mobile Marketing Statistics compilation. URL <http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/>
- [7] Alex Hern and agency. Smartphone now most popular way to browse internet Ofcom report. URL <http://www.theguardian.com/technology/2015/aug/06/smartphones-most-popular-way-to-browse-internet-ofcom>
- [8] Kerry MacLaine. Why Your App Sucks: The 20 Most Common Complaints About Mobile Apps. URL <http://appealingstudio.com/why-your-app-sucks-the-20-most-common-complaints-about-mobile-apps-2/>
- [9] Load (computing). URL [https://en.wikipedia.org/wiki/Load_\(computing\)](https://en.wikipedia.org/wiki/Load_(computing))
- [10] NewRelic. URL <http://newrelic.com>
- [11] Google Analytics. URL <https://developers.google.com/analytics/devguides/collection/ios/v3/>
- [12] Metrics. URL <https://dropwizard.github.io/metrics/3.1.0/>
- [13] BU-806a: How Heat and Loading affect Battery Life. URL http://batteryuniversity.com/learn/article/how_heat_and_harsh_loading_reduces_battery_life

Bijlage B

Poster Tracklytics



**KATHOLIEKE UNIVERSITEIT
LEUVEN**

**FACULTEIT
INGENIEURSWETENSCHAPPEN**

Master
Computer-
wetenschappen


Masterproef
Stef Van Gils

Promotor
Prof W. Joosen

Academiejaar
2015-2016

Tracklytics: een DevOps library voor het monitoren van operationele mobiele applicaties

Situering	Doelstelling
<ul style="list-style-type: none"> DevOps: continuous delivery <ul style="list-style-type: none"> => runtime monitoring Mobiele applicaties <ul style="list-style-type: none"> => 1/5 personen hebben een smartphone iOS als besturingssysteem Klachten over mobiele applicaties: <ul style="list-style-type: none"> Performance Design 	<ul style="list-style-type: none"> Ontwikkelen van een monitoring library Developers runtime operational data bezorgen van de applicatie Trade-off zoeken tussen de hoeveelheid meetpunten en de impact op de performance en de development effort



Toepassingen	Resultaten
<ul style="list-style-type: none"> Ontwikkelen van mobiele applicaties Verbeteren van mobiele applicaties Ontdekken waar performance issues zitten Ontdekken welke componenten van een mobiele applicatie gebruikt worden 	<ul style="list-style-type: none"> Prestaties <ul style="list-style-type: none"> – 20 simultane metingen doenbaar met opslaan op disk – >1000 simultane metingen doenbaar zonder opslaan op disk Developer effort <ul style="list-style-type: none"> – Niet groot genoeg om een significante impact te hebben Conclusie <ul style="list-style-type: none"> – Bruikbare applicatie voor zowel performance als developer effort – Trade-off gevonden tussen de hoeveelheid meetpunten en de impact op de performance en de development effort

Bibliografie

- [1] Angularjs-slider. <https://github.com/angular-slider/angularjs-slider>.
- [2] Chart.js. <http://www.chartjs.org>.
- [3] Metrics. <https://dropwizard.github.io/metrics/3.1.0/>.
- [4] New relic. <http://newrelic.com>.
- [5] Openstack. <http://www.openstack.org>.
- [6] Php. <http://php.net>.
- [7] Ubuntu. <http://www.ubuntu.com>.
- [8] M. Allen. Relational databases are not designed for scale. <http://www.marklogic.com/blog/relational-databases-scale/>.
- [9] Apple. App store. <https://developer.apple.com/support/app-store/>.
- [10] Apple. iphone 6. <http://www.apple.com/benl/iphone-6/specs/>.
- [11] Apple. Swift. <https://swift.org>.
- [12] Apple. Xcode. <https://developer.apple.com/xcode/>.
- [13] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, et al. Manifesto for agile software development. 2001.
- [14] BoloYoung. Angry birds. <https://github.com/BoloYoung/AngryBirds>.
- [15] J. Cito, P. Leitner, H. C. Gall, A. Dadashi, A. Keller, and A. Roth. Runtime metric meets developer: building better cloud applications using feedback. In *2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*, pages 14–27. ACM, 2015.
- [16] CocoaPods. <https://www.cocoapods.org>.
- [17] W. de Kort. What is devops? In *DevOps on the Microsoft Stack*, pages 3–8. Springer, 2016.

- [18] M. Fowler and J. Highsmith. The agile manifesto. *Software Development*, 9(8):28–35, 2001.
- [19] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1276–1284. ACM, 2013.
- [20] S. V. Gils. <https://github.com/Stefvg/Tracklytics>.
- [21] Google. Angularjs. <https://angularjs.org>.
- [22] Google. Google analytics. <http://www.google.com/analytics/>.
- [23] Y. Heisler. As ios 9 adoption reaches 79painfully old os. <http://bgr.com/2016/03/15/as-ios-9-adoption-reaches-79-most-android-users-are-still-running-a-painfully-old-os/>.
- [24] J. Highsmith and A. Cockburn. Agile software development: The business of innovation. *Computer*, 34(9):120–127, 2001.
- [25] N. Jones. Performance killer: Disk i/o. <http://www.nathanaeljones.com/blog/2009/performance-killer-disk-io>.
- [26] K. MacLaine. Why your app sucks: The 20 most common complaints about mobile apps. <http://appealingstudio.com/why-your-app-sucks-the-20-most-common-complaints-about-mobile-apps-2/>.
- [27] P. Malik. How many ios application developers are planning to "rewrite"their objective-c apps in swift? <https://www.quora.com/How-many-iOS-application-developers-are-planning-to-rewrite-their-Objective-C-apps-in-Swift>.
- [28] NetMarketShare. Mobile/tablet operating system market share. <https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1>.
- [29] J. Nielsen. *Usability engineering*. Elsevier, 1994.
- [30] Oracle. Mysql. <http://www.mysql.com>.
- [31] Statista. Number of smartphone users worldwide from 2014 to 2019 (in millions). <http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>.
- [32] E. Sverdllov. How to set up master slave replication in mysql. <https://www.digitalocean.com/community/tutorials/how-to-set-up-master-slave-replication-in-mysql>.

- [33] J. Waller, N. C. Ehmke, and W. Hasselbring. Including performance benchmarks into continuous integration to enable devops. *ACM SIGSOFT Software Engineering Notes*, 40(2):1–4, 2015.
- [34] A. I. Wasserman. Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 397–400. ACM, 2010.
- [35] Wikipedia. Devops. <https://en.wikipedia.org/wiki/DevOps>.
- [36] Wikipedia. Objective-c. <https://en.wikipedia.org/wiki/Objective-C>.

Fiche masterproef

Student: Stef Van Gils

Titel: Tracklytics: een DevOps library voor het monitoren van operationele mobiele applicaties

Engelse titel: Tracklytics: een DevOps library voor het monitoren van operationele mobiele applicaties

UDC: 621.3

Korte inhoud:

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: computerwetenschappen, hoofdspecialisatie Software engineering

Promotor: Prof. W. Joosen

Assessor: Dimitri Van Landuyt

Begeleider: Dimitri Van Landuyt