KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

DevOps: Operational data for developers

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

Academiejaar 2015 – 2016

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

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, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer 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

Lijst van tabellen

Hoofdstuk 1

Context

1.1 Relevantie monitoren mobiele applicaties

Op het moment van schrijven is het 2016, ongeveer één op de vijf mensen gebruikt een smartphone in het dagelijkse leven. 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 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 dit het geval is, 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 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, namelijk: *Resource-Heavy, Slow or lagging* en *Frequent Crashing*. Deze klachten worden vaak in reviews van gebruikers geuit. De eigenaars kunnen hieruit 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 de bottlenecks kunnen ontstaan. Hierdoor kunnen ze de exacte oorzaak van het probleem vinden. Op deze manier kan een potentiëel 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. 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. Deze metingen kunnen geïmplementeerd worden door de developer zelf of door een monitoring library. Het gebruik van een monitoring library brengt vele voordelen met zich mee ten opzichte van het zelf implementeren van een systeem. Een monitoring library is ontwikkeld met als doel het zo goed mogelijk monitoren van een applicatie en heeft dus alle diensten hiervoor in huis. De tijd en moeite die een developer in het monitoren van een applicatie steekt is significant lager dan dat de developer zelf nog een systeem moet implementeren. Dit komt omdat de monitoring library klaar is om gebruikt te worden en alle communicatie in deze library verwerkt is, terwijl een developer zelf nog deze communicatie zou moeten schrijven en een back end. Een monitoring library zorgt voor het verwerken en het weergeven van deze data in grafieken zodat er een goed beeld hiervan gevormd kan worden. Een nadeel van een externe monitoring library te gebruiken is dat de data opgeslagen staat op de servers van de externe partij en dat deze in vele gevallen niet opgevraagd kan worden bij die partij. Een ander nadeel is dat de developer de data anders of dieper wil analyseren dan dat gebeurt in de monitoring library. De eigenaars moeten dus een keuze maken tussen het ontwikkelen van een eigen, nieuw, monitoring systeem waar veel tijd en moeite in kruipt en waarvoor

voldoende infrastructuur voor moet voorzien worden (back end servers) en het in gebruik nemen van een bestaande monitoring library die misschien net niet genoeg functionaliteit voorziet of de data niet beschikbaar stelt aan de eigenaars van de applicatie.

Het monitoren van mobiele applicaties is anders dan het monitoren van websites, internet applicaties en pc applicaties omwille van een aantal factoren. Android en iOS zijn de twee besturingssystemen die het waard zijn om te vermelden. Een mobiele applicatie wordt dus meestal beiden ontwikkeld om een zo groot mogelijk aantal gebruikers te bereiken. Ontwikkelen voor de andere besturingssystemen is meestal niet rendabel voor de moeite die erin moet gestoken worden. De monitoring library draait dus maar op twee verschillende systemen, terwijl er tientallen web browsers bestaan waar websites en internet applicaties in draaien.

Indien de applicaties native worden ontwikkeld, dan zijn moet de applicatie een keer van Android en een keer voor iOS ontwikkeld worden. Dit zorgt ervoor dat er verschillende bottlenecks in de verschillende applicaties kunnen zitten. Het zou verkeerd zijn om de data van de twee applicaties samen te nemen.

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 de 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 bijna nooit 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 noodzakelijk. Enerzijds 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.

1.2 Development Scenarios

In deze sectie worden een aantal scenarios waar een developer of eigenaar er baat bij heeft om een monitoring library te gebruiken.

1.2.1 General Purpose App

Een general purpose app is een mobiele applicatie die geen game is (bv. Facebook, Shazam, maar ook een gewone camera app). De functionaliteiten van deze applicaties zijn heel uiteenlopend. Een eigenaar en een developer hebben verschillende informatie nodig. Dit wordt in de volgende paragrafen opgedeeld.

Eigenaar Een eigenaar wil dat de applicatie zoveel mogelijk gebruikt wordt en zoveel mogelijk geld opbrengt. Om gebruikers te lokken en deze ook te houden moet de eigenaar de applicatie af en toe updaten om gegeerde features toe te voegen of te verbeteren en niet gebruikte features te verwijderen. Zonder een of ander monitorsysteem is het voor de eigenaar bijna onmogelijk om te weten welke features vaak gebruikt worden en welke niet.

Indien de eigenaar weet welke onderdelen vaak gebruikt worden kan hij hierop inspelen door goed geplaatste advertenties in de applicatie in te bouwen en hiermee geld te verdienen. Een ander voordeel hiervan is dat indien een onderdeel verbeterd of veranderd moet worden en dat onderdeel zelden gebruikt wordt, dan is het niet voordelig om hier nog in te investeren. Het beste zou dan zijn om het onderdeel uit de applicatie te halen.

De eigenaar kan het design van de applicatie aanpassen om een weinig gebruikt onderdeel meer in de spotlight te zetten zodat gebruikers dit vaker zouden gebruiken.

Developer De eigenaar wil vooral weten hoe de applicatie gebruikt wordt. De developer wil vooral weten of de applicatie naar behoren werkt en er geen bugs of bottlenecks in zitten. De meeste bugs zouden er in de testfase uitgehaald moeten worden, maar er zijn bugs en bottlenecks die enkel op grote schaal zichtbaar zijn. Meestal heeft dit te maken met het ontvangen of verzenden van content over het netwerk, maar kan ook te maken hebben met het lokaal verwerken van data. Zonder een monitorsysteem is het voor de developers in vele gevallen een hele uitdaging om de bottleneck of de bug te vinden. Met een monitorsysteem kunnen de developers op bepaalde kritieke punten metingen uitvoeren. Indien er dan een bottleneck zich vormt in de applicatie, dan weten de developers op welk punt het fout loopt. Hierdoor wordt er veel tijd bespaard in het zoeken naar de bottleneck.

1.2.2 Game

De game die besproken wordt is het populaire spel **Angry Birds**. In dit spel krijgt een speler drie beurten om alle vijanden te verslaan door een constructie omver te

werpen en de vijanden schade te berokkenen. Indien alle vijanden verslagen zijn eindigt het level en krijgt de speler een score en een beoordeling van maximaal drie sterren. Dit scenario wordt opgedeeld in een scenario voor de eigenaar en een developer.

Eigenaar De eigenaar wil ervoor zorgen dat de applicatie zoveel mogelijk gebruikt wordt en dat de levels niet te moeilijk, maar ook niet te gemakkelijk zijn. Om te kijken dat de levels niet te moeilijk zijn kan de eigenaar de developers vragen om per level bij te houden hoe vaak er op de reset toets gedrukt is, hoeveel sterren de spelers gemiddeld verzamelen en hoeveel beurten er gemiddeld gebruikt worden. Hieruit kan de eigenaar dan opmaken of de levels de gewenste moeilijkheidsgraad hebben en in de toekomst deze getallen mee in rekening te nemen in het ontwikkelen van nieuwe levels.

De populariteit van een spel hangt vaak af hoe vaak het gespeeld wordt en hoe lang een spelsessie duurt. Dit kan gemeten worden met een monitoring library door elke keer dat de applicatie gestart wordt dit door te sturen naar de server en een timer te starten die stopt wanneer de gebruiker de app sluit. De eigenaar kan zo ingrijpen als de app minder (lang) gebruikt wordt dan voorheen.

Developer De developer wil vooral op de hoogte zijn van de prestaties van de applicaties en eventuele bugs vinden. Bij Angry Birds heeft dit impact op twee verschillende situaties, namelijk de periode dat de gebruiker zich in het menu bevindt en de periode dat de gebruiker het spel speelt. Het verschil hierin is wat er gemonitord wordt. In het spel zelf is het belangrijk dat de framerate niet te laag wordt. De developer zal dit dan ook monitoren in het spel zelf. In het menu is dit minder belangrijk. Hier is het eerder belangrijk dat er geen significante vertragingen oplopen in het navigeren door de menus en de verschillende collecties van levels.

Deze development scenarios tonen aan dat het gebruik van een monitoring library in mobiele applicaties niet alleen de developer, maar ook de eigenaar kan helpen in het maken van beslissingen omtrent de applicatie. Op deze manier kan er data verzameld worden wanneer de applicatie al in de handen is van de gebruikers. Dit staat tegenover een testomgeving die meestal enkel op kleine schaal uitgevoerd wordt.

Hoofdstuk 2

Architectuur

2.1 Architecturale beslissingen

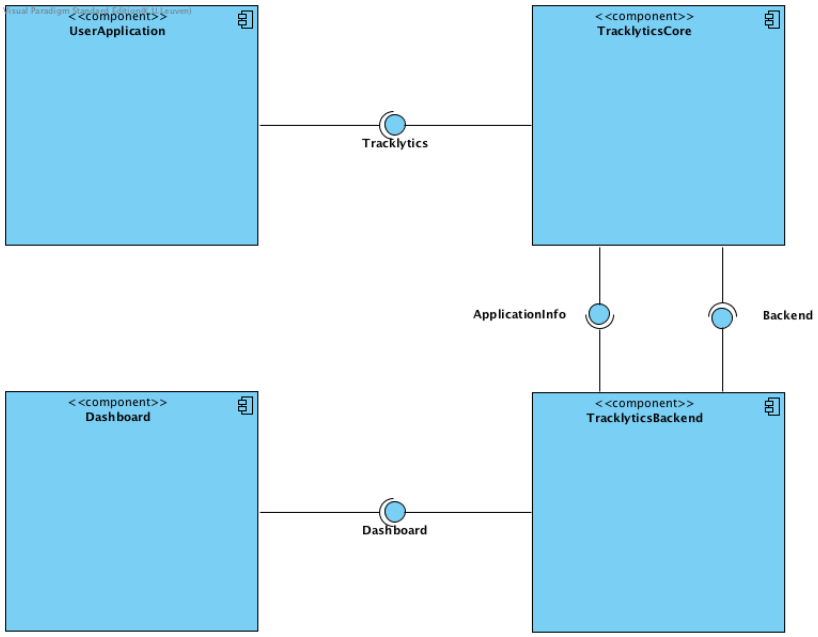
Zoals beschreven in vorige secties gaan we een library ontwikkelen om applicaties te kunnen monitoren. Tracklytics is de naam van de library en wordt in het vervolg van dit document gebruikt om de library aan te duiden.
In deze sectie worden de architecturale beslissingen die gemaakt zijn uit de doeken gedaan. Aan de hand van diagrammen wordt besproken hoe Tracklytics conceptueel werkt.

Tracklytics bestaat uit volgende hoofdcomponenten:

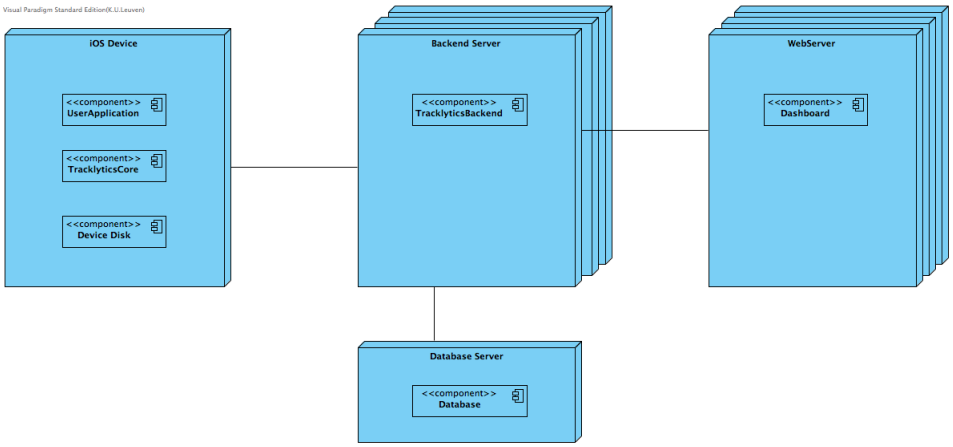
- UserApplication: De applicatie die de developer wil monitoren
- TracklyticsCore: De Tracklytics library die in de applicatie verwerkt is.
- TracklyticsBackend: De backend die de informatie verwerkt die van de TracklyticsCore komt.
- Dashboard: Een dashboard om alle informatie weer te geven.

2.1.1 Deployment Diagram

Het deployment diagram toont op welke nodes de componenten draaien. Zoals je kan zien draait de UserApplication en de TracklyticsCore op het toestel van de gebruiker zelf (een iOS device). 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.



Figuur 2.1: Component diagram Tracklytics library.



Figuur 2.2: Deployment diagram Tracklytics library.

2.1.2 Component Diagram

Het component diagram 2.1 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.

UserApplication

De UserApplication component bestaat uit de applicatie ontwikkeld door de developers. Deze verschilt per applicatie. De UserApplication stuurt data door naar de TracklyticsCore component via de Tracklytics interface. De interface wordt in de volgende sectie besproken. De developers kiezen zelf welke elementen getrackt en gemonitord worden. Deze component draait op het toestel van de gebruiker.

TracklyticsCore

Monitoring De TracklyticsCore component stelt de library voor die in deze thesis ontwikkeld wordt. De component draait op het toestel van de gebruiker. De TracklyticsCore component verzamelt informatie die hij doorkrijgt van de UserApplication component, slaat deze tijdelijk op het toestel op om deze dan later door te sturen naar de TracklyticsBackend component via de Backend interface. Deze interface wordt in de volgende sectie besproken.

De TracklyticsCore component biedt een Tracklytics interface aan. De UserApplication kan deze interface gebruiken om data te verzamelen. De TracklyticsCore stuurt deze later naar de server zoals besproken wordt in volgende sectie.

De interface bestaat uit het gebruik van 5 elementen, namelijk:

- Counter
- Meter
- Histogram
- Timer
- Gauge

Deze elementen worden verder besproken in het hoofdstuk over het klassediagram 3.2.

Data availability Smartphones vallen weleens zonder internet connectiviteit. Tracklytics zend de data over het internet naar de backend om deze op te slaan in een database. Als de internetverbinding zou wegvallen, dan zou er mogelijks belangrijke data verloren gaan. Om dit scenario te voorkomen slaan we de data tijdelijk op op de harde schijf van de telefoon. Indien de smartphone dan terug verbinding met het internet heeft gemaakt kan de data alsnog gesynchroniseert worden met de backend. De data wordt tijdelijk opgeslagen, dus nadat de data gesynchroniseerd is wordt deze van de smartphone verwijderd.

On Demand Het is de bedoeling de developer zoveel mogelijk vrijheid te geven in het monitoren van de applicatie. Om deze vrijheid te verhogen voegen we het On Demand aan- of uitzetten van het monitoren toe aan Tracklytics. Een developer kan vanuit het dashboard aangeven of de applicatie 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. Dit zorgt ook voor een prestatieboost, omdat er niet meer gesynchroniseerd moet worden naar de server en de data moet niet meer op het toestel verwerkt en opgeslagen worden door Tracklytics.

Aggregatie De data die doorgestuurd wordt naar de server zegt op zichzelf niets. Om een goede analyse weer te geven in het dashboard moet deze data geaggregeerd worden. Er zijn twee plaatsen waar deze aggregatie uitgevoerd kan worden, namelijk op de smartphone zelf in de Tracklytics library of in de backend.
De reden om de aggregatie op de smartphone uit te voeren is dat er zo minder data gecollecteerd moet worden in de backend om de data visueel voor te stellen. Er is dus een prestatiewinst om de aggregatie op de smartphone uit te voeren. Een nadeel is dat deze aggregatie meer CPU tijd van de smartphone gebruikt dan als we de aggregatie in de backend uitvoeren. Er is dus een goede afweging nodig waar deze aggregatie gebeurt.

Een functie in het dashboard zou ervoor kunnen zorgen dat de developer de optie krijgt om te kiezen of deze aggregatie op de telefoon of in de backend uit te voeren. Dit geeft de developer ook weer meer vrijheid om deze keuzes te maken.

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 de versie B geleidelijk verloopt. De gebruikers van de software worden dus in twee (of meerdere) groepen opgedeeld door een eigenschap. Deze eigenschap kan vanalles zijn, namelijk: 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.

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 uit te rollen. Dit staat pal tegenover websites die hun webpaginas van een server halen en dus bij elk vernieuwen van een webpagina helemaal anders

zijn. Een workaround van dit probleem is dat de mobiele applicatie de code van de nieuwe versie al bevat en ook nog de oude code 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 aangeroepen 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 punt is de privacy van de gebruiker. Zoals eerder al aangehaald is NewRelic TODO een closed source library, wat wil zeggen dat developers niet weten welke informatie er doorgestuurd wordt naar de backend. Dit gegeven zorgt ervoor dat er privacygevoelige applicaties deze library niet zou mogen gebruiken, omdat er gebruikersinformatie gelekt zou kunnen worden naar de backend van NewRelic.
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 langs de monitoring kant. De metadata die gecollecteerd wordt door Tracklytics bestaat uit: het type toestel, de versie, de naam van de applicatie en de bundle naam, de UDID TODO van het toestel, de datum en het type connectie waarop het toestel zich bevindt. Zo wordt ervoor gezorgd dat de privacy van de gebruiker gegarandeerd wordt en dat er toch genoeg gegevens zijn om deze te representeren op een dashboard.

Naast privacy is security ook belangrijk. Er staat namelijk een verbinding tussen de mobiele applicatie en de backend. 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 een backend bestand.

TracklyticsBackend

De TracklyticsBackend component representeert de server kant van de library. Hierop draait de code om de getrackte en gemonitorde informatie op te slaan in de database. De TracklyticsBackend component bevat dus ook de database om alle informatie in op te slaan.
De TracklyticsBackend component biedt twee interfaces aan, namelijk: Backend en Dashboard.
De Backend interface wordt gebruikt om data uit te wisselen tussen de TracklyticsCore en de TracklyticsBackend. Dit is de data die de TracklyticsCore verzameld heeft van de applicatie. De TracklyticsBackend verwerkt deze informatie en slaat

deze op in de database.
De Dashboard interface wordt door het Dashboard component gebruikt om data op te halen om te kunnen weergeven in een webpagina.

Dashboard

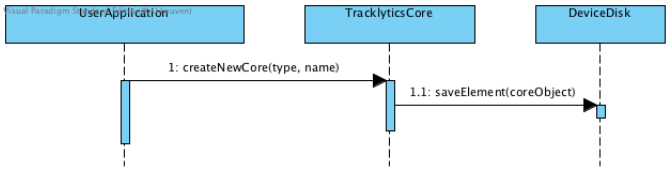
De Dashboard component dient om de verzamelde informatie door de Trashlytics library voor te kunnen stellen in een overzicht. Deze haalt de data op via de, in vorige sectie beschreven, Dashboard interface.

2.1.3 Belangrijkste flows

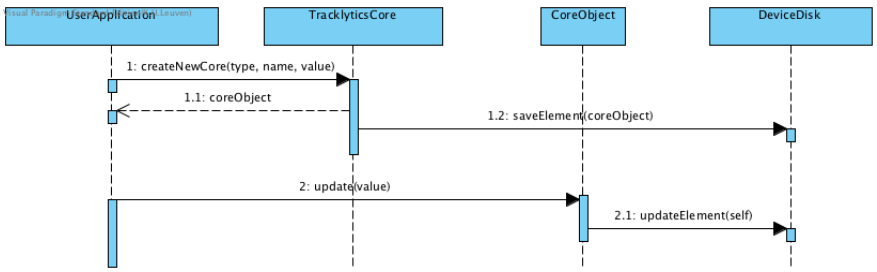
De belangrijkste flows door Tracklytics zijn:

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

Tracking en monitoring van gegevens



Figuur 2.3: Flow diagram 1.



Figuur 2.4: Flow diagram 2.

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

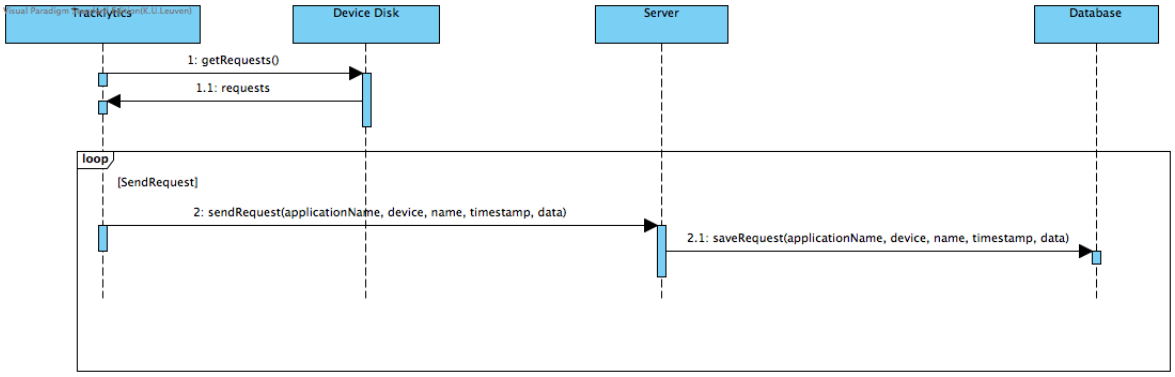
De eerste flow 2.3 toont de meest simpele flow. De developer geeft aan dat een bepaalde waarde gecollecteerd moet worden. De TracklyticsCore slaat deze informatie op op de schijf van het toestel van de gebruiker, zodat deze later verwerkt

kan worden. De verwerking van de data wordt uitgelegd in de volgende sectie 2.1.3. Deze situatie is geldig in de gevallen dat de developer data voor een histogram of een gauge wil collecteren. De uitleg over deze componenten staat in het volgende hoofdstuk 3.2.

De tweede flow 2.4 toont een uitgebreidere flow van de vorige flow. Er wordt weer een bepaalde waarde gecollecteerd en deze wordt opgeslagen op de harde schijf van het toestel. Bij deze flow wordt een Core Object terug gegeven waar verdere acties mee mogelijk zijn. Deze acties worden gebundeld in de `update` call. Deze flow is geldig als de developer data voor een Counter, een Timer of een Meter wil collecteren. De uitleg over deze componenten en de acties die op deze componenten mogelijk zijn staan in het volgende hoofdstuk 3.2

Verzenden en opslaan van de gegevens

In het volgende diagram 2.5 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 2.5: Flow diagram 3.

Hoofdstuk 3

Implementatie

In het vorige hoofdstuk is 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. 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.

3.1 Details Implementatie

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

3.1.1 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 en Objective-C. Swift is een redelijk recente taal, op het moment van schrijven is deze nog geen twee jaar oud. Omdat de ondersteuning van Swift door de meerderheid van de developers nog zeer karig is, is er besloten om de library in Objective-C te schrijven. Objective-C en Swift hebben dezelfde functionaliteiten, maar programmeren in Swift is overzichtelijker door de andere syntax. Swift is meer geïntegreerd in de IDE die Apple aanbiedt om applicaties mee te ontwikkelen. Swift is veiliger en sneller dan Objective-C. Al deze redenen pleiten om Swift als programmeertaal te kiezen. Het grote nadeel is dat Swift nog niet in veel applicaties aanwezig is en dat het moeilijker is om in een Swift applicatie een Objective-C library te integreren dan andersom. Dit is de grote reden waarom er voor Objective-C gekozen is.

De Tracklytics library slaat tijdelijk de data op op de harde schijf van het toestel. Dit heeft twee voordelen, namelijk: er gaat geen data verloren en de library gebruikt minder RAM. Als het toestel geen internet connectie zou hebben, kan de library

de data niet verzenden naar de back end en indien de applicatie dan sluit zijn alle gegevens verloren. Dit scenario is opgelost door het tijdelijk opslaan van de gegevens op de harde schijf. De data wordt pas verwijderd als de library er zeker van is dat de data succesvol naar de back end is verstuurd.

Een mobiel toestel heeft een beperkte grootte RAM geheugen en applicaties kunnen hier niet volledig gebruik van maken omdat het besturingssysteem dit geheugen ook gebruikt. Als de applicatie teveel RAM geheugen gebruikt, dan gaat heel de applicatie deze trager werken, omdat het RAM geheugen dan uitgeswapt wordt naar de harde schijf. Het is dus noodzakelijk om het RAM gebruik zo laag mogelijk te houden. Indien alle data in RAM geheugen zou gehouden gestoken worden, dan kan dit snel vollopen. Het opslaan van de data op de harde schijf verhelpt dit probleem. Het is dan wel noodzakelijk dat het opslaan van de data op schijf in de achtergrond gebeurt, omdat I/O operaties relatief lang duren.

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 onmiddelijk 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. Een library call neemt maar één lijn code in om een methode uit te voeren in plaats van twee. Dit zorgt voor minder code. De impact hiervan hangt af van het aantal monitoring punten die in de applicatie zijn ingevoerd. Dit verbetert ook de leesbaarheid van de code.

3.1.2 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.

In de Tracklytics architectuur is gekozen voor een linux distributie (in dit geval Ubuntu). Deze keuze is gemaakt op basis van de gebruiksvriendelijkheid van dit besturingssysteem. Zo is het gemakkelijk om snel 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, wat 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 gekozen om een MySQL database te gebruiken, omdat deze een gestructureerde tabellenstructuur heeft en zo de metadata gemakkelijk van de data kan scheiden. De data kunnen door SQL queries gecombineerd worden in views om de data overzichtelijk te maken. Een alternatief is een NoSQL database. Dit alternatief past niet in de manier waarop de data opgeslagen wordt, omdat deze een niet-gestructureerde database aanbiedt.

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 als programmeertaal. Met PHP is het simpel om een database connectie op te zetten en via SQL queries deze dat in of uit de database te krijgen. Een tweede voordeel van PHP is dat deze met POST data om kan. Via 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.

3.1.3 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 automatisch worden 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 is er gebruik gemaakt van AngularJS. 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. Een tweede reden waarom AngularJS in deze situatie voordelig is, is dat met AngularJS het gemakkelijk is om stukken HTML te laten 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 (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.

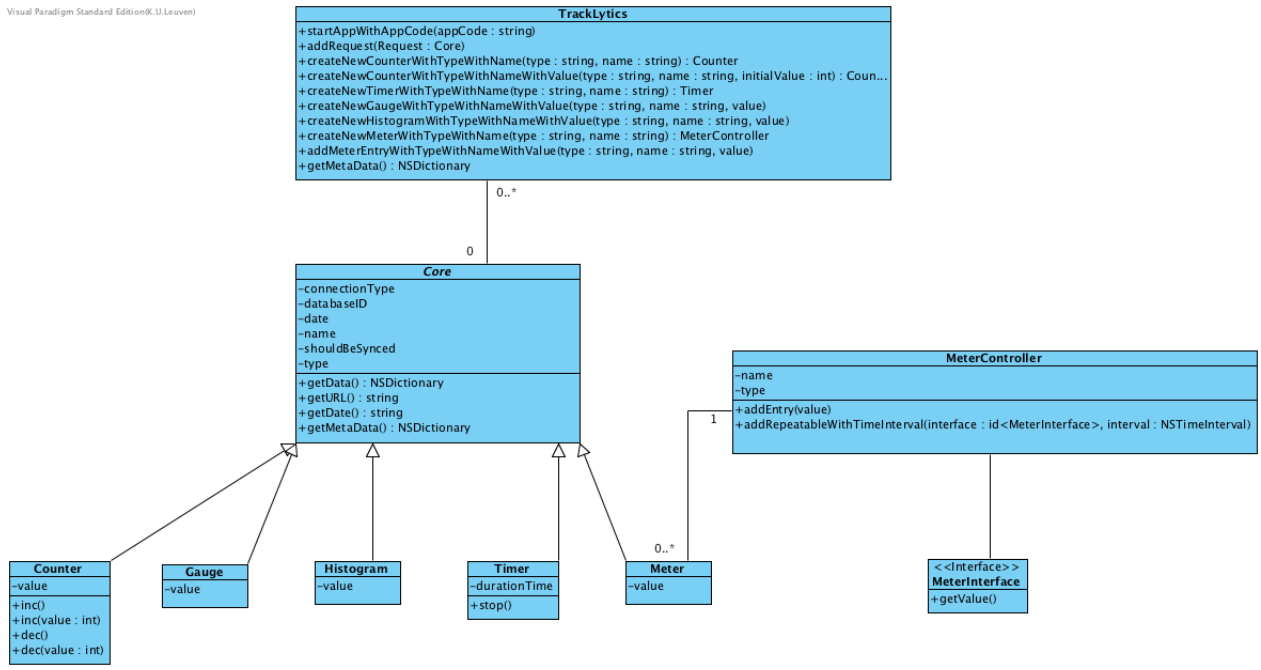
Om de histogrammen en de meters bruikbaarder te maken zijn hier sliders bij toegevoegd om bij de histogrammen het interval te veranderen en zo een kleinere dataset te gebruiken. Bij de meters kan de dataset verkleind worden door het tijdsinterval te wijzigen. Omdat het dashboard gebouwd is in AngularJS en de histogrammen en meters herhaald worden over de pagina moest er een oplossing gevonden worden die compatibel was met AngularJS. Een andere moeilijkheid was dat de slider langs twee kanten zou moeten kunnen sliden om een minimum en maximum te kunnen definiëren. Door deze twee praktische zaken is er een keuze gemaakt voor de AngularJS-slider. Deze biedt alle functionaliteit aan die nodig is om de histogrammen en meters interactief te maken.

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. De data wordt via AngularJS opgehaald en aan de juiste variabelen gekoppeld die ervoor zorgen dat deze data correct kan worden weergegeven.

3.1.4 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 beveligingskeuze 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 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.

3.2 Klassediagram



FIGUUR 3.1: Klassediagram Tracklytics library.

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

3.2.1 TrackLytics

De Tracklytics klasse is de klasse die de synchronisatie met de server verzorgt. De klasse is ook verantwoordelijk voor het creëren en opslaan van de meetobjecten. De klasse heeft enkel statische methodes, dit zorgt ervoor dat er niet telkens opnieuw een instantie van deze klasse aangemaakt moet worden. Dit bespaart enerzijds code, door niet telkens opnieuw een statement te moeten schrijven die de klasse aanmaakt en anderzijds bespaart dit RAM geheugen, omdat de verschillende Tracklytics objecten in het RAM geheugen op het toestel gaan zitten.

3.2.2 Core

Om de gemeenschappelijke elementen 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 subklasses indien nodig dit kunnen overriden.

3.2.3 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. Dit is opgesplitst omdat inc() vaker gebruikt gaat worden en zo moet er niet telkens inc(1) uitgevoerd worden. Hetzelfde geldt voor dec dat de waarde van de counter gaat verlagen.

3.2.4 Gauge

De gauge klasse stelt een object met een waarde voor. Dit object wordt aangemaakt en opgeslagen met die waarde en gesynchroniseerd naar de server. Er zijn geen methodes beschikbaar speciaal voor dit object, omdat deze waarde niet aangepast zal worden.

3.2.5 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.

3.2.6 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. Zo weten we de lengte van de gebeurtenis. 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.

3.2.7 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 2.4. In plaats van dat een Meter object wordt terug gegeven, wordt er een MeterController object terug gegeven. Een MeterController beheert de collectie van de data van Meter waardes.

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 moet er een Meter object aangemaakt worden. Dit is een van de reden 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 MeterInterface3.2.7 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.

3.3 Documentatie

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

3.3.1 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.

3.3.2 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.

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 verlangen: `dec`, ook met een optionele waarde. De Tracklytics library zorgt ervoor dat het CounterObject in sync blijft met de back end.

3.3.3 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 opgeslagen en doorgezonden worden naar de server. Om de waarde op te slaan 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 over de verschillende grafieken 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 grafieken verder opgedeeld worden per naam.
- value: De waarde die gesynchroniseerd moet worden naar de server (bv. het aantal resultaten van de zoekopdracht).

3.3.4 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`. De parameters worden nog eens opgesomd om een goed overzicht te hebben.

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.

3.3.5 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.

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 date 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.

3.3.6 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 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.

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 deeze waarde bij en synchroniseert deze naar de server.

3.4 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 aan 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.

3.4.1 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.

3.4.2 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%). 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). In contrast, 79% van de toestellen die op iOS draaien hebben de laatste versie geïnstalleerd (op moment van schrijven). 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.

3.4.3 Aggregatie

In de architectuur beschreven in deze thesis wordt de aggregatie van de data uitgevoerd in de back end wanneer het dashboard de gegevens opvraagt. Het voordeel hieraan is dat de data opgehaald kan 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 bruikbaarder bij grotere schaal. Een nadeel hieraan is dat dit extra verwerkingskracht op het toestel van de gebruiker kost en dus een grotere impact gaat hebben op de vertraging van de applicatie.

Een tweede oplossing is om periodiek (bv. één keer peer 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 groei 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.

3.4.4 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.

Bibliografie

[1]

Fiche masterproef

Student: Stef Van Gils

Titel: DevOps: Operational data for developers

Engelse titel: DevOps: Operational data for developers

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