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

Architectuur

1.1 Architecturale beslissingen

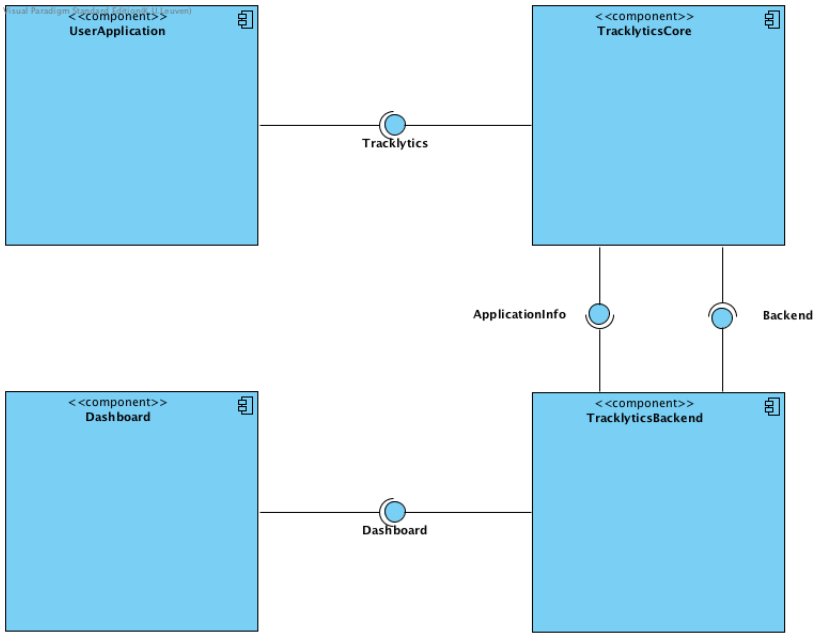
Zoals beschreven in vorige secties gaan we een library ontwikkelen om applicaties te kunnen monitoren. Tracklytics is de naam van de library en wordt in het vervolg van dit document gebruikt om de library aan te duiden.
In deze sectie worden de architecturale beslissingen die gemaakt zijn uit de doeken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

Tracklytics bestaat uit volgende hoofdcomponenten:

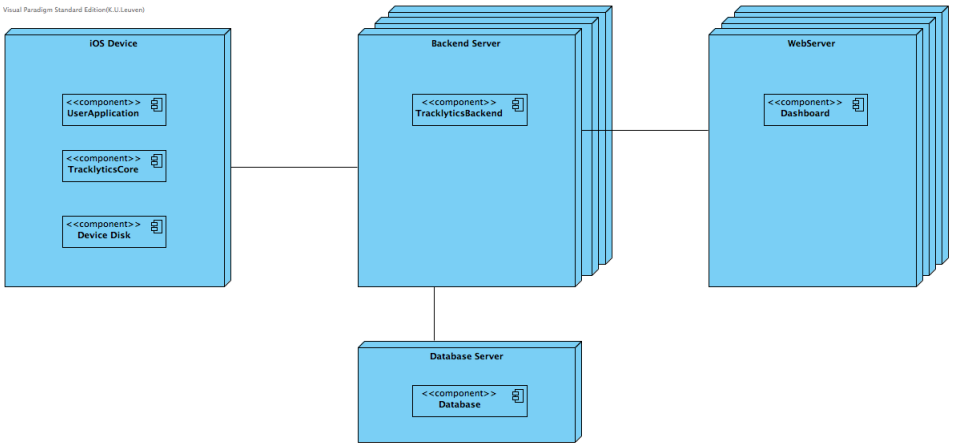
- UserApplication: De applicatie die de developer wil monitoren
- TracklyticsCore: De Tracklytics library die in de applicatie verwerkt is.
- TracklyticsBackend: De backend die de informatie verwerkt die van de TracklyticsCore komt.
- Dashboard: Een dashboard om alle informatie weer te geven.

1.1.1 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 1.1: Component diagram Tracklytics library.



Figuur 1.2: Deployment diagram Tracklytics library.

1.1.2 Component Diagram

Het component diagram 1.1 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 2.2.

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

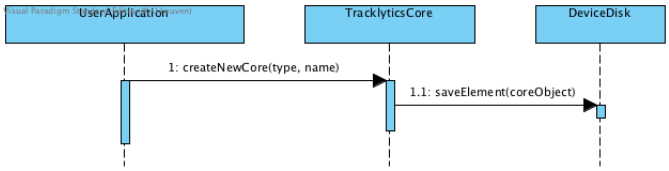
De Dashboard component dient om de verzamelde informatie door de Trashlytics library voor te kunnen stellen in een overzicht. Deze haalt de data op via de, in vorige sectie beschreven, Dashboard interface.

1.1.3 Belangrijkste flows

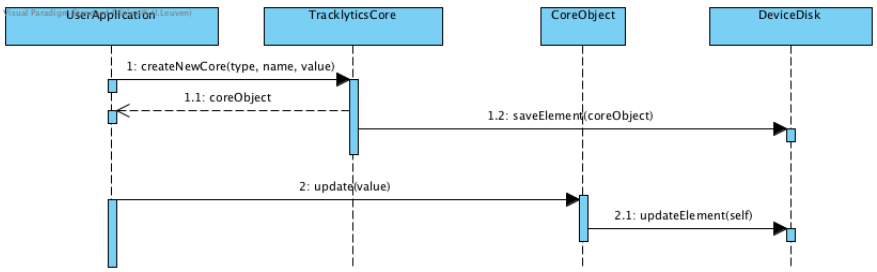
De belangrijkste flows door Tracklytics zijn:

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

Tracking en monitoring van gegevens



Figuur 1.3: Flow diagram 1.



Figuur 1.4: Flow diagram 2.

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

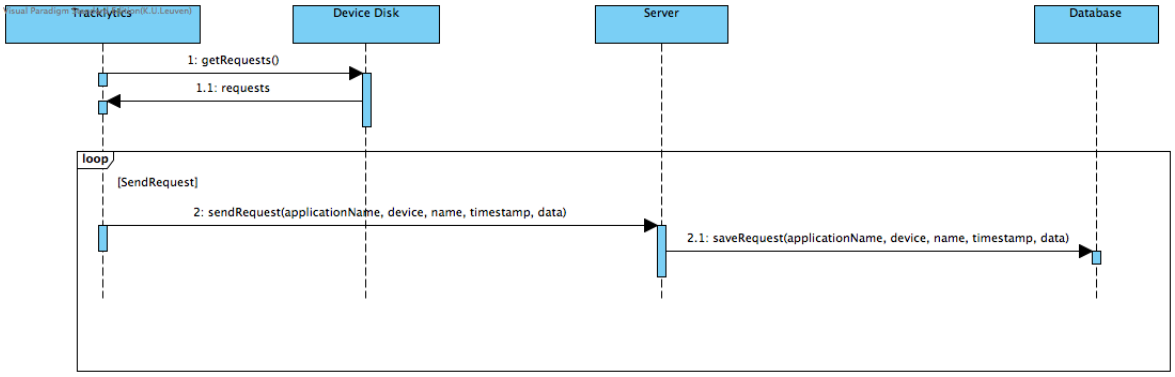
De eerste flow 1.3 toont de meest simpele flow. De developer geeft aan dat een bepaalde waarde gecollecteerd moet worden. De TracklyticsCore slaat deze informatie op op de schijf van het toestel van de gebruiker, zodat deze later verwerkt

kan worden. De verwerking van de data wordt uitgelegd in de volgende sectie 1.1.3. 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 2.2.

De tweede flow 1.4 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 2.2

Verzenden en opslaan van de gegevens

In het volgende diagram 1.5 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 1.5: Flow diagram 3.

Hoofdstuk 2

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.

2.1 Details Implementatie

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

2.1.1 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.

2.1.2 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.

2.1.3 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.

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.

2.1.4 Connectie tussen Front end en Back end

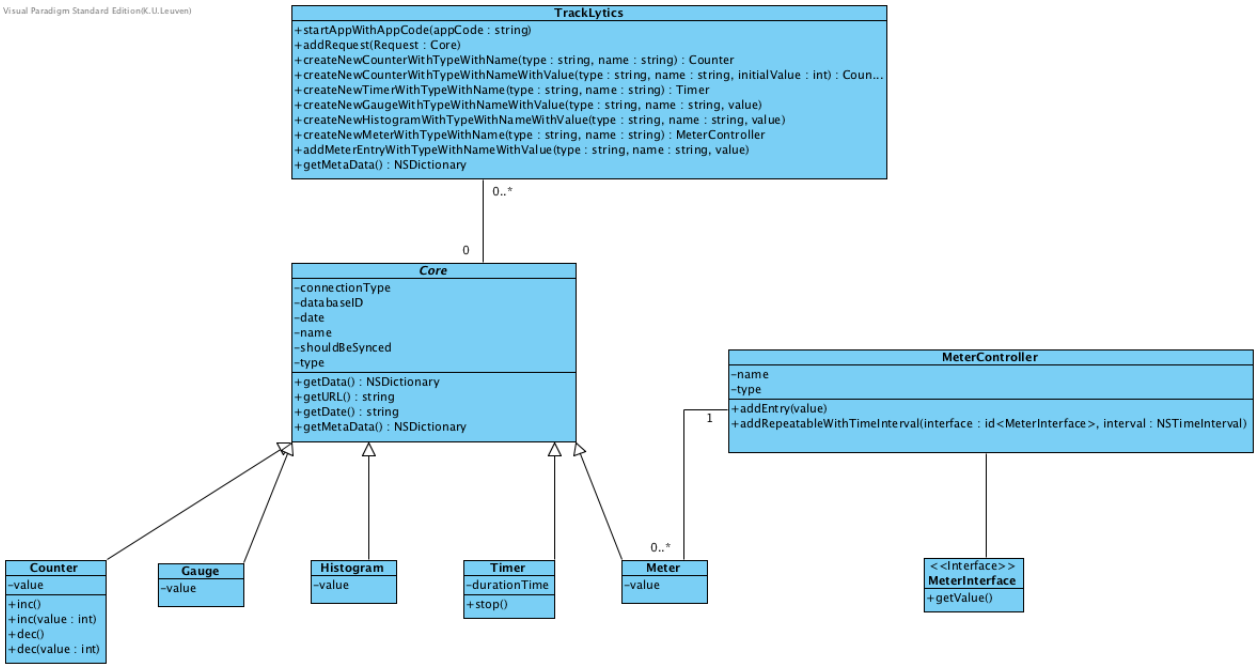
Zoals eerder aangegeven stuurt de front end (de Tracklytics iOS library) data naar de back end over een internetverbinding. Om ervoor te zorgen dat deze data veilig overgedragen wordt is ervoor gekomen om een HTTPS verbinding te gebruiken. Dit zorgt er automatisch voor dat er een veilige verbinding tussen 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.

2.2 Klassediagram

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

2.2.1 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 de library gemakkelijker te



FIGUUR 2.1: Klassediagram Tracklytics library.

gebruiken is omdat er geen object van deze klasse aangemaakt moet worden telkens de library gebruikt zou worden.

2.2.2 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.

2.2.3 Counter

De counter klasse stelt een telobject voor. Je kan bij een counter een getal optellen en aftrekken. De inc() functie verhoogt de value met 1 terwijl inc(x) de value 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 value gaat verlagen. Deze functies zijn een onderdeel van de update call zoals besproken in vorige sectie 1.1.3.

2.2.4 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.

2.2.5 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 zijn ze dus hetzelfde, enkel de getUrl() methode geeft een andere url terug.

2.2.6 Timer

De timer klasse kan gebruikt worden om de duur 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 duur 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. De stop() call is een onderdeel van de update call zoals besproken in vorige sectie 1.1.3.

2.2.7 Meter

Een meter is bedoeld om een reeks van waarden te kunnen meten. Om dit voor de developer makkelijk te maken hebben we hiervoor 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 1.4. In plaats van dat een Meter object wordt terug gegeven, wordt er een MeterController object terug gegeven. Deze keuze is gemaakt, omdat het de structuur van de library versimpeld en het de developer gemakkelijker maakt.

Meter

Het meter object is het object dat opgeslagen wordt en dat gesynchroniseerd wordt naar de server. Het object houdt de value bij en ook het tijdstip van aanmaken.

MeterController

De metercontroller zorgt ervoor dat nieuwe meters aangemaakt kunnen worden. De metercontroller bevat de naam en het type van de meter. We kiezen ervoor om meters via deze weg aan te maken, omdat dit het werk van de programmeur vermindert. De programmeur moet enkel de waarde meegeven aan de metercontroller in plaats van telkens opnieuw de type en de naam mee te geven. We hebben het de programmeur nog op een manier gemakkelijk gemaakt. De metercontroller heeft de mogelijkheid om periodiek data op te halen van een functie die de programmeur specifiëert. Zo moet de programmeur dit niet zelf gaan doen in een bepaalde methode.

MeterInterface

De meterinterface is de verbinding tussen de periodieke functie van de MeterController en het object waar de data op moet gehaald worden. Er is maar 1 functie, namelijk de getValue() die een waarde gaat ophalen. De programmeur vult deze methode dan in om de correcte waarde terug te geven.

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