Tallinna Ülikool

Digitehnoloogiate Instituut

Kolmanda osapoole teenused rakenduse logimiseks

Bakalaureusetöö

Autor: Robin Saar

Juhendaja: Jaagup Kippar

Autor: ……. „ ……..... „ 2017

Juhendaja: …….. „ ……..... „ 2017

Instituudi direktor: ……. „ ……..... „ 2017

Tallinn 2017

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....

(autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina _____ (sünnikuupäev: _____)

(autori nimi)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

---

(lõputöö pealkiri)

mille juhendaja on _____,

(juhendaja nimi)

säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas, _____

(digitaalne) allkiri ja kuupäev

Sisukord

Sõnastik

API (Application Programming Interface)	Rakendusliides, arvuti operatsioonisüsteemiga või rakendusprogrammiga määratud reeglistik mille alusel rakendusprogramm kasutab operatsioonisüsteemi või teiste rakenduprogrammide teenuseid.
TCP	Edastusohje protokoll Levinuim võrgu transpordikihi protokoll, mida kasutatakse Etherneti võrkudes ja Internetis.(Vallaste, 2016)
UDP	Kasutajadatagrammi protokoll. Sideprotokoll, mis pakub suhteliselt piiratud teenust andmete vahetamisel intentetiprotokolli (IP) kasutavasse võrku ühendatud arvutite vahel. (Vallaste, 2016)
Wrapper klass	Üldises mõistes on wrapper klassiks iga klass, mis kapseldab või ümbritseb teise klassi või komponendi funktsionaalsust.
Container	Konteiner, kuhu registreeritakse sõltuvused ning mida saab kasutada üle rakendus. Väldib klasside vahelist jäika sidet.
JSON (Javascript Object Notation)	Kerge kaaluga andmevahetuse formaat.
MVC (Model-View-Controller)	Levinud rakenduste arhitektuur, milles rakendus on jagatud kolmeks suuremaks osaks.

Sissejuhatus

Käesoleva bakalaureusetöö eesmärk on tutvustada kahte kolmanda osapoole teenust rakenduse logimiseks ning näidata teenuste poolt pakutavaid logide analüüsimise võimalusi. Koolis, programmeerimise õppeainetes, tihtipeale ei räägita logimisest, või räägitakse pealiskaudselt. Samas on logimine rakenduse pikaealisuse tagamiseks ja haldamiseks tähtis tegevus ning iga kaasaegse rakenduse kindel osa. Kui rakendus käitub mitte ettenähtud moel, siis on tarvidus näha, milles probleem on. Vigade kasutajale näitamine on äärmiselt halb praktika, seega on vaja need kokku koguda kusagile logi faili. Samuti on vaja ka koguda serverite siseseid vigu. Teiseks põhjuseks on vajalike tegevuste logimine. Näitena võib tuua olukorra, kus klient ei saa rakendust kasutades soovitud tulemust. Kui teatud võtmetegevused on logitud, saab minna ja uurida, mis läks valesti ja miks. Korralikult logitud rakenduse puhul kajastub viga logides. See on kindlasti kiirem variant, kui hakata sama protsessi läbi tegema, mida klient tegi.

Siiani logivad enamikud rakendused oma logifailid otse serveri kettale. See on muidugi parem, kui mittemidagi, kuid kätkeb endas mitmeid ohtusid. Esmalt on oht, kui serveriga juhtub midagi. Sellisel juhul kaovad nii rakendus, kui ka logid. Siinkohal muidugi tuleb mängu serverite varundamine ning varundi pealt taastamine. Samas võib juhtuda, et rakenduses toimunu, mis põhjustas serveri tõrke, jäi logimata. Sellisel juhul ei saa kindel olla, mis põhjustas selle tõrke. Siinkohal on kolmanda osapoole teenus lahenduseks. Rakenduse logimine toimub asünkroonselt ning võib juhtuda, et rakendus jõudis logi ennem ära saata, kui serveriga midagi juhtus. Teiseks, kõik arendajad on kunagi jõudnud olukorda, kus tekitatakse lõpmatu tsükkel. Kui see jääb mingitel põhjustel avastamata ning jõuab serverisse ja antud tsüklis logitakse andmeid, siis võib logifaili kirjete arv suureneda ohjeldamatult. Seega kasvab ka faili maht. Kui jõutakse äärmusliku olukorrani, kus eelnevalt kirjeldatud olukord jääb pikalt avastamata, siis võib rakendus halvemal juhul täis kirjutada serveri andmemahud ja sellega kaasneb juba hulk muid probleeme.

Käesolev töö on jätk autori poolt kirjutatud samateemalisele seminaritööle ning on jaotatud neljaks peatükiks. Esimeses peatükis tutvustab autor lühidalt valitud kolmanda osapoole teenuseid ja nende poolt pakutavaid võimalusi. Selles peatükis on kasutatud autori poolt seminaritöös kogutud informatsiooni, mis on esitatud kokkuvõtlikult.

Teises peatükis tutvustatakse eesmärgi saavutamiseks loodud rakendust ning samme, mis võeti kasutusele, et rakendus saaks hakata suhtlema teenustega. Selles peatükis tuuakse välja vaid tähtsamad osad rakendusest. Kogu koodibaas on kättesaadav autori Githubi repositooriumist, mis asub aadressil https://github.com/SaarRobin/TODOapp. Antud aadressil on viidatud ka toimivale rakendusele autori veebilehel.

Kolmandas peatükis näidatakse teenuste poolt pakutavaid võimalusi logide analüüsimiseks ja kokkuvõtete tegemiseks. Teenusepakkujate võrdlemiseks tehakse samasugugused päringud samade baasandmete pealt.

Neljandas peatükis analüüsib autor läbi tehtud protsessi ning annab oma arvamuse teenustest ning nende poolt pakutavatest võimalustest. Lisaks tuuakse välja ka tekkinud muljed ja raskuskohad.

1. Teenuste tutvustus

Kolmanda osapoole teenuseid rakenduse logimiseks on mitmeid ja neid tuleb iga päevaga aina juurde. Enne kui kasutama hakata mõnda kolmanda osapoole teenust, tuleb kindlasti selgeks teha, kas teenus võimaldab hallata logisid vajalikul moel. Kõik turul olevad lahendused ei ole kindlasti mõeldud suurtele rakendustele. Esmalt tasub uurida, lugeda ja proovida, ennem kui mõne teenusepakkuja peale kindlaks jääda. Käesolevas töös kasutab autor kahte teenusepakkujat, Logentries ja Loggly. Need said valitud, kuna nad on turul suurte teenusepakkujate seas ning autor on eelnevalt nendega kokku puutunud. Mõlemis teenuses on autoril registreeritud tasuta konto, ning töö teostatakse just tasuta pakutavate võimalustega.

1.1. Logentries

Logentries on Dr. Dublinis Viliam Holubi ja Dr. Trevor Parsonsi poolt loodud tarkvara ja teenuse pakkumise ettevõte (Techcrunch, 2012). Logentriese tehnoloogia võimaldab koguda ja analüüsida logisid, ning on üsnagi lihtsalt kasutatav. Lisaks on võimalik hiljem pärida kokkuvõtteid, statistikat ning seda kõike näha visuaalsel kujul. Kõik logid on olemas tsentraalsel kujul, ehk olenemata sellest, kus serveris, arvutis või domeenis rakendused on. (Logentries, 2016)

Ettevõte pakub mitmeid erinevaid pakette ning ka tasuta paketti väikeste mahtude tarvis. Tasuta paketi puhul on võimalik üles laadida 5GB infot kuus, ning andmeid talletatakse 7 päeva. Kõige suurema standardpaketi puhul on võimalik üles laadida kuni 150GB kirjeid ning seda talletatakse 30 päeva. Lisaks kõigele on võimalik nendega koos kokku panna ettevõtte jaoks kohandatud versioon. (Logentries, 2016)

1.1.1. Tähisega TCP

Üks võimalustest on token-based ehk tähisega logimine. Teenuses on võimalik ära määrata erinevad logide kogumikud, mis näiteks koosnevad erinevate rakenduste logidest, või ühe rakenduse alamosadest. Kui luua uus kogumik, on võimalik selle kogumiku jaoks luua uus ja unikaalne tähis, mille kuju on 2bfbea1e-10c3-4419-bdad-7e6435882e1f. Seda

nimetatakse UUID (Universally unique identifier). Kasutades seda tähist logikirjes, saab teenus aru, millisesse kogumisse see hiljem paigutada. See lahendus sobib rakendustele, millest saadetakse logisid erinevatesse kogumikesse või millel võib IP aadress muutuda, ning seda ei saa kasutada identifitseerimiseks. Selle lahenduse kasutamiseks on vaja teha TCP ühendus aadressil data.logentries.com, kasutades porti 80, 514 või 1000. Juhul kui kasutatav võrk ei ole usaldusväärne, on võimalik teha krüpteeritud ühendus samale aadressile, kuid kasutades porti 2000 (Logentries, 2016). Tähisega logide eelisteks on:

- saab lihtsalt saata suunatud logisid
- saab saata mitu logi rida, mitmest allikast ühele tähisega kogumikule
- saab lihtsalt luua ja organiseerida tähiseid iga logi kogumiku jaoks
- võimaldab laialdaselt oma logi struktuuri kohandada

1.1.2. TCP ja UDP

TCP ja UDP on teine Logentriese pool välja pakutud lahendus keskkonnaga suhtlemiseks. Selle lahenduse kasutamiseks tulen samuti luua TCP ühendus aadressil data.logentries.com, nagu ka eelmise lahenduse puhul. Erinevus seisneb selles, et antud ühenduse puhul määratakse keskkonnas igale ühendusele kindel pordi number. Lahendus on siiski kasutatav vaid rakendustes, mille IP aadress ei muutu. Nimelt on IP aadressi ja prodi numbri kombinatsioon iga rakenduse puhul selle identifikaatoriks. See tähendab ka seda, et sellist edastuse viisi saab kasutada juhul, kui logi kirjed saadetakse *Syslog daemon*i kaudu. Protsess algab tuvastusega, mis kestab maksimaalselt 15 minutit ning mille käigus oodatakse esimest TCP ühendust või esimest esimese UDP paketi saabumist. Turvalisuse tõttu on kestvuse piiranguks 15 minutit, et tuvastus ei oleks avatud igavesti. Juhul, kui võrguühendus on ebaturvaline, saab kasutada ka krüpteeritud suhtlust. Selleks tuleb antud pordi numbrile liita juurde 10000 ja teha ühendus eelnevalt nimetatud aadressi asemel aadressile api.logentries.com (Logentries, 2016).

Erinevaid logi ridu või sõnumeid saab sildistada, et hiljem oleks logisid lihtsam analüüsida ja koondada. Lisaks saab määrata teatud siltide puhul, nagu näiteks “error” sildi korral, et sellest antaks kliendile märku. Siis saab klient koheselt teada, kui mõnes rakenduses midagi valesti läks. Lisaks on võimalus tellida märguandeid juhul, kui logides tekib mingisugune anomaalia. Näiteks kui logi parameetrid väljuvad kasutaja poolt ette seatud piiridest, tekkib logides teatud muster või muutub logi mingiks ajaperioodiks tegevusetuks.

Selleks, et passiivsuse teade liiga tihti ei korduks, on võimalus veel määrata, et sellest antakse teada ainult teatud arv tunnis või päevas.

Logentriese logisid saab kasutada ülesannete jagamiseks. Logile saab juurde panna meeskonna märkuse. See on hea funktsionaalsus, kui mõni rakendus on veel arendamise järgus. Märkustega saab logisid kommenteerida, saab määrata veateate mõnele arendajale või panna juurde, et probleem on töös või juba lahendatud (Logentries, 2016).

1.1.3. Datahub

Suurte rakenduste puhul, kus on kasutusel erinevad kasutajad ja nende autentimine, tuleb mõnikord maha logida ka salastatud infot. Nagu näiteks need samad isikuandmed, mis ei tohiks kunagi jõuda avalikku ruumi. Samas on vajalik ka seda infot hiljem analüüsida ja selle põhjal kokkuvõtteid teha. Logentries pakub selle probleemi lahendamiseks välja Datahub ja DataLock lahendused.

Datahub on keskne andmete hoida, mis on rakenduse tootja käes ja hallata. Kõigepealt jõuab logikirje Datahubi, kus määratud reeglite kohaselt krüpteeritakse salastamist vajav logi kirje osa. Peale krüpteerimist saadetakse logi Logentriese keskkonda. Keskkonda jõudnud logikirjes on salastatud info kohal genereeritud hash. Töötlemata informatsioon on endiselt talletatud kohalikku DataHubi ning vajadusel saavad volitatud isikud neid andmeid töödelda. (Logentries, 2016)

DataLock on eelnevaga kaasnev laiendus, millega saab määrata isikud, kellel on õigus näha filtreerimata informatsiooni. Samuti vastutab see laiendus selle eest, et kui volitatud kasutaja Logentriese keskkonnas hakkab kirjeid analüüsima, siis tema näeb salastatud infot. Laiendus saab aru, et tegemist on selle kasutajaga, ning keskkonnas oleva info ja kohalikus DataHubis olev info sünkroniseeritakse. (Logentries, 2016)

Kuigi valik informatsiooni krüpteeritakse, ei tähenda see, et ülejäänud seda infot ei saaks analüüsida või töödelda. Endiselt saab teha Logentriese poolt pakutavaid funktsioone, kuid seda siis juba krüpteeritud väärtustega. Seega ei tähenda info mitte nägemine seda, et inimene, kes logisid analüüsib, ei saaks teatud hulka kirjeid töödelda.

1.2. Loggly

Loggly on logide analüüsimiseks ja käitlemiseks loodud pilveteenus, mis sai alguse 2009 aastal. Tegemist on kolme mehe poolt loodud ettevõttega, mille mõte oli lahendada ettevõtete operatiivsed probleemid. 2013 aastal anti välja uuendatud versiooni oma algsest lahendusest, mis võimaldas koguda logisid syslogi kaudu ning pakuti võimalust kasutada graafilist kasutajaliidest. Ettevõtte põhisõnum on, et DevOps, SysOps ja teised insenerid ei peaks muretsema logide pärast muretsema ja neid analüüsima.

Nagu ka Logentries, pakub Loggly erinevaid pakette ning ka tasuta paketti, mis jääb igavesti tasuta. Tasuta paketi puhul on nad valmis võtma 200MB päevas ja talletama seda 7 päeva. Kõige suurem pakett võimaldab saata kuni 75GB andmeid päevas ja neid talletada üle viieteistkümne päeva. Samuti pakub ka Loggly võimalust kokku leppida erilahendusi, kus andmemahud võivad olla veel suuremad ja talletamise periood veel pikem. (Loggly, 2016)

1.2.1. Andmete kogumine

Loggly kasutab andmete kogumiseks juba olemasolevaid ja avatud standardeid, nagu näiteks syslog ja HTTP. Seda selleks, et klient ei peaks installima suurt tarkvara igasse arvutisse. Arvestades, et *native*-lahendusega rakendus võib olla tuhandetes seadmetes, ei ole logide kogumiseks lisarakenduse installimine mõtekas. Veebipõhiste rakenduste puhul ei oleks probleem niivõrd suur. Samuti pole vahet, mis keeles rakendus on kirjutatud. Kui rakendusest on võimalik logida, siis on seda võimalik ka Logglysse logida (Loggly, 2016).

1.2.2. Andmete vastu võtmine, kirjete analüüs ja otsing

Loggly ise ütleb andmete vastuvõtmise kohta, et siin neelatakse andmeid. Selles protsessis saab logi kirjetesse lisada märkusi ja muud metadatat. Teenus võimaldab algsetele logikirjetele lisada tuletatud väljasid, mis tähendab, et enne andmete talletamist suudetakse juba teha arvutusi ja kokkuvõtteid. Siin faasis tehtud töö tulemus on näha hilisemas analüüsimise järgus.

Kirjete analüüsimisel pakutakse reaalajas analüüsitud andmeid. Iga kord, kui logikirje lisatakse, arvutab ja analüüsib keskkond selle info läbi ja pakub seeläbi võimalust kiirelt reageerida sündmustele. Lisaks paigutab Loggly väga kiirelt kirjed ka kataloogidesse ning seeläbi on andmed sisuliselt koheselt kättesaadavad ning kasutatavad. Selleks, et kogu asi kiirelt ja sujuvalt toimiks, on Loggly välja töötanud *Dynamic Field Explorer*i. Sisuliselt tähendab see, et analüüs ei alga kunagi tühjalt lehet. Keskkond pakub kasutajatele juba eelnevalt summeeritud tulemusi, logikirjete struktuuri ja määratud märksõnade esinemise sagedust. See kõik toimub keskkonna poolt loodud intelligentse algoritmi läbi, mis eeldab, mida kasutajad näha soovivad. Peamiselt on Loggly loonud selle võimaluse, kuna kasutajad tihtipeale ei tea, millisest kohast andmeid analüüsima hakata. (Loggly, 2016)

1.2.3. Monitooring, reaalajas häired ja anomaaliate avastamine

Loggly keskkonnas on peamine rõhk pandud just sellele osale. Seda selleks, et andmete kogumine ei oleks tulutu tegevus. Igat päringut ja kogutud andmeid saab muuta graafikuteks ning jälgida reaalajas visuaalselt. Samuti on graafikuid võimalik asetada üksteise peale, mis loob võimaluse korraga hallata suurt info mahtu. Lisaks kuvatakse kasutajale reaalajas häireid. Teenusesse saab kindlaks määrata, millistel juhtudel kuvada häiret. Häire puhul saab paika panna teatud logikirje tüübi esinemist, kindlad künnised, mille ületamisel antakse haldajale märku või muude parameetrite järgi paika pandud piirid, millest väljumisel tuleb teavitada. Häireid kuvatakse muidugi Loggly keskkonnas endas, kuid lisaks on võimalik seadistada häirete kuvamine muudesse teenustesse, nagu näiteks Slack, Hipchat ja PagerDuty. Loggly on oma teenusesse sisse ehitanud iseõppiva algoritmi, mis õpib logide normaalset mustrit (Loggly, 2016). Võib juhtuda olukord, kus tekivad logikirjed, mis ei vasta mustrile ja ei ole määratud ka häirena. Selleks näitab keskkond koheselt haldajale ka logisid, mis ei vasta antud mustrile ning tänu millele võib kasutaja varakult märgata tekkivat viga. Kui näiteks on tekkinud muster, et andmeid sünkroniseeritakse mingi aja tagant API pihta ning järsku seda enam ei toimu, saab haldur tekkinud anomaaliast kiirelt aru. Mõndade rakenduste puhul, nagu näiteks kassasüsteemid või veebipoed, võib andmete sünkroniseerimine olla väga tähtis, kuna need sünkroniseerivad näiteks laoseisusid API pihta. Anomaaliate tuvastamine on üldiselt ikka selleks, et kasutaja saaks võimalikult ruttu läbi logide teada, kus on tekkinud probleemid rakenduses.

Logikirjed on struktureeritud ja jagatud kataloogidesse, seega on võimalik ka määrata kasutajate grupp, kes teatud kirjeid näeb ja kellel häire või anomaalia kuvatakse. Kuna logidesse kirjutatakse nii serveri infot, kui ka rakenduse infot, siis ei ole mõistlik kuvada serveri haldajale rakenduses tekkinud probleeme ja vastupidi. Vastasel juhul tekiks palju müra ning analüüs oleks raskendatud ja mitte kontsentreeritud. (Loggly, 2016)

2. Rakenduse tutvustus

Eesmärgi saavutamiseks lõi autor eraldi rakenduse, läbi mille koguda andmeid, mida hiljem teenustes analüüsida. Samuti oli rakenduse eesmärk, näidata, kuidas rakenduste külge siduda nii Logentries kui ka Loggly. Lisaks näidatakse, mis on vajalik eelnevalt kasutusele võtta, et teenustesse logimise komponendid kasututsele võtta. Rakendust on kirjutatud järgides MVC arhitektuuri ning seeläbi on autori meelest rakendus lihtsamini mõistetav. Rakendus on oma loomuselt lihtne keskkond, kus inimene saab ennast registreerida, sisse logida ja kasutada seda kui oma ülesannete haldajat. Iga kasutaja saab sisestada ülesandeid, mida ta peab tegema ja kuupäev, mis ajaks see tehtud peab saama. Kasutajate poolt sisestatud infot siinse töö käigus ei analüüsita, kuna seal võib olla isiklikke andmeid. Rakendus logib maha sisselogimised, registreerimised, ülesannete lisamised ja kustutamised. Nende andmetega näitab autor hiljem teenuste võimalusi. Selleks, et tegevused maha logida, tuleb kõigepealt külge panna nii Logentriese kui ka Loggly teenustega suhtlemiseks vajalikud komponendid. Autor toob välja tööriista nimega composer ja näitab kuidas seda kasutades toimus mõlemi teenuste integreerimine.

2.1. Composer

Selleks, et sõltuvusi lihtsalt hallata on võimalik rakendustes kasutusele võtta composer, mis on PHP jaoks loodud sõltuvuste haldamise tööriist. Seda on autor teinud ka siinse töö juures loodud rakenduses. Composeri eelisteks on see, et kõik välised sõltuvused on lihtsasti kirjeldatud ning rakendust on võimalik skaleerida, ilma et peaks kaasa liigutama sõltuvusi. Selleks, et composerit kasutada tuleb see kõigepealt operatsiooni süsteemis üles seada. Selleks on lihtsaim viis kasutada käsurida. Käsureal saab alla laadida vastava installeri ning kontrollida, kas kõik toimus korrektselt.
Composeri installeerimiseks UNIX laadsetes operatsioonisüsteemides:

- curl -sS https://getcomposer.org/installer | sudo php -- --install-dir=/usr/local/bin --filename=composer
- composer

Kui eelnevad käsklused toimisid korrektselt peaks kasutajatele olema kuvatud composeri poolt loodud tervituse vaade, mis kinnitab, et composer töötab ja näitab, mis versiooniga on tegemist.

2.2. Teenuste integreerimine

Logentriese integreerimiseks on mitmeid võimalusi. Seda saab teha käsitsi faile rakendusse kopeerides või on võimalik kasutusele võtta eelpool tutvustatud composer. Autor otsustas kasutada composerit, kuna see on tänapäeval väga levinud tööriist ja lihtsustab protsessi märgatavalt. Mõlema teenuse puhul on olemas composeri pakk. Selleks, et rakendus saaks hakata saatma kirjeid Logentriese teenusesse tuleb composeris ära kirjeldada sõltuvus (Koodinäide 1).

```
"require": {
  "logentries/logentries-monolog-handler": "dev-master"
},
```

Koodinäide 1. Logentriese sõltuvuse lisamine composer.json faili

Seejärel saab käivitada käsu “composer update” ning seejärel luuakse projekti kaust nimega vendor, juhul kui seda ennem olemas ei olnud. Selles kaustas hakkavad olema kõik sõltuvused, mida projekti composeriga lisatakse. Selleks, et seda mugavalt rakenduse sees kasutada saaks, kirjutas autor sellele wrapper klassi (Koodinäide 2). Kasutades seda klassi koos containeriga on võimalik seda kasutada lihtsasti üle rakenduse.

```
<?php
namespace TODO\Loggers;

use Monolog\Logger;
use Logentries\Handler\LogentriesHandler;

/**
 * Class LoggerLE
 *
 * @package TODO\Loggers
 *
 * @author  Robin Saar
 */
class LoggerLE {

    const CLASS_NAME = 'TODO\Loggers\LoggerLE';

    /** @var \Monolog\Logger */
    public $logger;

    public function __construct() {

        $this->logger = new Logger('BakaTest');
        $this->logger->pushHandler(new LogentriesHandler('GENERATED TOKEN'));
    }

}
```

Koodinäide 2. Loodud wrapper klass Logentriese komponendile

Järgnevalt tuleb luua teenuses vastav kogumik ning saada sealt tähis (Pilt 1). Tähis tuleb lisada wrapper klassi, ning lisaks tasuks ära kirjeldada ka loggeri nimi. Seejärel on võimalik seda kasutada ka teistes klassides. Selleks tuleb teises klassis see kasutusele võtta ning seejärel ongi võimalik kirjeid teenusesse saata (Koodinäide 3).

Pilt 1. Uue kogumiku loomine

```
use TODO\Loggers\LoggerLE;
```

Koodinäide 3. Näide "use" lausest mujal rakenduses

Samuti nagu Logentriese puhul, on ka Loggly puhul võimalik integratsioon teha mitut moodi. Ka siinse teenuse puhul on autor kasutuse võtnud composeri. Loggly puhul tuleb lisada composer.json faili sõltuvus ning seejärel käivitada “composer update” (Koodinäide 4). Seejärel tuleb analoogselt Logentriese kasutusele võtuga, kirjutada wrapper klass, genereerida uus tähis Loggly keskkonnas ja wrapper klass mujal projektis kasutusele võtta.

```
"require": {
  "monolog/monolog": "^1.19"
},
```

Koodinäide 4. Näide composer require Loggly monolog komponendist

Kui wrapper klassid on kasutusele võetud, saab neile konstruktoris omistada muutuja ning luua neist uus objekt (Koodinäide 5). Seejärel on võimalik, muutujatesse salvestatud objekte kasutades, hakata välja saatma logisid teenustesse. Selleks, et teenustes logide analüüs lihtsam ja arusaadavam oleks, saadab autor logid välja JSON kujul. See ei pea nii ilmtingimata olema, kuid autori jaoks on see siiani kõige töökindlam lahendus olnud.

```php
public function __construct(Container $container) {

    /** @var DatabaseConnection databaseConnection */
    $this->databaseConnection = $container->get(DatabaseConnection::CLASS_NAME, Container::SHARED_INSTANCE);
    /** @var LoggerLE logger */
    $this->logger = $container->get(LoggerLE::CLASS_NAME);
    /** @var LoggerLY loggly */
    $this->loggly = $container->get(LoggerLY::CLASS_NAME);

}
```

Koodinäide 5. Näide mõne teise klassi constructorist

Kui wrapper klassid on kasutusele võetud, saab hakata nende kaudu logisid saatma. Näitena toob autor välja registreerimisel saadetava logikirje. See sisaldab märksõna, et tegemist on registreerimisega, riiki, mis valiti registreerimisel ning veebilehitseja infot (Koodinäide 6). Veebilehitseja info küsitakse välja eelnevalt teise meetodi poolt. Kuna kirjed saadetakse asünkroonselt, siis PHP ei jää ootama vastust, vaid läheb oma protsessidega edasi.

```php
$logData = [];
$logData['action'] = 'signup';
$logData['country'] = $country;
$logData['browser'] = $browser;
$toLog = json_encode($logData);
$this->logger->logger->addInfo($toLog);
$this->loggly->loggly->addInfo($toLog);
```

Koodinäide 6. JSON objekti koostamine ja saatmine

2.3. Rakenduse struktuur

Rakendus on kirjutatud järgides MVC arhitektuuri. Peamiselt koosneb rakendus neljast suurest kataloogist (vt Lisa 1).
Rakenduse peamised kataloogid:

- "*config*" - sisaldab endast seadistuse faile
- "*public*" - on rakenduse sissepääsu koht
- "*vendor*" - kus on ära kirjeldatud kõik sõltuvused, mida rakenduses kasutatakse
- "*library*" - sisaldab kogu rakenduse tarkust

Kataloog "*library*" on jaotatud loogilisteks osadeks, kus samalaadsete ülesannetega klassid on kokku kogutud ühte alamkataloogi. Struktuuris on näha, kuidas on eraldatud

kontrollerid, vaated ja muud osad rakendusest. Struktuuris on ka näha alamkataloogi "*Loggers*", kuhu alla on koondatud Logentriese ja Loggly komponentidele kirjutatud wrapper klassid.

Rakenduses kasutatud arhitetkuur võimaldab rakendust hoida võimalikult liigendatuna. See on kaasaegse rakenduste kirjutamise peamine mõte. Kõik tegevused peavad olema omaette meetodid ning klass peab olema loodud selliselt, et see vastutab ainult ühe kindla osa eest. Seda toetab ka MVC arhitektuur, kus on võimalik rakenduse osad teineteisest eraldatuna hoida.

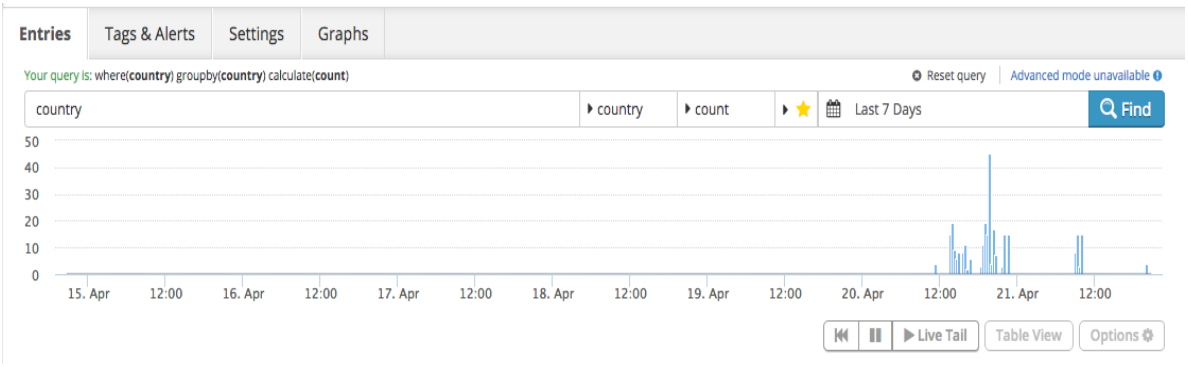
3. Logide analüüsimine teenustes

Käesoleva töö jaoks loodud rakenduse laadis autor üles oma kodulehele, ning lasi inimestel seda kasutada. Rakendus on samaaegselt loginud mõlemasse teenusesse paralleelselt andmeid. Seda selleks, et saaks teha analüüsi samade baasandmete pealt ning mõlemas teenuses oleksid logikirjed ühel kujul. Autor proovib mõlemist teenusest välja küsida samasuguseid päringuid, graafikuid ja kokkuvõtteid.
Välja küsitavad päringud:

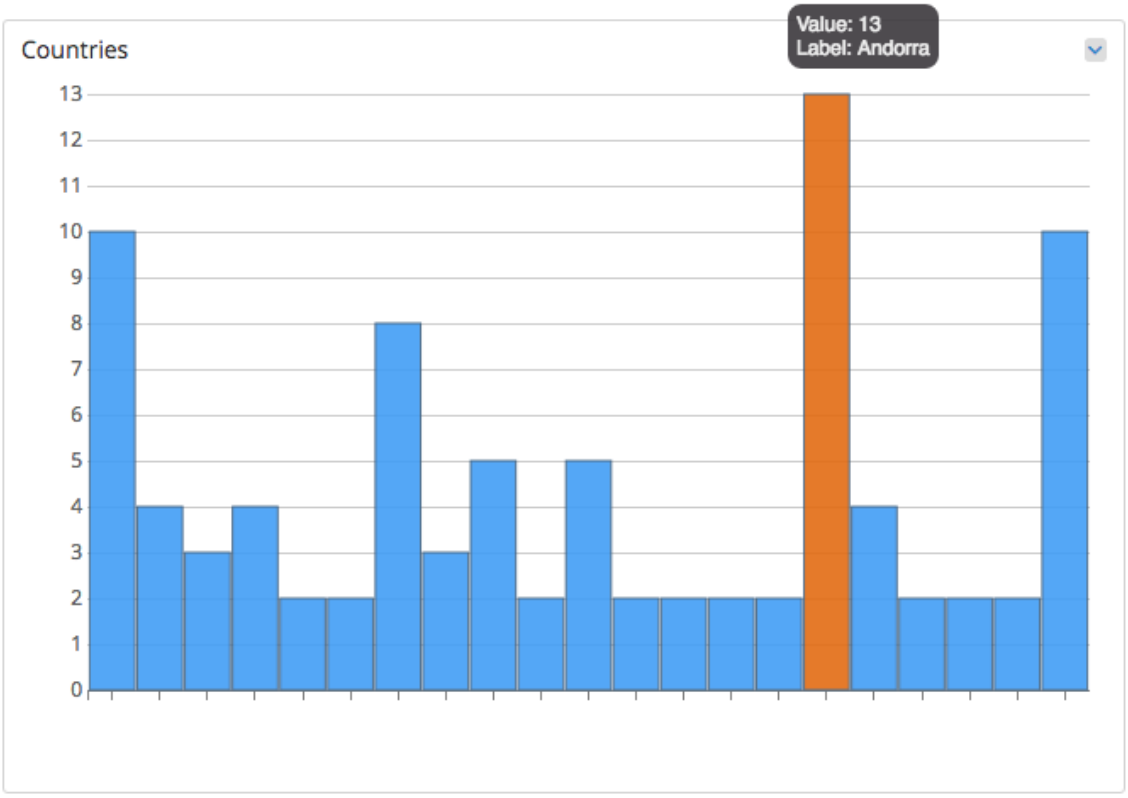
- Riikide esinemise sagedus registreerimisel
- Mis veebilehitsejat on kasutatud sisselogimisel
- Mitu korda on kasutajad keskmiselt päevas sisse loginud
- Mitu ülesannet on kokku lisatud läbi rakenduse
- Mitu ülesannet on kokku kustutatud läbi rakenduse

3.1. Logentries

Logentriese teenusesse on käesoleva töö jaoks loodud eraldi PHP kogumik ning teenusesse saadetud kirjed on JSON kujul, kuna teenus suudab lihtsalt selles formaadis erinevaid päringuid teostada. Selleks, et Logentriese teenuses andmeid välja küsida, on võimalik kasutada nende poolt välja töötatud päringu keelt nimega LEQL, mis lahtikirjutatuna tähendab "*Logentries Query Language*". (Logentries, 2016). Päringute esitamiseks tuleb kasutada teenuse päringuakent (Pilt 2) ning seejärel kuvatakse tulemusi. Selleks, et tulemustest saaks luua diagramme, tuleb päringuaknas päring salvestada, vajutades tähe ikoonile. Kui päring on salvestatud saab minna teenuses armatuurlauale ning hakata looma diagramme, kasutades eelnevalt salvestatud päringuid. Kõigepealt küsitakse välja erinevate riikide esinemise sagedus registreerimisel ning mitmest erinevast riigist üldse on kasutajad registreerunud (Pilt 3). Pärimiseks kasutatakse LEQL lauset "*where(country) groupby(country) calculate(count)*". Tekkinud tulpdiagrammis on näha, millistest riikidest on kõige rohkem registreeritud. Kuna erinevaid variante on palju, ei kuvata diagrammi alla tulpade nimesid, vaid need ilmuvad hiirega peale liikudes.

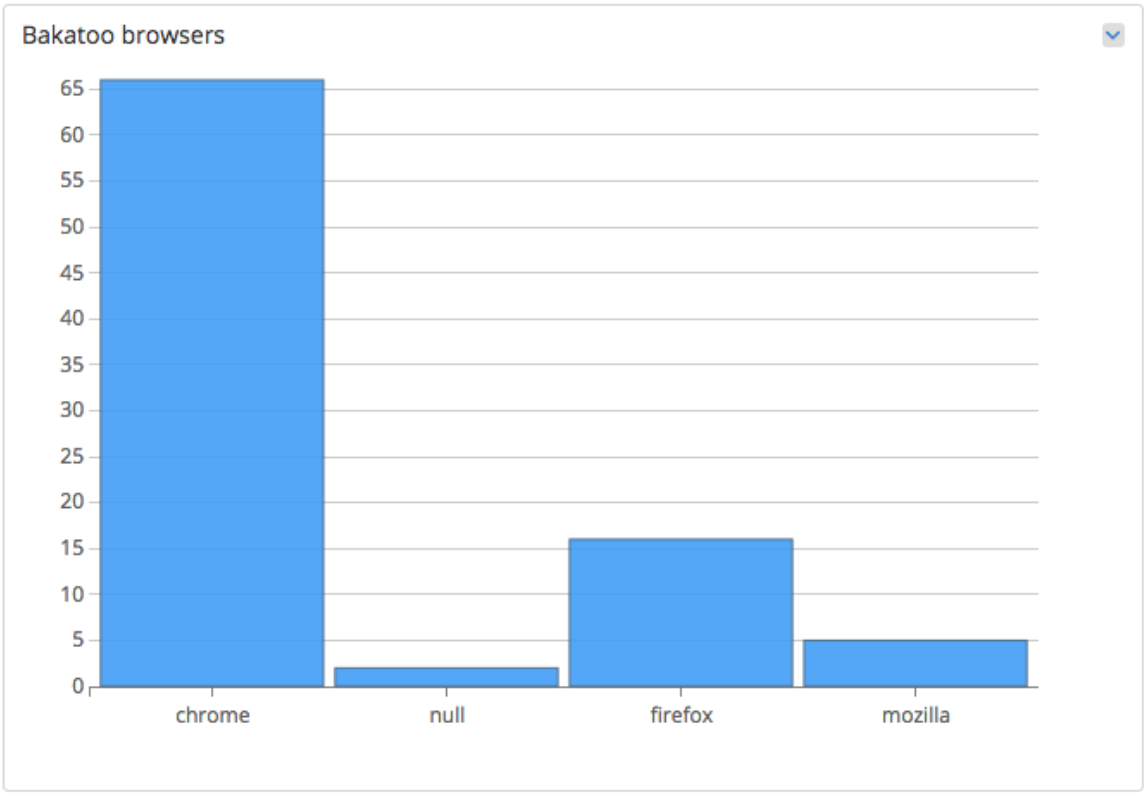


Pilt 2. Logentriese teenuse päringuaken



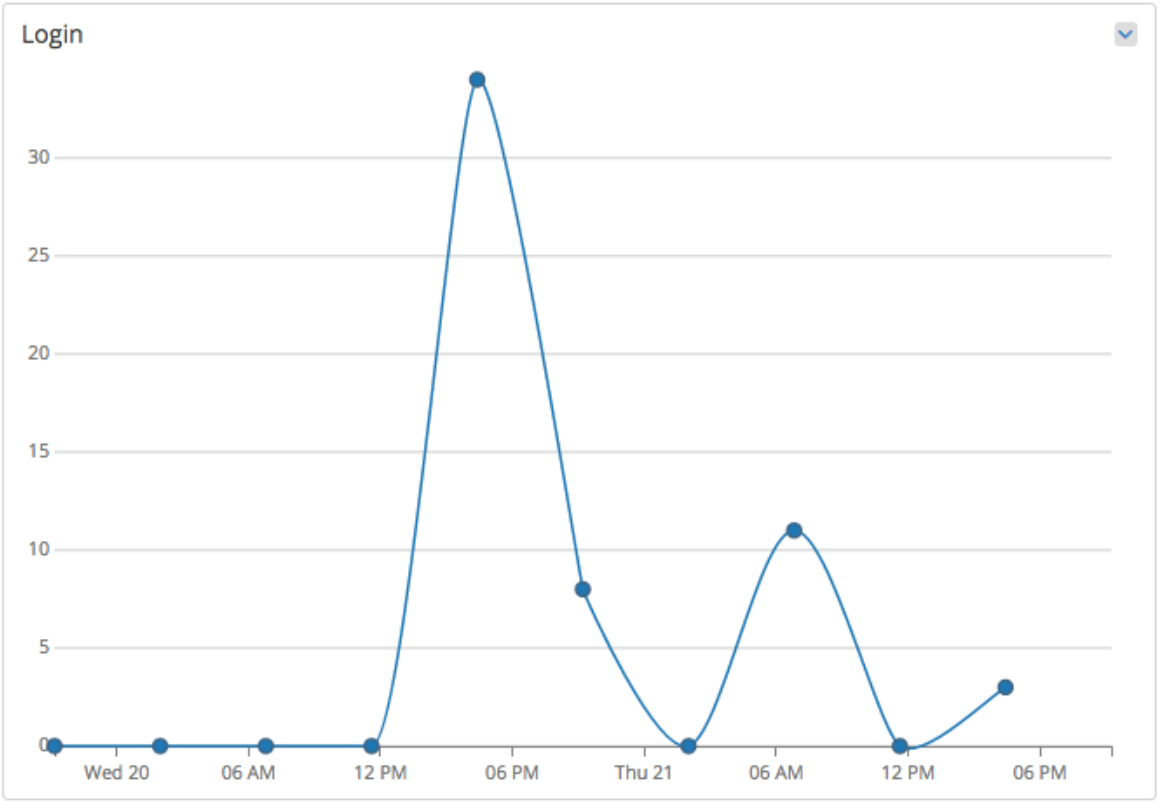
Pilt 3. Riikide esinemine registreerimisel

Kasutades samuti päringuakent ning salvestades päringu, saab hiljem välja küsida koondatud diagrammi selle kohta, milliseid veebilehitsejaid on kasutatud rakenduses registreerimiseks ja sisselogimiseks (Pilt 4). Pärimiseks kasutati LEQL lauset "*where(browser) groupby(browser) calculate(count)*" Diagrammilt on näha, et tulemuste seas on ka null väärtuseid. Seda nimelt seetõttu, et rakendust ülesse seades testis autor, kas veebilehitseja info jõuab teenusesse ning esimesel paaril korral see ei õnnestunud.



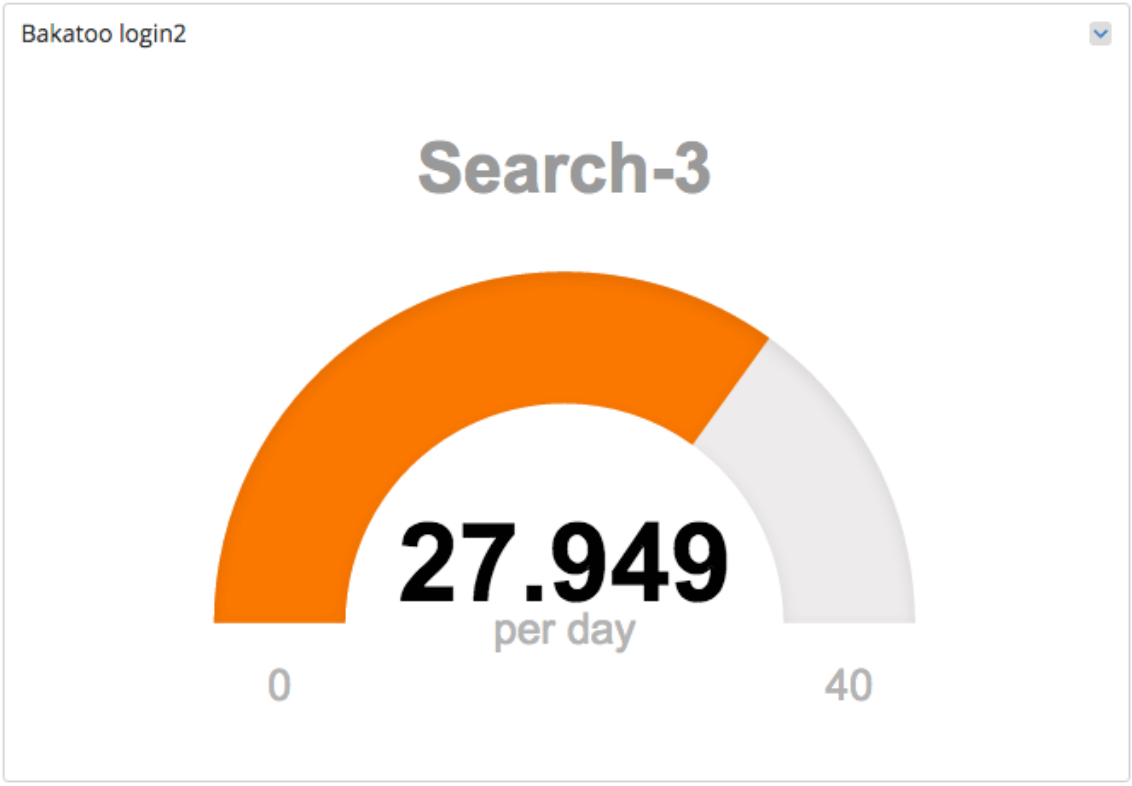
Pilt 4. Välja päritud veebilehitsejate info

Sama protseduuri kasutades, nagu eelmiste päringute puhul, on võimalik ka välja küsida külastamise statistika. Järgnevalt küsitakse välja, kui tihti on viimase kahe päeva jooksul raskendusse sisse on logitud (Pilt 5).



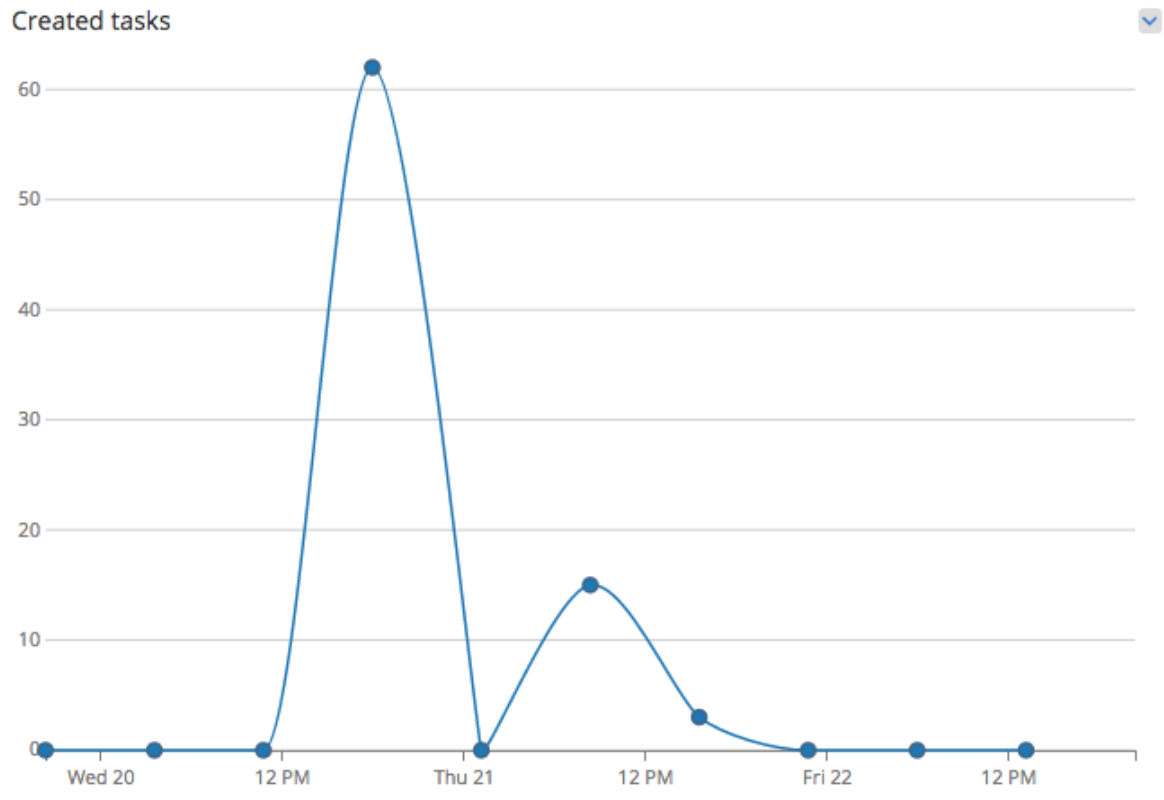
Pilt 5. Sisselogimise sagedus

Samuti on võimalik küsida päeva keskmist sisselogimiste arvu. Selle diagrammi puhul saab ka ära määrata, mis on ootus ning seejärel tekitatakse skaala, mis näitab, mitu protsenti ootusest on täidetud. Autor on loodud rakenduse puhul pannud ootuseks nelikümmend sisselogimist päevas. Skaala näitab, et eesmärgist ei jäänud palju puudu (Pilt 6). Skaala välja küsimiseks kasutati LEQL lauset "*where(login) calculate(count)*". Skaalal kirjeldatud "Search-3" näitab, et tegemist on kolmanda salvestatud päringuga. Autor proovis seda teenuses asendada päris päringuga, kuid keskkond seda ei võimaldanud.

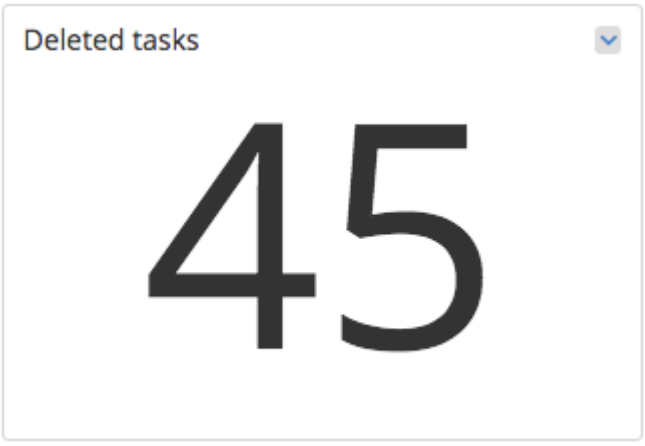


Pilt 6. Sisselogimiste arv skaalal

Viimase näitena küsitakse välja lisatud ja kustutatud ülesannete arv. Selleks, et välja küsida lisatud ülesannete arv, tuleb teostada LEQL käsk "*where(create) calculate(count)*" ning kustutatud ülesannete välja küsimiseks, tuleb teostada käsk "*where(delete) calculate(count)*". Lisatud käskude puhul tuuakse välja ajajoon, kus on näha, et kõige rohkem käske on lisatud 20. kuupäeva õhtul ning mõned ülesanded on lisatud ka 21. kuupäeva jooksul (Pilt 7). Kustutatud käskude puhul kasutati lihtsalt loendamist, ilma diagrammita. Tulemuseks kuvatakse arv, mitmest logikirjest soovitud märksõna "*delete*" leiti (Pilt 8).



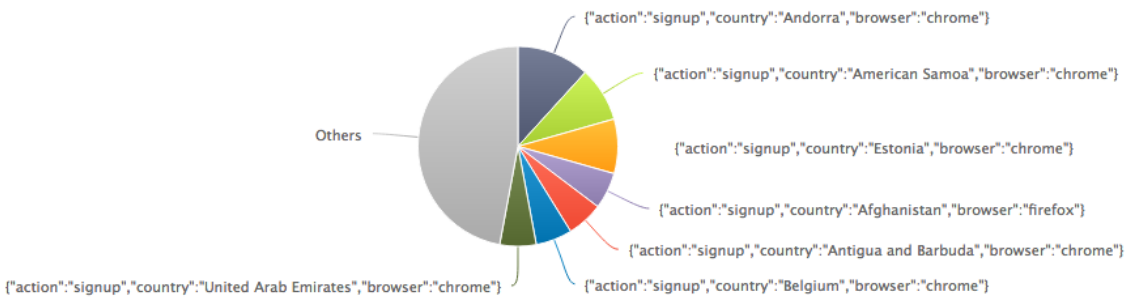
Pilt 7. Lisatud ülesannete graafik



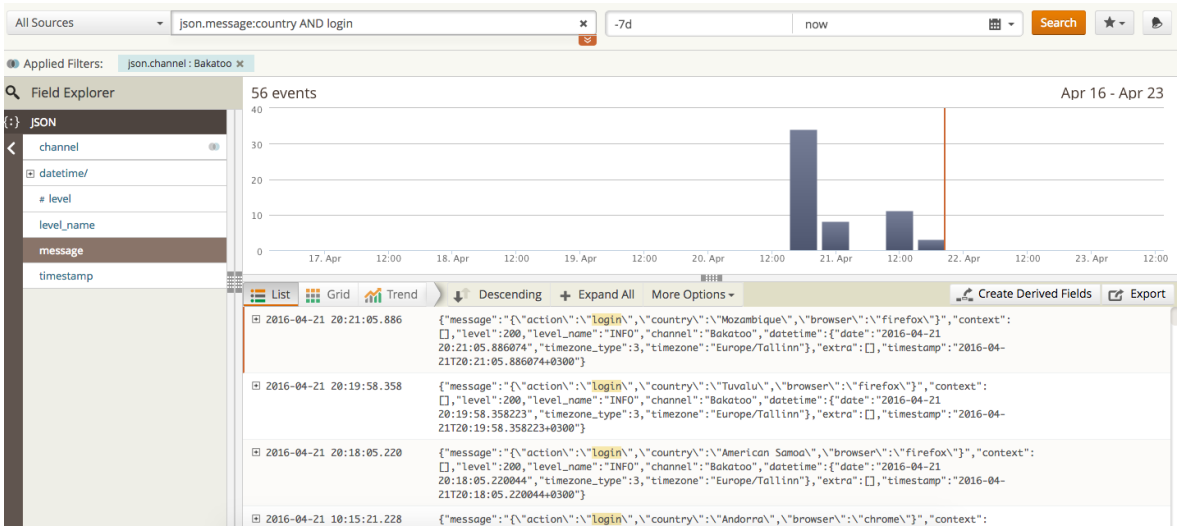
Pilt 8. Kustutatud ülesannete kogusumma

3.2. Loggly

Loggly keskkonda on käesoleva töö jaoks loodud eraldi kanal, kuhu saadetakse rakendusest logikirjeid. Kirjed on samuti JSON kujul ja identsed Logentriese teenusesse saadetud kirjetega. Päringute esitamiseks tuleb kasutada teenuses päringuakent (Pilt 9) ning seejärel näidatakse kasutajale soovitud tulemusi. Samuti nagu eelneva teenuse puhul, võimaldab ka Loggly sisestatud otsinguid salvestada, et hiljem neid uuesti sisestama ei peaks. Mõlemis teenuses on selle nupu ikooniks täht. Siinmaal kahe teenuse sarnasused lõpevad ning edaspidi on mindud kahte täiesti erinevat rada. Loggly on välja töötanud omaltpoolt uue päringukeele, mis ei ole autori arvates väga lihtsasti mõistetav ning vajab palju rohkem aega harjumiseks. Seda esmalt seetõttu, et see ei sarnane ühelegi SQL tüüpi keelele ning dokumentatsioonis olevad näited ei ilmesta väga hästi, kuidas seda keelt peaks kasutama. Riikide esinemise sageduse välja küsimiseks ei ole Loggly keskkonnas niivõrd mugavaid lahendusi, kui seda on Logentriesel. Autor ei leidnud tasuta versiooni kasutades lahendusi, kuidas luua samasugune koondülevaade kõikidest esinenud riikidest ja nende sagedusest. Pärast mõningaid katsetusi saadi tulemuseks sektordiagramm, mis ei toob välja kõige rohkem esinenud riigid (Pilt 10). Vähemesinenud riikide nimed on koondatud "*Others*" sektori alla. Päringu teostamiseks kasutatud käsklus oli "*json.message:country AND signup*".

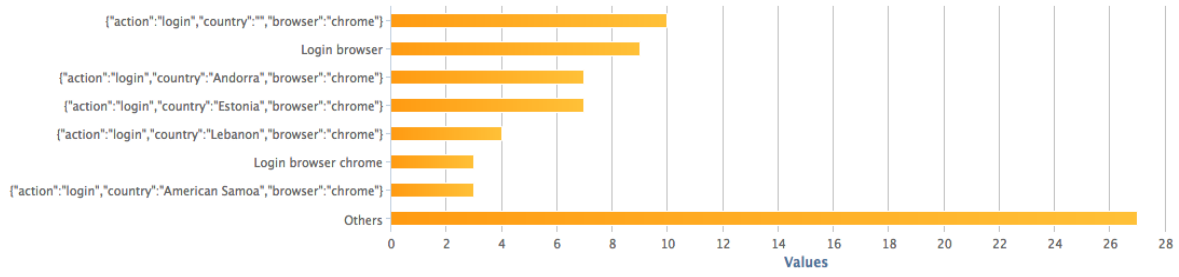


Pilt 10. Registreerimisel esinenud riigite diagramm



Pilt 9. Loggly teenuse päringuaken

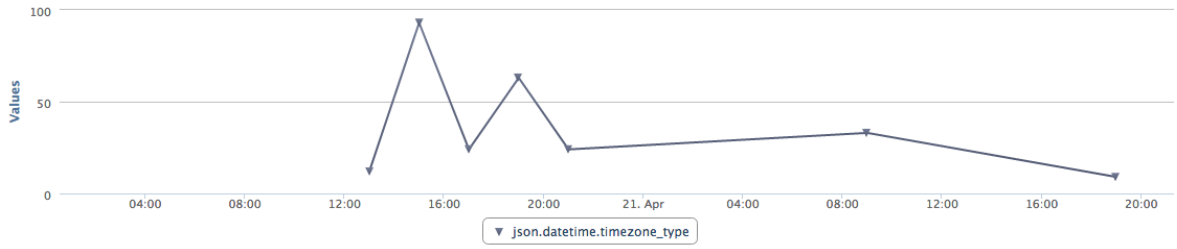
Kasutades sama protseduuri ja päringuakent saab välja küsida ka veebilehitsejate info kasutajate sisselogimisel. Koondatud info on esitatud tulpdiagrammina, milles on samuti näidatud vaid rohkem esinenud väärtused. Kõik vähem esinenud väärtused on koondatud tulpa "*Others*" ning toodud välja, kui üks väärtus. Veebilehitsejate info välja küsimiseks kasutati päringut "*json.message:browser AND login*". Loodud diagrammis (Pilt 10) on teisel kohal kirje "*Login browser*", mida autor tegelikult ei tahtnud, kuna päringus täpsustati, et päring peaks toimuma ainult JSONi attribuudile "*message*" . Loodud diagrammist on näha, et Loggly ei suuda JSON objekti seest teha koondamisi, kuna mitu korda on näidatud veebilehitseja Chrome esinemist.



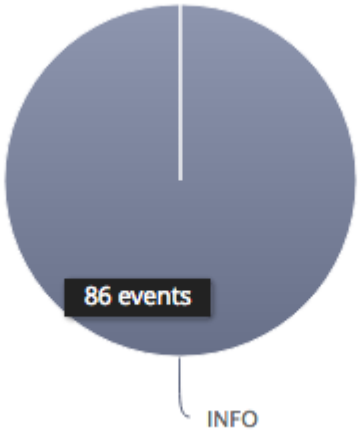
Pilt 10. Sisselogimisel kasutatud veebilehitsejad

Loggly tasuta keskkonnas ei pakuta võimalust välja küsida sisselogimiste arvu, kui koondsummat selliselt, nagu seda võimaldas Logentries. Võimalik on välja küsida kindlal perioodil toimunud sisselogimiste ajatelg (Pilt 11) või sektordiagramm, mis koosneb ühest sektorist ja selle väärtuseks on koondsumma sisselogimiste arvust (Pilt 12). Selleks, et see

välja pärida, kasutati teenuses lauset "*json.message:login*", mis päris välja kõik kirjed, mis sisaldasid märksõna "*login*".



Pilt 11. Sisselogimiste ajatelg



Pilt 12. Sisselogimiste koguarv

Ka ülesannete lisamise ja kustutamise koguarvu välja kuvamine ei ole võimalik nii arusaadavalt nagu seda ole Logentriese teenuses. Sellel samal põhjusel, et Loggly ei võimalda välja küsida kogusummat. Selleks, et tulemus ikkagi kuidagi kuvada, kasutas autor teenuses sektordiagramme, mis koosnesid ühest sektorist. Seeläbi oli võimalik ikkagi kogusumma saada. Ülesannete lisamise arvu väljaküsimiseks kasutati lauset "*json.message:task AND create*" ja kustutatud ülesannete arvu väljaküsimiseks kasutati sarnast lauset "*json.message:task AND delete*". Loodud ülesannete diagrammist (Pilt 13) on näha, et kokku loodi 80 ülesannet ning kustutatud ülesannete diagramm (Pilt 14) näitab, et kustutatud on neist 45.

Pilt 13. Lisatud ülesannete koguarv

Pilt 14. Kustutatud ülesannete koguarv

4. Analüüs

Käesolevas töös näitas autor esmalt, kuidas loodud rakenduse külge siduda vajalikud komponendid, et rakendus saaks hakata suhtlema logimise teenustega ning teiseks, kuidas kasutada teenuste keskkonda hilisemal logide analüüsil. Teenuste integreerimine rakenduses oli autori arvates lihtne ning sirgjooneline, kuna kasutusele võeti composer, mis poole tööst ise ära tegi. Lisaks loodud wrapper klassid olid samuti lihtsad ning läbi nende logide välja saatmine oli samuti lihtsalt tehtud.

Teenuste keskkondade kasutamine logide analüüsiks oli ühe teenuse puhul lihtne ja arusaadav, kuid teise teenusepakkuja puhul üsna keeruline. Rakendusest saadeti välja ühel ja samal JSON kujul logikirjed ning seda just eesmärgiga, et baasandmed oleksid samad mõlemi teenuse puhul. Mõlemad teenusepakkujad soovitavad kasutajatel kasutada just JSON kujul logi kirjeid. Logentriese puhul oli vastavate koondandmete ja diagrammide välja küsimine väga lihtne ning intuitiivne kuna LEQL sarnaneb paljude teiste SQL tüüpi keeltega. Samuti oli teenuse keskkond üsna lihtsasti õpitav ning autorile see lahendus meeldis. Tööd sai keskkonnas teha kiirelt ning ei olnud üleliigset infomüra, mis segaks.

Loggly puhul nägi autob palju vaeva, et aru saada, kuidas keskkond töötab. Nimelt on nende dokumentatsioon otsingute osas väga nõrk ning ostinguteks kasutatav keele süntaks ei ole iseenesest mõistetav. Kui keskkonnas rohkem tööd teha ja saada kogemusi, siis kindlasti mingi hetk muutub ka nende poolt kasutatav süntaks mõistetavaks. Samuti nägi autor palju vaeva, et teostada otsinguid saadetud JSON objekti seest. Nimelt ei ole selle kohta üldse näiteid dokumentatsioonis ning selleks tuli hakata uurima foorumeid, kus inimesed on sama probleemi otsa sattunud. Väga raske oli saada tulemuseks samasuguseid diagramme, nagu Logentriese puhul. Lisaks on Loggly jätnud tasuta kasutajatele väga vähe võimalusi, võrreldes konkurendiga. Loodud diagrammid ei olnud nii informatiivsed, kui Logentriese poolt võimaldatud lahendused.

Peale mõlema teenusepakkuja kasutamist arvab autor, et Logentries on lihtsamini kasutatav ning pakub lihtkasutajale rohkem võimalusi. Samuti on nende keskkond tavakasutajale kiiremini vastuvõetav ning päringukeel arusaadavam. Samal ajal on Loggly väga tehniline ning infot, mida kasutajatele pakutakse on palju. Esialgu on seda kõike liiga palju ning esimesel kasutamisel on raske seda kõike hoomata. Autor ei välista, et pikemal

kasutamisel tuleb see infoküllus kasuks ning suurte ettevõtete puhul, kes seda kasutavad, on infot piisavalt ning logidega tegelevad spetsialistid saavad keskkonnas kõik vajaliku tehtud. Võrreldes neid kahte on autor arvamusel, et Logentries on suunatud pigem lihtsale kasutajale ja suurele massile ning Loggly on võtnud suunaks olla väga tehniline tööriist spetsialistidele.

Kokkuvõte

Käesolevas töös tutvustatud teenused ja lahendused on autori arvates vaid põgus sissevaade kolmanda osapoole teenuste kasutamisse rakenduste logimisel ja hilisemasse logide analüüsimisse. Kindlasti saab sügavamalt süveneda teistesse teenustesse ja nende võimalustesse. Samuti ka tutvustatud teenuste teistesse lahendustesse. Näiteks ei kasutanud autor käesolevas töös Logentriese Datahub lahendust, kuna puudus taristu selle üles seadmiseks ning sellest võiks autori arvates kirjutada lausa omaette töö. Siinse bakalaureusetöö eesmärk oli tutvustada kahte kolmanda osapoole teenust rakenduse logimiseks ning näidata teenuste poolt pakutavaid logide analüüsimise võimalusi. Eesmärgi saavutamiseks loodi eraldi rakendus ning pandi see autori veebilehele üles. Rakenduses võeti kasutusele vajalikud komponendid, et see saaks hakata suhtlema logimise teenustega ning hiljem tehti teenuste keskkonnas päringuid andmete koondamiseks ja analüüsimiseks.

Teenuste põhimõte ja vajadus on hoida rakendused ja nende logid eraldi. Seda nii turvakaalutlustel, kui ka hilisema logide analüüsi hõlbustamiseks. Logide kirjutamine ja nende haldus on hetkel Tallinna Ülikooli programmeerimise õppeainetes nõrgalt esindatud ning autor arvab, et käesolev töö võib olla õppematerjal järgnevatele kursustele.

Autori hinnangul sai käesoleva töö eesmärk saavutatud, kuna anti ülevaade kahest teenusepakkujast ja nende poolt pakuavatest võimalustest, loodi rakendus logide tekitamiseks ning näidati mõlemi teenusepakkuja võimalusi logide analüüsimiseks. Autor ise sai juurde teadmisi ja kogemusi logide olemusest ja nende saatmisest teenusepakkujate keskkondadesse ning rakenduses vaja läinud komponentidest. Autor arvab, et antud teemal on võimalik uurida põhjalikumalt Logentriese Datahub lahendust ning mõlemi teenusepakkuja teisi lahendusi, mida tasuta kontoga ei ole võimalik proovida. Samuti saab edasi uurida ja läbi proovida, kuidas töötab häirete reaalajas teavitamine mõlemi teenusepakkuja puhul.

Summary

Third-Party Services for Application Log Management

For this bachelors thesis an application was created to generate data and, from where logs were sent to two selected services, Logentries and Loggly. The author showed the necessary steps to integrate these two services into an application. Also a brief introduction was made into the use of composer in PHP applications. This application was uploaded to the authors website and used by random people.

The main reason for separating logs from applications is that it is a security risk when they are in the same server. When something happens to the server, all will be lost and there are no logs that show what happened. This is where third-party services come in and provide an alternative to that. Those services also provide a simpler way to analyse and collect large amounts of logged data.

While analysing logs in the selected services environments, the author found that Logentries was much more user-friendly and mainly orianted towards all programmers. On the other hand, the author found that Loggly was very specific and quite hard to use at first. Loggly provides a lot of information which may be overwhelming at the beginning. In the authors opinion Loggly is mainly orianted towards specialists.

The main goal of the bachelors thesis was to give an oversight to third-party services for application log management. In the authors opinion, the goal was met. There is a lot more to learn and explore and this bachelors thesis was a small insight into the use of third-party services for application log management.

Kasutatud kirjandus

Logentries (2016) Pricing. Loetud 27.02.2016 aadressil:

https://logentries.com/pricing/

Logentries (2016) Token-based TCP. Loetud 27.02.2016 aadressil:

https://logentries.com/doc/input-token/

Logentries (2016) Syslog Forwarding. Loetud 27.02.2016 aadressil:

https://logentries.com/doc/syslog/

Logentries (2016) Tags and Alerts. Loetud 27.02.2016 aadressil:

https://logentries.com/doc/setup-tags-alerts/

Logentries (2016) Datahub. Loetud 27.02.2016 aadressil:

https://logentries.com/product/datahub/

Loggly (2016) Pricing. Loetud 27.02.2016 aadressil:

https://www.loggly.com/plans-and-pricing/

Loggly (2016) Product. Loetud 27.02.2016 aadressil:

https://www.loggly.com/product/

Loggly (2016) Dynamic Field Explorer. Loetud 27.02.2016 aadressil:

https://www.loggly.com/docs/dynamic-field-explorer/

Techcrunch (2016) Logentries. Loetud 27.02.2016 aadressil:

http://techcrunch.com/2012/07/17/logentries-raises-1m-to-join-the-big-data-log-management-crowd/

Vallaste (2016) Loetud 24.04.2016 aadressil:

http://www.vallaste.ee/index.asp

Composer (2016) Loetud 22.04.2016 aadressil:

https://getcomposer.org/

Github Logentries handler (2016) Loetud 14.04.2016 aadressil:

https://github.com/logentries/logentries-monolog-handler

Github Monolog (2016) Loetud 14.04.2016 aadressil:

https://github.com/Seldaek/monolog

LISAD

Lisa 1. Rakenduse struktuur