Janne Saraste

Sosiaalisen median mobiilipalvelun suunnittelu ja toteutus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

14.4.2014

Metropolia

Tekijä Otsikko Sivumäärä Aika	Janne Saraste Sosiaalisen median mobiilipalvelun suunnittelu ja toteutus 35 sivua 14.4.2014
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja	lehtori Peter Hjort

Insinöörityön tavoitteena oli tutustua sosiaalisen median mobiilipalvelun rakennusprosessiin sekä tutkia, mikäli olemassa olevissa sosiaalisen median palveluissa esiintyviä ongelmia voitaisiin ratkaista uusien lähestymistapojen avulla. Työssä keskityttiin erityisesti mobiilipalvelun tekniseen toteutukseen Android-sovelluksen sekä palvelimella ajettavan taustajärjestelmän osalta. Lisäksi tutkittiin ratkaisumahdollisuuksia kahden sosiaalisen median palveluja vaivaavan ongelman ratkaisemiseksi. Projektin käsittelemät ongelmat olivat uusien kontaktien tapaaminen sekä käyttäjien kontaktilistojen paisuminen. Projektin aihe ja tavoitteet asetettiin itsenäisesti ilman ulkopuolista yhteistyökumppania.

Työn aikana rakennetussa palvelussa hyödynnettiin useita eri moderneja mobiili- ja web-kehityksessä käytettyjä työkaluja sekä avoimen lähdekoodin komponentteja. Tämän lisäksi projektissa tutustuttiin mobiilipalvelujen rakennuksessa käytettyihin tekniikoihin. Projektin aikana kiinnitettiin myös jatkuvaa huomiota palvelun jatkokehittämisen mahdollistamiseen.

Työn lopputuloksena saatiin aikaan useista komponenteista koostuva Android-alustaa hyödyntävä mobiilipalvelu, joka koostuu Android-sovelluksesta, sekä palvelimella ajettavasta, Rails-ohjelmistokehystä hyödyntävästä taustajärjestelmästä. Lopputuloksena havaittiin, että käyttäjien kontaktilistoihin liittyvien ongelmien ratkaiseminen vaatii palvelulta huomattavan määrän käyttäjiä, jonka vuoksi kyseiseen ongelmaan kehitetyn ratkaisun toimivuutta ei voitu testata.

Työn avulla saatiin selville suunnitellun palveluidean toimivuus sekä opittiin huomattava määrä asioita mobiilipalvelujen rakentamisesta.

Avainsanat	Android, Rails, Redis, MariaDB

Author	Janne Saraste
Title	Design and implementation of a mobile social network
Number of Pages	35 pages
Date	14 April 2014
Degree	Bachelor of Engineering
Degree Program	Information Technology
Specialisation option	Software Engineering
Instructor	Peter Hjort, Senior Lecturer

The objective of this thesis project was to construct a mobile social network and determine if location based methods for forming relationships between users could work as an alternative to some of the more commonly used methods on popular social networking services. As a result, the thesis describes new type of systems for handling contact list overpopulation and relationship formation.

The mobile social network built during the project utilizes various different modern mobile and web development tools such as the Ruby On Rails framework and the Android bootstrap framework for creating a backend service as well as an Android application which provides a way to access the social network.

The final product of this project was a mobile social network, constructed of multiple different components such as an Android application and a backend service. The created mobile social network made it clear that testing some of the originally planned methods for common social networking functionality was not feasible without a considerably larger user base. Regardless of all the problems, the project was a highly educational experience and the created product has high potential for further development.

Keywords	Android, Rails, Redis, MariaDB

Sisällys

Lyhenteet

COC	*Convention over Configuration*. Rails-ohjelmistokehyksen sääntöihin perustuva automatisointi, jolla helpotetaan ohjelmistokehittäjän työtä.
DRY	*Don't Repeat Yourself*. Ohjelmistokoodin uudelleen käyttämiseen kannustava periaate.
GPS	*Global Positioning System*. Satelliitteihin pohjautuva, maailmanlaajuinen paikannusjärjestelmä.
HTTP	*Hypertext Transfer Protocol*. Internetin yleisimmiten käytetty protokolla, jota selaimet, sekä palvelimet käyttävät kommunikointiin.
JSON	*JavaScript Object Notation*. Yksinkertainen tiedonsiirtomuoto.
MVC	*Model-View-Controller*. Ohjelmistoarkkitehtuurityyli, jossa käyttöliittymä pyritään erottamaan sovellusaluetiedosta.
NFC	*Near Field Communication*. Lyhyille matkoille tarkoitettu tiedonsiirtomenetelmä.
REST	*Representational State Transfer*. HTTP-protokollan yhteydessä kehitetty arkkitehtuurimalli.
SQL	*Structured Query Language*. Relaatiotietokantojen kyselykieli.
TCP/IP	*Transmission Control Protocol / Internet Protocol*. Useiden Internet-liikennöinnissä käytettävien tietoverkkoprotokollien yhdistelmä.
URI	*Uniform Resource Identifier*. Merkkijono, jolla osoitetaan resurssin sijainti.
YAML	*YAML Ain't Markup Language*. Ihmisystävällinen merkintäkieli.

1 Johdanto

Insinöörityön aiheena on sosiaalisen median mobiilipalvelun toteuttaminen kaikilta osa-alueiltaan. Tämä tarkoittaa palvelun mobiilisovelluksen, web-rajapinnan sekä palvelimella ajettavan taustajärjestelmän suunnittelua ja rakentamista. Projektissa rakennettava sosiaalisen median palvelu eroaa muista saatavilla olevista palveluista sen ainutlaatuisen idean osalta, joka korvaa perinteisen ystävien lisäämisessä käytettävän mekaniikan uudella, sijaintiin perustuvalla mekaniikalla. Käytännössä tämä tarkoittaa sitä, että käyttäjät saadaan oikeasti tapaamaan ihmisiä sen sijaan, että he vain keskustelisivat internetin välityksellä. Palvelu tarjoaa myös uniikin tavan täysin uusien ihmisten tapaamiseen. Työssä rakennettavan palvelun mobiilisovellus tulee hyödyntämään Android-käyttöjärjestelmää sekä useita kolmannen osapuolen kirjastoja. Vastaavasti palvelun taustajärjestelmä hyödyntää useita eri moderneja web-sovelluskehityksessä käytettyjä teknologioita.

Maailmalla suositut sosiaalisen median palvelut kuten Facebook ovat usein rakentuneet käyttäjille asetettujen olettamusten päälle, ne muun muassa olettavat, että palveluun rekisteröityvällä henkilöllä on jo entuudestaan ystäviä tai kontakteja, joiden kanssa palvelua käytetään. Kyseiset palvelut eivät siis pohjimmaiselta rakenteeltaan ole suunniteltu uusien suhteiden solmimiseen, vaan niiden pääasiallinen tarkoitus on tarjota kanava muualla solmittujen sosiaalisten suhteiden ylläpitämiseen. Tämä käyttäjille asetettu perusolettamus on sisäänrakennettuna lähes kaikkiin maailmalla suosiota saavuttaneisiin sosiaalisen median palveluihin. Kyseisen olettamuksen tekeminen ei kuitenkaan ole välttämätöntä, ja sen vaihtoehdoksi on jo olemassa eri menetelmiä. Näitä menetelmiä hyödyntävät palvelut eivät kuitenkaan ole saavuttaneet huomattavaa suosiota, vaan ovat väistämättä jääneet taka-alalle. Tämän olettamuksen korvaaminen toimivalla menetelmällä onkin mielenkiintoinen ongelma, jolla saattaisi olla jopa potentiaalista markkina-arvoa. Korvaavan menetelmän löytämiseksi tämä insinöörityö esittelee sosiaalisen median mobiilipalvelun rakennusprosessia, jonka tarkoituksena on tuottaa palvelu, jolla ongelma voitaisiin ratkaista.

Puhuttaessa sosiaalisen median palveluista on tärkeää ymmärtää, että käsite on itsenään hyvin laaja ja kattaa käytännössä kaikki web- ja mobiilisovellukset, jotka sisältävät jonkin sosiaaliseen vuorovaikutukseen perustuvan toiminnallisuuden. Useimmat näistä palveluista voidaan kuitenkin erottaa kolmen päätoiminnallisuuden

perusteella. Ensinnäkin kyseiset palvelut mahdollistavat lähes poikkeuksetta käyttäjäprofiilin luonnin, toiseksi ne mahdollistavat kuvien ja tekstin lähettämisen palveluun, esimerkiksi kommentin muodossa. Viimeisenä, ne mahdollistavat palvelun käyttäjien välisten ystävyyssuhteiden luonnin, jolloin käyttäjien keskeinen kommunikointi helpottuu.

Vastaavaa rakennetta noudattavia, maailmalla suosittuja, sosiaalisen median palveluja yhdistävänä tekijänä on niiden luoma ympäristö, jossa ystävyyssuhteiden luontiprosessi on loppukäyttäjän näkökulmasta niin yksinkertainen, ettei hän kiinnitä siihen enää huomiota. Tämä johtaa huomattavan usein tilanteeseen, jossa käyttäjien kontaktilistat paisuvat niin suuriksi, etteivät he kykene enää edes muistamaan kaikkia ystäväksi hyväksymiään henkilöitä. Ennen nykyisiä web-pohjaisia verkostoja, ihmisten sosiaaliset kontaktit olivat käytännössä rajoittuneet kännykän kontaktilistaan, joka oli usein vielä hyvin rajoitetun kokoinen. Vastaavasti ennen kännyköitä nämä suhteet olivat vieläkin rajoittuneempia ja kattoivat käytännössä vain aivan läheiset ystävät. Vastaava sosiaalisten suhdemäärien paisuminen herättääkin kysymyksen, olisiko kontaktimäärien rajoittamisella positiivista vaikutusta suhteiden laatuun. Vastauksen löytämiseksi, insinöörityön yhteydessä rakennettava mobiilipalvelu tarkastelee menetelmää, joka mahdollistaisi kyseisen asian tutkimisen.

Moderneja sosiaalisen median palveluja vaivaavia ongelmia ratkaistaessa kattavin lähestymistapa on suunnitella ja rakentaa kokonaan uusi palvelu, jossa aikaisemmissa palveluissa esiintyvät ongelmat on otettu huomioon, jo aivan suunnitteluvaiheesta lähtien. Vastaavien palvelujen laajuuden vuoksi palvelu, jota tässä insinöörityössä käsitellään, on kuitenkin keskittynyt aikaisemmin mainittujen ongelmien ympärille ja sen toiminnallisuus on rajoitettu vain työn kannalta oleelliseen kokoonpanoon. Työssä rakennetun palvelun pääasiallinen tarkoitus on toimia kokeena, jolla testataan, ovatko kyseiset ongelmat ratkaistavissa täysin mobiilipohjaisessa palvelussa, sekä käsitellä yleisiä haasteita, joita sosiaalisen median mobiilipalvelujen rakentamisessa tulee vastaan. Työn rajoitetusta tarkoituksesta huolimatta palvelun toteutuksessa tullaan huomioimaan mahdollinen toiminnallisuuden laajennettavuus sekä jatkokehitys, jotka olisivat huomattavan oleellisia seikkoja, mikäli palvelua toteutettaisiin yritysolosuhteissa.

2 Mobiilipohjainen sosiaalisen median palvelu

Sosiaalisen median palveluilla tarkoitetaan ihmisten väliseen kommunikointiin suunniteltuja palveluja, joissa käyttäjien välisiä viestejä välitetään lähes aina internet-yhteyden avulla. Sosiaalisen median palveluja on pääasiassa kahta eri tyyppiä, on olemassa web-pohjaisia sekä mobiilipohjaisia palveluja. Näiden tyyppien eroavuus näkyy palvelun käyttövaatimuksissa. Web-pohjaisia sosiaalisen median palveluja voi käyttää laitteesta riippumatta, mikäli saatavilla on internet-yhteys sekä palvelua tukeva verkkoselain. Vastaavasti mobiilipohjaiset sosiaalisen median palvelut vaativat käyttäjältä tietyntyyppisen laitteen, johon tulee asentaa palvelun käytön mahdollistava sovellus. Useissa tapauksissa mobiilipohjaisia sosiaalisen median palveluja ei ole mahdollista käyttää ilman vaadittavaa ohjelmistoa. Tämä ei kuitenkaan ole kaikkien palvelujen kohdalla vaadittua, vaan jotkin palvelut tarjoavat myös verkkosivuston, jonka kautta palvelua pääsee käyttämään.

2000-luvun alkuvuosina, jolloin sosiaalisen median palvelut tekivät läpimurtoaan ihmisten arkipäivään, ne olivat käytännössä poikkeuksetta web-pohjaisia, eivätkä niiden mobiilikäyttäjien määrät olleet erityisen merkityksellisiä. Vuonna 2007 julkaistun iPhonen myötä alkaneesta älypuhelinten yleistymisestä on kuitenkin aiheutunut radikaali muutos palvelujen käyttötavoissa, sillä tilastojen mukaan mobiilikäyttäjien aiheuttamat, rahalliset tuotot palveluissa kuten Twitter ja Facebook, ovat viimevuosina ylittäneet web-käyttäjät [Facebook Q4 2013 results 2014; Smith 2014]. Vastaava kehitys on nähtävissä myös globaalissa internet liikenteen määrässä, josta mobiililaitteet tuottivat 2014 vuoden alussa lähes 24 % [StatCounter Global Stats 2014]. 2000-luvun alun tilanne on siis kääntynyt täysin päälaelleen, mobiilisuosion räjähdys on tuonut mukanaan myös uusia palvelusuunnittelussa käytettyjä malleja, kuten mobiili-ensin. Kyseinen malli painottaa mobiililaitteiden huomioon ottamista palvelusuunnittelussa. Käytännössä tämä tarkoittaa esimerkiksi web-sivustojen kohdalla sitä, että sivusto osaa näyttää käyttäjälle sopivaa sisältöä riippuen siitä, millä laitteella käyttäjä sivustoa käyttää.

2.1 Mobiilipohjaisuuden haasteet

Nykyisestä kehityksestä voisi pikaisesti ajatellen päätellä, että suositun mobiilipohjaisen palvelun tekeminen olisi helppoa, sillä mobiilialustojen käyttäjämäärät

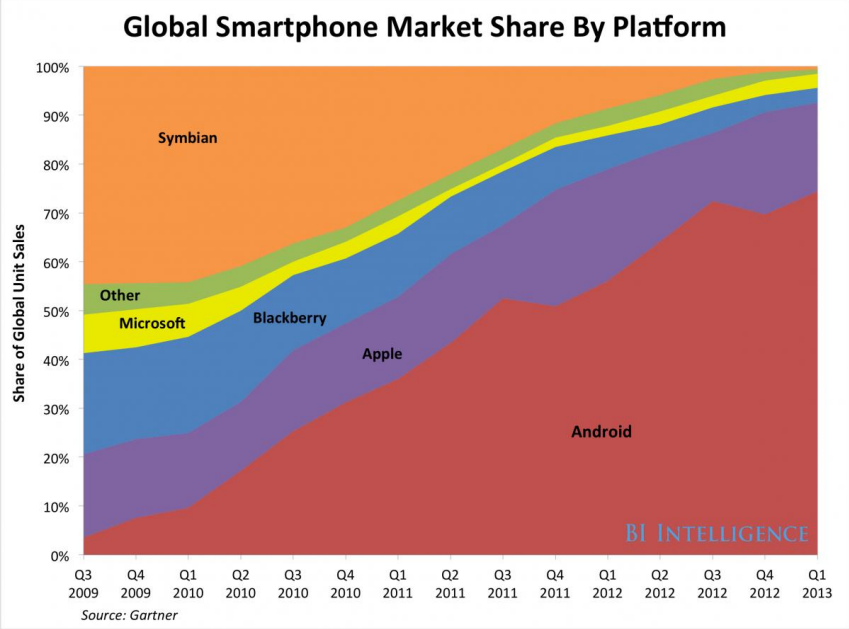
ovat jatkuvassa kasvussa. Suosituimmat mobiilialustat tarjoavat myös keskitetyt kauppapaikat kuten Google Play Store ja Apple App Store, joiden kautta oman sovelluksen saa jaettua maailmalle ilman erityistä vaivannäköä. Tilanne ei kuitenkaan ole aivan niin yksinkertainen, sillä ympäristöt, joissa mobillipohjaiset sovellukset elävät, ovat jatkuvan muutoksen ja kovan kilpailun alaisia. Keskitetyt kauppapaikat toimivat usein vain pullonkaulana sovellusvalikoimien esittämisessä käyttäjille, eivätkä uudet sovellukset saa niistä erityistä hyötyä. Vasta alustavan suosion saavuttamisen jälkeen, kauppapaikkojen etusivut alkavat toimia hyödynnettävänä mainoskanavana ja ruokkimaan jo ennestään suosittuja sovelluksia.

Web-pohjaisiin palveluihin verrattuna mobiilipalvelujen aiheuttamat haasteet ovat siis hyvin erilaisia. Web-palvelujen tapaan sovellusten täytyy pystyä tarjoamaan käyttäjälle jotain mielenkiintoista, mutta toisin kuin web-palvelujen käyttäjien, mobiilikäyttäjien asettamat vaatimukset sovelluksen ulkoasulle ja toiminnalle ovat huomattavasti ankarammat. Mobiilisovellusten käyttäjien kärsivällisyys on huomattavasti web-käyttäjiä heikompaa, ja he vaativat lähes välittömän pääsyn sovelluksen tarjoamaan, oleelliseen sisältöön. Pienetkin viiveet, kuten tunnuksen luominen tai ohjeiden lukeminen, ovat usein liikaa keskivertomobiilikäyttäjälle. Tämä johtaa usein sovelluksen käyttämisen lopettamiseen jo ennen kuin se on kunnolla alkanutkaan. Kynnys mobiilisovellusten poistamiseen on myös huomattavasti pienempi, kuin niiden asentamiseen, jonka vuoksi käyttäjien mielenkiinnon ylläpitäminen asennuksen jälkeen, vaatii huomattavan määrän työtä. [Baker 2013]

Mobiilisovelluksen käyttöliittymä onkin ehkä yksi tärkeimmistä asioista, joihin kehitysvaiheessa tulisi kiinnittää huomiota, mikäli sovelluksen käyttäjän halutaan palaavan ensimmäisen kokeilukerran jälkeen. Sovelluksen käyttöliittymän tulee olla suunniteltu niin, että käyttäjä osaa käyttää sitä oikein ilman erillistä ohjeistusta ja sen käyttökokemus sekä ulkoasu ovat alusta alkaen miellyttäviä. Käytännössä tämän tavoitteen saavuttaminen vaatii tarkkaan suunnittelua sille, millaisia värejä sovelluksessa käytetään ja mihin eri toimintoja kuvaavat napit, valikot ja tekstit sijoitetaan.

2.2 Sopivan mobiilialustan valinta

Palvelun suunnittelussa tulee väistämättä vastaan kysymys, mille tarjolla olevista mobiilialustoista palvelun sovellus kannattaa tehdä. Käytännössä maailmalla suosittuja alustoja ovat iOS, Android ja Windows Phone, jotka kaikki eroavat toisistaan niin huomattavasti, että sovelluksen rakentaminen täytyy aloittaa lähes tyhjästä jokaisen alustan kohdalla. Yksinkertaista tapaa eri alustojen välillä toimivan natiivisovelluksen tekemiseen ei siis ole. Teknisistä eroistaan huolimatta palvelun tavoitettavuuden kannalta tärkein tekijä on kuitenkin kunkin alustan suosio loppukäyttäjien keskuudessa. Alustojen välistä suosion kehitystä voidaan tarkastella useasta eri näkökulmasta, mutta palvelun tulevaisuutta ajatellen ehkä tärkeimpänä vertailutekijänä ovat kutakin alustaa edustavien laitteiden myyntimäärät. Myyntimäärät antavat viittauksen alustojen käyttäjämäärien kehityksestä tulevaisuudessa. Myyntimääriä voidaan tarkastella esimerkiksi Business Insider -verkkosivuston kokoamasta kuvasta 1. Kuvassa 1 esitetyn kaavion mukaan globaalisti myytyjen laitteiden välisessä kilpailussa Android on saavuttanut ehdottoman johtoaseman ja onkin tämän vertailutekijän perusteella paras valinta sovelluksen alustaksi.



Kuva 1. Kaavio globaalista älypuhelinten myynnin kehityksestä [Blodget 2013].

Ideaalitapauksessa sovelluksen kehitys suoritettaisiin yhtä aikaa Android- sekä iOS-alustoille, jolloin palvelu voisi hyödyntää molempien alustojen parhaita puolia sekä tavoittaa huomattavan suuren käyttäjäkunnan. Yhtäaikainen kehitys molemmille

alustoille on kuitenkin huomattavasti aikaa vievä prosessi, eikä se ole tämän projektin kannalta oleellista. Mahdollisimman suuren käyttäjäkunnan tavoittamiseksi, mobiilisovellusten kehittämiseen on nykyään olemassa myös web-pohjaisia, HTML5-teknologiaa hyödyntäviä ratkaisuja. Näiden avulla sovellus olisi mahdollista toteuttaa web-pohjaiseksi, jolloin itse käyttäjän laitteella ajettava natiivisovellus toimisi käytännössä vain web-selaimena. Web-pohjaisissa mobiilisovelluksissa on kuitenkin huomattavia rajotteita natiivisovelluksiin verrattuna, jonka vuoksi tarjolla olevien ratkaisujen käyttö olisi palvelun kannalta epäedullista.

2.3 Alustavan kokoonpanon suunnittelu

Palvelun rakentaminen aloitetaan suunnittelemalla sen ensimmäinen julkaisukelpoinen versio. Kyseinen versio tulee kattamaan vain aivan palvelun kannalta välttämättömän perustoiminnallisuuden, jonka perusteella voidaan arvioida, onko palvelun jatkokehitys järkevää ja löytyykö kyseiselle palveluidealle kysyntää. Rakenteellisesti sosiaalisen median mobiilipalvelu asettaa muutamia vaatimuksia, kuten tarpeen palvelimelle sekä tiedon välitykselle loppukäyttäjän ja palvelimen välillä. Näiden vaatimusten vuoksi, palvelu onkin loogista jakaa kolmeen pääkomponenttiin, joita ovat mobiilisovellus, web-rajapinta sekä taustajärjestelmä.

Mobiilisovellus on komponenteista lähinnä loppukäyttäjää, ja se tulee rakenteellisesti sisältämään käyttöliittymän, sovelluksen sisäisen tietokannan sekä web-rajapinnan hallintaan tarvittavan toiminnallisuuden. Natiivisovellus tullaan rakentamaan Android-alustalle ja sen rakentamisessa tullaan hyödyntämään niin Androidin mukana tulevia apukirjastoja, kuten myös kolmannen osapuolen ActionBarSherlock- sekä AndroidBootstrap-kirjastoja. Web-rajapinta puolestaan tulee olemaan palvelimen sekä mobiilisovelluksen välille sovittu, tiedonsiirrossa käytettävä kokoelma kutsuja ja vastauksia. Rajapinnan tiedonsiirto tullaan toteuttamaan REST-arkkitehtuurimallin mukaisesti HTTP-protokollan ylitse siirrettävillä, JSON-muotoon paketoiduilla viesteillä. Viimeisenä komponenttina palvelun taustajärjestelmä tullaan rakentamaan Ruby On Rails-ohjelmistokehyksellä, hyödyntäen kolmannen osapuolen RailsAPI-pakettia, jonka avulla Rails-ohjelmistokehyksestä saadaan poistettua palvelun kannalta turhia ominaisuuksia ja yksinkertaistettua alustavaa toteutusta. Taustajärjestelmän tiedon talletuksessa tullaan hyödyntämään avoimen lähdekoodin MariaDB-tietokantapalvelinta sekä Redis-palvelinohjelmistoa.

Loppukäyttäjän kannalta palvelun alustavan toteutuksen tulisi mahdollistaa palvelulle suunniteltu perustoiminnallisuus, eli käyttäjän pitäisi pystyä rekisteröimään uusi käyttäjätunnus palveluun sekä kirjautumaan sisään luodulle tunnukselle. Palveluun tunnistautumisen lisäksi sen tulisi mahdollistaa käyttäjälle uuden tapaamispisteen luonti, sekä tarjota selkeä tapa selata muiden käyttäjien palveluun lisäämiä tapaamispisteitä. Käyttäjälle tarjotaan myös mahdollisuus ilmoittautua toisen käyttäjän ilmoittaman tapaamispisteen kerääjäksi, jolloin pisteen luojalle lähetetään tapahtumasta ilmoitus ja prosessi kahden käyttäjän välisen ystävyyssuhteen luonnista saa alkunsa. Tapaamispisteen luonnissa määriteltyjen tietojen perusteella tulee olla mahdollista muodostaa kahden käyttäjän välinen ystävyyssuhde tapaamalla pisteen luoja ennalta sovitussa sijainnissa, jolloin molemmat käyttäjät saavat toisensa omiin kontaktilistoihinsa. Kontaktilistaan kerätyille kontakteille tulee olla mahdollista lähettää sekä vastaanottaa tekstimuotoisia viestejä, ja kontaktilistan kontakteja tulee olla mahdollista poistaa, jolloin kahden käyttäjän välinen suhde purkautuu.

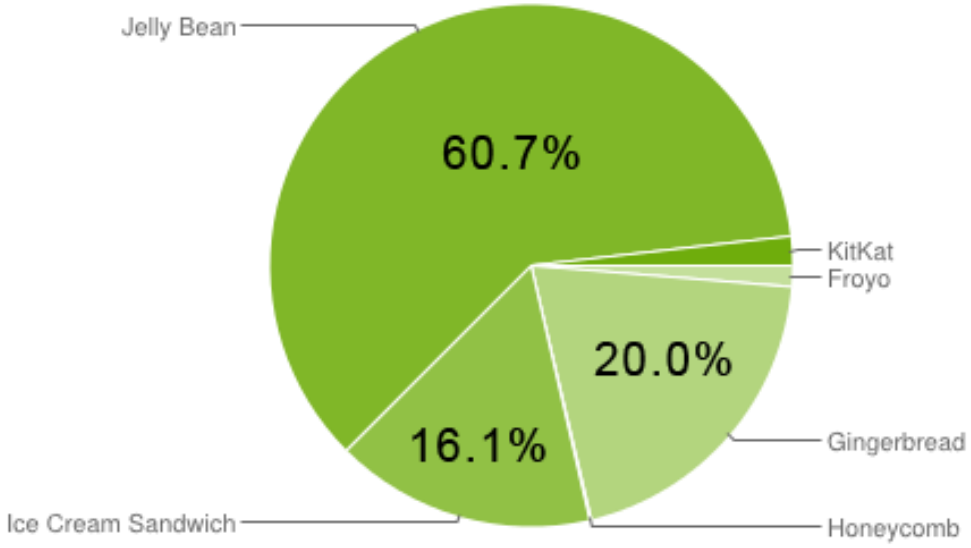
3 Palvelun Android-sovellus

Sosiaalisen median palvelussa tärkeimpänä osana toimii itse loppukäyttäjälle näkyvä sovellus. Loppukäyttäjän kokemus palvelusta koostuu pitkälti siitä, kuinka vaivattomasti käyttäjä voi sovellusta käyttää ja kuinka miellyttäväksi käyttäjä näkee sovelluksen ulkoasun. Jotta käyttäjälle voidaan tarjota paras mahdollinen käyttökokemus palvelusta, tulee palvelun sovelluksen teknisessä toteutuksessa ottaa huomioon useita seikkoja, kuten tuetut käyttöjärjestelmäversiot, sovelluksen käyttöliittymän toiminta sekä sovelluksen sisäisten komponenttien rakentaminen toimiviksi ja käyttäjälle näkymättömiksi.

Sovelluksen ulkoasun eli käyttöliittymän ja sen teknisen toimivuuden lisäksi on mobiilisovellusten kehityksessä huomioitava käyttäjien kärsimättömyys sovelluksen kaatuilun suhteen. Mikäli sovellus kaatuu ensimmäisen kahden käyttökerran aikana, on hyvin todennäköistä, ettei kyseinen käyttäjä enää jatka sovelluksen käyttöä. Jotta sovelluksen kaatuilemista saadaan ehkäistyä, on kehitysvaiheessa painotettava sovelluksen huolelliseen testaamiseen. Android-pohjaisten sovellusten testaamiseen on kuitenkin olemassa useita tapoja ja työkaluja, joista palvelun kannalta sopivien valinta on oma ongelmansa.

3.1 Sopivan Android-version valinta

Sosiaalisen median sovelluksille eräs hyvin tärkeä asia on kyky mahdollistaa niiden käyttö mahdollisimman suurelle yleisölle. Tämän vuoksi sovellusta suunniteltaessa tulee ottaa huomioon maailmalla olevien Android-laitteiden jakautuminen eri käyttöjärjestelmäversioihin. Kuten alla olevasta kuvasta 2 nähdään, suurin osa käytössä olevista Android-laitteista käyttää Android-käyttöjärjestelmän versiota 4.0 tai sitä uudempaa. Tämän vuoksi sovellusta tehdessä onkin järkevää painottaa erityisesti uudempien Android-versioiden tukemiseen. Kuvasta 2 kuitenkin nähdään, että huomattavan suuri osa käyttäjistä käyttää vielä nykyäänkin Androidin, vuonna 2010 julkaistua Gingerbread 2.3 versiota.



Kuva 2. Android-versioiden käyttöosuudet tammikuussa 2014 [Android Developer 2014].

Mahdollisimman suuren käyttäjämäärän tavoittamiseksi olisi siis järkevää lisätä sovellukseen tuki kyseiselle 2.3-versiolle, jonka jälkeen sovellusta voisi käyttää käytännössä kuka tahansa. Tämä johtuu siitä, että Androidin 2.3-versiota vanhempien Android-versioiden käyttäjämäärät ovat lähes olemattomia.

Androidin 2.3-version tukemiseksi on sovellusta kehitettäessä otettava huomioon, ettei kyseinen versio tue kaikkia 4.0- ja sitä uudempien versioiden ominaisuuksia. Tämä ongelma on kuitenkin ratkaistavissa suhteellisen helposti käyttämällä kolmannen osapuolen ActionBarSherlock-kirjastoa sekä Androidin mukana tulevia apukirjastoja. Näiden kirjastojen avulla on mahdollista saada rakennettua sovellus, jonka

toiminnallisuus on lähes identtinen niin vanhemmilla kuin uusimmillakin Android-käyttöjärjestelmän versioilla.

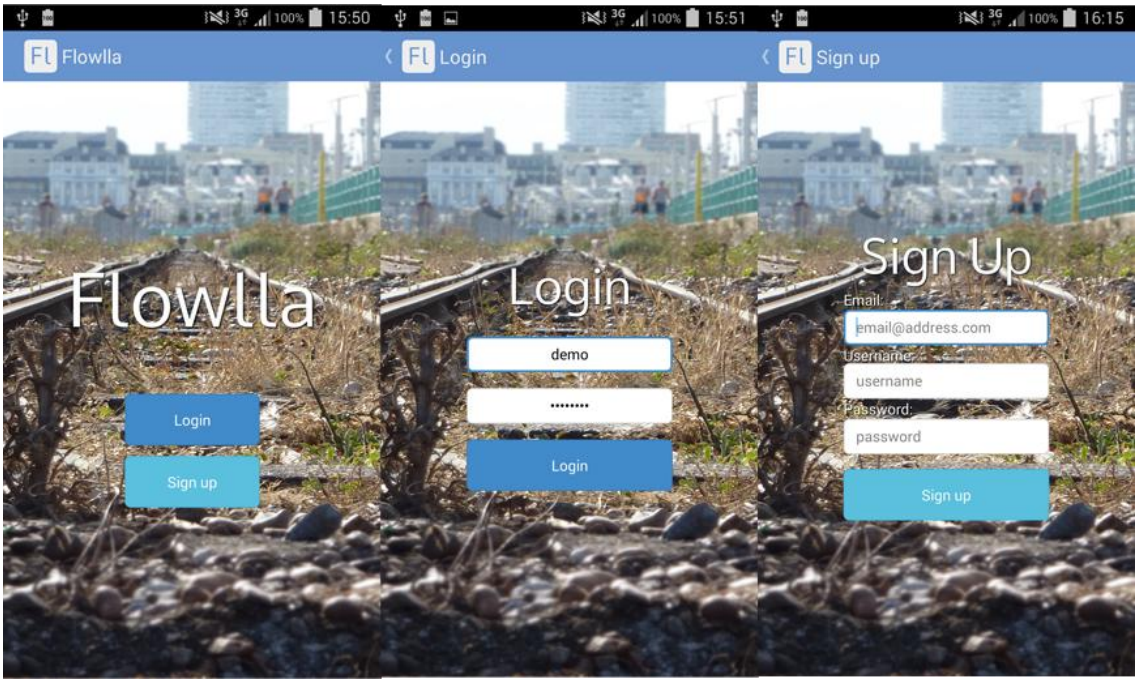
3.2 Käyttöliittymä ja käyttäjäkokemus

Käyttöliittymän osuus sosiaalisen median palveluissa on hyvin tärkeä, sillä yleisesti ottaen käyttäjät kuluttavat suurimman osan sovelluksen käyttöajasta sen käyttöliittymän selaamiseen. Palvelulle sopivan käyttöliittymän suunnittelussa on myös hyödyllistä pitää mielessä mahdollinen jatkokehitys, kuten esimerkiksi sovelluksen kääntäminen muille kielille. Kääntäminen muille kielille on mahdollista tehdä helpoksi, mikäli muistaa sovellusta kehittäessä käyttää Androidin tarjoamaa strings.xml-tiedostoa kaikkien sovelluksen käyttöliittymässä näkyvien tekstien keskittämiseksi sen sijaan, että ripottelisi eri näkymissä olevia tekstejä ympäriinsä esimerkiksi sovelluksen koodin sekaan.

Mobiilisovellusten kehityksessä laitteiden ruutujen pieni koko asettaa haasteita eri käyttöliittymäkomponenttien luomiselle, sillä pitkät tekstit eivät aina mahdu kaikkialle, johon niitä mahdollisesti tarvittaisiin. Tekstien sijaan mobiilisovellusten kehityksessä on hyvin yleinen käytäntö korvata ne sopivien ikonien avulla, mutta tämä vastaavasti asettaa vaatimuksen laadukkaiden ikonien luomiselle. Ongelman ratkaisemiseksi on sovelluksessa päätetty käyttää kolmannen osapuolen AndroidBootstrap-kirjastoa, joka tarjoaa käyttöliittymän rakentamiseen muutamia laadukkaita komponentteja sekä kattavan ikonikirjaston. AndroidBootstrap-kirjaston avulla sovelluksen visuaalinen ulkoasu saadaan näyttämään ammattimaisen selkeältä ilman aikaisempaa kokemusta graafisesta suunnittelusta. Ikonikirjaston lisäksi AndroidBootstrap-kirjasto sisältää muutamia visuaalisesti miellyttävän näköisiä käyttöliittymäkomponentteja, kuten nappeja ja tekstikenttiä, joiden avulla Androidin vakiokomponentit on helppo korvata toiminnallisuutta menettämättä.

Sovelluksen käyttöliittymää suunniteltaessa on pidetty mielessä loppukäyttäjän käyttökokemus, jonka sulavuuteen voidaan vaikuttaa tekemällä käyttöliittymässä etenemisestä suoraviivaista. Uuden käyttäjän näkökulmasta, sovellus tarjoaa ensimmäisenä näkymänään yksinkertaisen päävalikon, joka on varmasti käyttäjälle entuudestaan tuttu, mikäli hänellä on yhtään aikaisempaa kokemusta mobiilisovellusten käytöstä. Kuvassa 3 näkyvässä päävalikossa käyttäjälle tarjotaan

mahdollisuus joko kirjautua palveluun aikeisemmin luomallaan tunnuksella tai rekisteröidä kokonaan uusi tunnus. Käyttäjätunnusten luonti mobiilipalveluissa on mielenkiintoinen ongelma, jossa sovelluksen kehittäjän tarvitseman tiedon ja käyttäjän kärsivällisyyden väliltä täytyy löytää sopiva välikohta. Useat mobiilipalvelut vaativat rekisteröitymiseen esimerkiksi käyttäjän koko nimeä ja eräät suosituimmista viestintäpalveluista kuten, WhatsApp, haluavat jopa varmentaa käyttäjän puhelinnumeron tekstiviestin avulla ennen kuin uuden tunnuksen luonti on mahdollista. Käyttäjän kannalta edullisinta on, mitä vähemmän tietoja sovelluksen rekisteröitymisnäkymään tarvitsee täyttää ja mitä nopeammin tästä kynnyksestä pääsee ylitse.

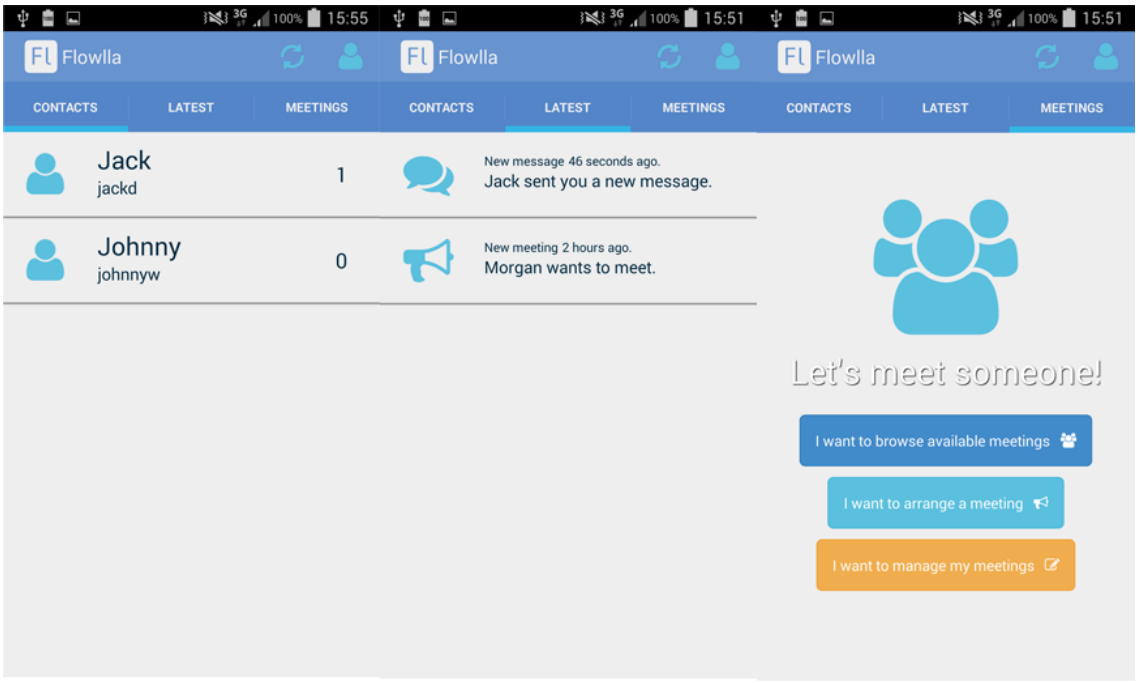


Kuva 3. Palvelun Android-sovelluksen päävalikko.

Useiden internetsivustojen sekä mobiilipalvelujen rekisteröitymisnäkymien vertailun tuloksena palveluun rekisteröitymiseen valittiin kolme vaadittavaa kenttää. Sähköpostiosoitteen vaatimiseen käyttäjältä päädyttiin lähinnä siksi, että se mahdollistaa tunnuksen palauttamisen, mikäli käyttäjä unohtaa salasanansa. Teknisesti käyttäjätunnuksen nimen vaatiminen ei olisi ollut yhtä välttämätöntä kuten sähköpostiosoitteen ja salasanan, vaan se olisi mahdollista korvata esimerkiksi satunnaisesti arvotulla merkkijonolla, mutta palvelun luonteen vuoksi sitä päädyttiin kuitenkin vaatimaan. Käyttäjätunnuksen korvaaminen esimerkiksi puhelinnumeron

varmennukseen perustuvalla mekaniikalla olisi myös ollut yksi vaihtoehto, joka on ollut viime aikoina myös esillä internetin blogisivustoilla. [Wester 2013]

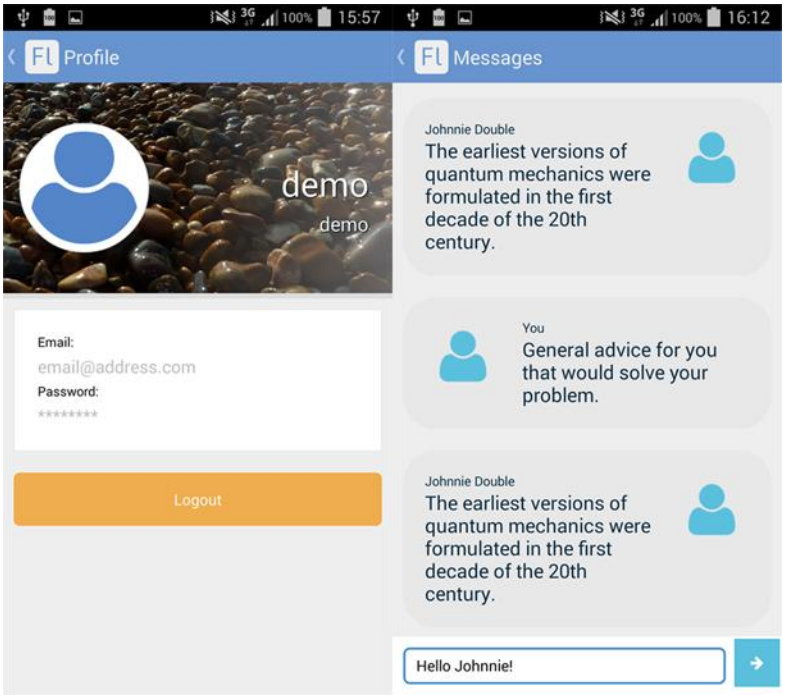
Sisään kirjautumisen jälkeen käyttäjä pääsee käsiksi sovelluksen varsinaiseen toiminnallisuuteen, joka on jaettu kolmelle eri välilehdelle. Välilehtirakenteen käyttäminen sovelluksen sisällön näyttämisessä on yleinen tapa sosiaalisen median mobiilisovelluksissa, ja se nopeuttaa sovelluksen käyttöä. Tämä johtuu siitä, että välilehdeltä toiselle siirtyminen on Android-alustaa käyttävissä sovelluksissa huomattavasti nopeampaa verrattuna kokonaan uuden näkymän avaamiseen.



Kuva 4. Palvelun Android-sovelluksen välilehdet.

Ensimmäinen välilehti, joka käyttäjälle näytetään aina sovellusta avattaessa, on kuvassa 4 keskimmäinen. Tältä välilehdeltä käyttäjä näkee kaikki uudet tapahtumat sovelluksessa. Jos esimerkiksi toinen käyttäjä lähettää viestin, tulee ilmoitus uuden viestin saapumisesta tähän välilehteen. Kuvassa 4 vasemmalla näkyvä välilehti listaa kaikki käyttäjän tapaamat ystävät sekä mahdollistaa viestien lähettämisen kullekin ystävälle. Vastaavasti kuvan 4 oikean reunan välilehti tarjoaa työkaluja uusien käyttäjien tapaamiseen. Sovelluksen käyttöliittymän oikeassa yläkulmassa olevien nappien avulla käyttäjä voi päivittää sovelluksen sisällön palvelimelta sekä muokata omaa käyttäjätunnustaan. Sisällön päivittäminen on useissa maailmalla suosituissa mobiilisovelluksissa toteutettu vetotoiminnon avulla, jolloin käyttäjä voi pyyhkäistä

sormella ruutua ylhäältä alas, joka vastaavasti käynnistää päivitysoperaation. Vastaavan toiminnallisuuden lisäämistä tähän sovellukseen ei katsottu tarpeelliseksi käsiteltävän sisällön luonteen vuoksi.



Kuva 5. Palvelun Android-sovelluksen profiilinäkymä, sekä viestinäkymä.

Kuvassa 5 näkyvästä päänäkymästä olevan profiili-ikonin kautta käyttäjän on mahdollista päästä kuvan 5 vasemman puoleiseen profiilinäkymään. Kuvassa 5 vasemmalla olevassa alustavan kokoonpanon näkymässä käyttäjälle tarjotaan mahdollisuus muokata nimeään sekä vaihtaa tunnuksen sähköpostiosoite ja salasana. Tavallisesti käyttäjän sisään kirjautumisen jälkeen sovellus tallentaa sisään kirjautuneen käyttäjän istunnon puhelimeen, jonka vuoksi sisään kirjautumista ei vaadita jokaisella sovelluksen avauskerralla. Mikäli käyttäjä haluaa kirjautua sisään toisella käyttäjätunnuksella hänen tulee ensin kirjautua ulos edelliseltä tunnukselta. Uloskirjautumisen mahdollistava nappi on sovelluksessa sijoitettu käyttäjän profiilisivulle, joka on muihin sosiaalisen median mobiilisovelluksiin verrattaessa yleinen tapa.

Käyttäjän pääasiallinen interaktio muiden palvelun käyttäjien kesken on viestien lähettäminen. Kontaktilistan kautta käyttäjän on mahdollista avata kuvassa 5 oikealla näkyvä viestiketju haluamansa käyttäjän kanssa. Viestiketjut ovat kahden käyttäjän välisiä, eikä toiminnallisuutta ryhmäviestinnälle ole suunniteltu ensimmäiseen

sovelluksen versioon. Viestinäkymä on mahdollista avata kontaktilistan nimiä painamalla. Sovelluksen ensimmäisessä versiossa viestiketjut tukevat vain mahdollisuutta lähettää tekstimuotoisia viestejä palvelun muille käyttäjille, mutta kuvien ja äänen lähettäminen ovat mahdollisia jatkokehitysominaisuuksia.

3.3 Sovelluksen yhteys taustajärjestelmään

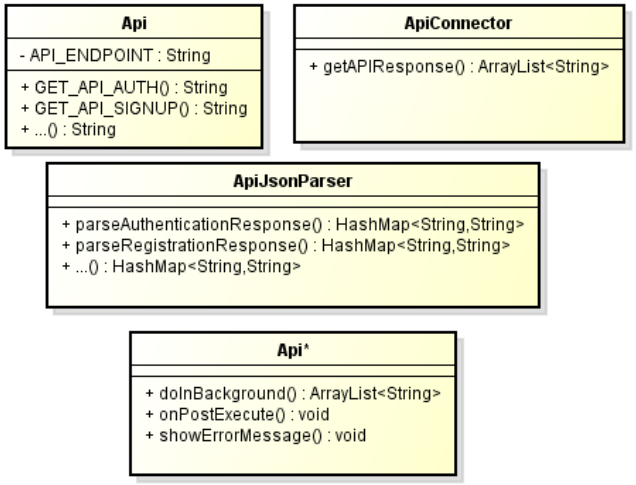
Käytännössä kaikki sosiaalisen median mobiilisovelluksilla tehtävät toimenpiteet vaativat yhteyden taustajärjestelmään. Taustajärjestelmän kautta voidaan esimerkiksi välittää viestejä käyttäjien välillä sekä suorittaa käyttäjätietojen talletusta. Koska taustajärjestelmään tulee olla pääsy kaikilla sovelluksen käyttäjillä, sen tulee sijaita palvelimella, johon on mahdollisuus yhdistää internet-yhteyden avulla. Sovelluksen kyky suorittaa toimenpiteitä puhelimen internet-yhteyttä käyttäen vaatii sopivien ohjelmistokomponenttien käyttöä sovellusta rakennettaessa. Internet-yhteyden käyttö mobiilisovelluksissa rasittaa myös puhelimen akkua, jonka vuoksi yhteydenottoja on hyödyllistä yrittää välttää sekä niputtaa, mikäli mahdollista.

Mobiilisovelluksissa yhteys taustajärjestelmiin toteutetaan lähes poikkeuksetta HTTP-protokollalla muodostettavalla yhteydellä taustajärjestelmän web-rajapintaan. Android-alusta tarjoaa HTTP-yhteyksien muodostamiseen käytännössä kahta eri ohjelmistokomponenttia. Android-alustan tarjoamat ohjelmistokomponentit ovat Apache HTTP Client sekä HttpUrlConnection. Näistä komponenteista Apache HTTP Client on vakaampi, mutta tarjoaa sovelluksen kannalta turhan laajan toiminnallisuuden, jonka vuoksi käyttöön on valittu HttpUrlConnection-komponentti. Komponentin valintaan vaikuttavana tekijänä on myös Android-dokumentaation suosittelu sen käytölle, mikäli sovellusta tullaan kehittämään Android-käyttöjärjestelmän 2.3-versiolle tai sitä uudemmalle. Muissa tapauksissa Apache HTTP Client olisi saattanut olla toimiva vaihtoehto.

Rakenteellisesti Android-alustan tarjoamaa HttpUrlConnection-komponenttia käytetään sovelluksen koodin rajapintaluokkien sisällä. Näiden luokkien toiminnallisuus on toteutettu niin, että ne mahdollistavat web-rajapintakutsujen paketoimisen Java-ohjelmointikielen ArrayList-säiliöön, jolloin useita rajapintakutsuja voidaan suorittaa keskitetysti niputettuna ja näin vähentää puhelimen virrankäyttöä, koska yhdellä internet-yhteydenotolla voidaan suorittaa useita toimintoja peräkkäin. Käytännössä

tämä toimii niin, että käyttäjän suorittaessa toimintoja sovellus rakentaa tarvittavien tietojen välittämiseksi HTTP GET- ja POST-kutsuja, jotka pakataan säiliöön. Säiliö puretaan lähettämällä sen sisältämät kutsut taustajärjestelmän palvelimelle sopivalla hetkellä niiden kiireellisyyden mukaan.

Sovelluksen suorittaman rajapintakutsun paluuviestinä, taustajärjestelmä lähettää JSON-muotoon pakatun vastauksen, joka sisältää kaikille kutsuille yhteistä metatietoa, kuten onnistuiko kyseisen kutsun suorittaminen, sekä tietysti kullekin kutsulle ominaiset vastaustiedot. JSON-muotoon pakatun tiedon purkaminen ja välittäminen käyttäjälle vaatii ohjelmistokomponentin, joka kykenee käsittelemään suuria tietomääriä nopeasti, ilman että se aiheuttaa sovelluksen kaatumista. Kyseisen toiminnallisuuden toteuttamiseksi sovelluksessa tehtiin vertailua kahden eri kolmannen osapuolen JSON Parser-kirjaston välillä. Vertailua suoritettiin Apache 2.0-lisenssoitujen GSON-, sekä Jackson-kirjastojen välillä, joista sovelluksessa otettiin lopulta käyttöön Jackson, sen nopeuden ja helppokäyttöisyyden vuoksi. Jackson kirjasto on sovelluksessa pakattu kuvassa näkyvän ApiJsonParser-luokan sisälle, joka sisältää kaikkien taustajärjestelmän paluuviestien käsittelyihin vaadittavat koodit.



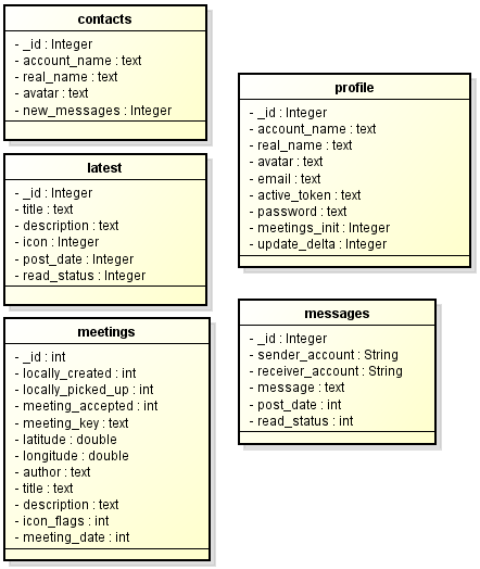
Kuva 6. Palvelun Android-sovelluksen rajapintaluokat.

Android-sovelluksen puoleinen web-rajapinnan käsittely on toteutettu kuvan 6 mukaisella luokkarakenteella, jossa Api-luokkaan on paketoitu taustajärjestelmän URI-osoite sekä toiminnallisuus kunkin rajapintakutsun tarvitseman osoitteen rakentamiseksi. Vastaava rakenne yksinkertaistaa sovelluksen jatkokehitystä ja helpottaa taustajärjestelmän osoitteen vaihtoa, sillä kaikki asiaan liittyvä logiikka on

keskitetty yhteen pisteeseen. ApiConnector-luokka sisältää itse yhteyden muodostamiseen vaadittavan logiikan ja perii HttpUrlConnection-luokan toiminnallisuuden. Jokaista rajapintakutsutyyppiä varten on sovelluksessa luotu oma Api-luokkansa, joka hyödyntää Android-alustan tukea moniajolle. Tämä mahdollistaa tiedonvälityksen taustajärjestelmän ja sovelluksen välillä ilman, että sovelluksen käyttöliittymän käytössä esiintyisi häiriötä. Moniajotoiminnallisuus on toteutettu perimällä Androidin AsyncTask-luokan toiminnallisuus.

3.4 Sovelluksen paikallinen tietokanta

Sovellusta käytettäessä, taustajärjestelmän palvelimelta ladattavaa tietoa täytyy pystyä säilömään käyttäjän puhelimeen, jotta sovelluksen uudelleenkäynnistäminen ei vaatisi kaiken tiedon uudelleen lataamista. Tietojen pitkäaikaista säilömistä varten Android-alustassa on sisäänrakennettu tuki SQLite-tietokannoille. Sovelluksen sisäiseen tietokantaan tallennetaan kaikki käyttäjän käyttöliittymässä tekemät muutokset, jonka jälkeen ne päivitetään taustajärjestelmään sopivan ajan kuluttua.



Kuva 7. Palvelun Android-sovelluksen sisäinen tietokantarakenne.

Rakenteellisesti sovelluksen sisäinen SQLite-tietokanta on jaettu kuvassa 7 näkyviin viiteen eri tauluun, joista jokainen pitää sisällään eri sovelluksen toiminnallisuuteen vaadittavia tietoja. Ensimmäistä kertaa sisään kirjautuessaan, käyttäjän tunnukselle luodaan uusi merkintä profiili tauluun. Samalla tauluun tallennetaan palvelimelta haettu,

aktiivisen istunnon avain, sekä käyttäjän salasanatiiviste uusien istuntojen avaamiseksi. Salasanan tallentaminen suojaamattomana tietokantaan on lähes poikkeuksetta huono ratkaisu, johon suurten yritysten sortuminen on aiheuttanut myös kritiikkiä mediassa [Sean Hollister 2014], tämän vuoksi sovellus ei missään vaiheessa tallenna käyttäjän suojaamatonta salasanaa, vaan muuntaa sen turvallisempaan muotoon hajautusalgoritmin avulla. Lisäsuojan antamiseksi, sovelluksen tietokantaan pääsy on estetty muilta puhelimeen asennetuilta sovelluksilta Android-alustan tarjoaman, sovelluksen manifest-tiedostosta hallittavan, toiminnallisuuden avulla.

Käyttäjän painaessa käyttöliittymän päivitysnappia sovellus lähettää profiilitaulun delta-arvon taustajärjestelmän palvelimelle, jolta saadaan vastauksena listaus viimeksi muuttuneista tiedoista sekä uusi delta-arvo. Uudet tiedot sovellus tallentaa omiin tietokannan tauluihinsa, joista ne tämän jälkeen lisätään käyttöliittymään käyttäjän nähtäviksi. Kyseinen päivitysmekanismi on myös kaupallisissa sovelluksissa käytetty tapa, jonka avulla voidaan välttää turhaa tiedon välittämistä sovelluksen ja taustajärjestelmän välillä [Dropbox API 2014]. Päivitysmekanismi saa nimensä matemaattisesta, kahden arvon erotusta merkitsevästä delta-merkinnästä. Ensimmäisellä käyttäjän kirjautumiskerralla tietokannan delta-arvo on nolla, jolloin sovellus pyytää taustajärjestelmältä kaikki käyttäjän kontaktit ja viestit sekä muut tiedot, mikäli niitä on saatavilla. Tämä mahdollistaa saman käyttäjätunnuksen käytön jatkamisen esimerkiksi puhelimen vaihtamisen jälkeen.

Tietokannan taulujen hallinta on sovelluksessa toteutettu DatabaseHelper-luokan avulla, joka perii Androidin mukana tulevan SQLiteOpenHelper-luokan toiminnallisuuden. SQLiteOpenHelper tarjoaa yksinkertaisen tavan luoda ja hallita sovelluksen sisäistä tietokantaa. Jokaiselle sovelluksen tietokannan taululle on luotu oma luokkansa, joka on määritetty perimään Androidin ContentProvider-luokan toiminnallisuus, tämä yksinkertaistaa tiedon hakua ja talletusta kuhunkin tauluun. Tiedon välitys sovelluksen käyttöliittymään on toteutettu hyödyntämällä Androidin CursorLoader-luokkaa, joka vastaavasti hyödyntää Androidin moniajoa tietokannan tietojen hakemisessa eikä täten aiheuta hidastelua sovelluksen käyttöliittymän käyttämiseen.

3.5 Käyttäjien välinen suhteen rakentaminen ja purkaminen

Palvelun käyttäjien tapaaminen on tapa, jolla kahden käyttäjän välinen suhde voidaan muodostaa. Jotta palvelu voi varmistaa, että kaksi käyttäjään oikeasti tapaavat, on molemmilta käyttäjiltä saatava tietoa heidän sijanneistaan tai vaihtoehtoisesti käyttäjien puhelinten välillä täytyy välittää tietoa tavalla, joka ei ole mahdollista pitkän etäisyyden päästä. Mahdollisia teknisiä ratkaisuja vastaavan toiminnallisuuden toteuttamiseksi on useita, kuten NFC, GPS ja Bluetooth. Palvelun alustavassa versiossa näistä on kuitenkin valittu yksinkertaisin eli GPS-sijainnin käyttö käyttäjien tapaamisen varmentamiseksi. Käytännössä ratkaisu vaatii molempien käyttäjien saapumisen tapauspisteen läheisyyteen määritettynä ajankohtana. Yksinkertaisuudestaan huolimatta ratkaisu on palvelun kannalta riittävän toimiva. Hyvänä puolena GPS-sijainnin käytössä on myös se, että GPS-toiminnallisuus on sisäänrakennettuna lähes jokaiseen Android-puhelimeen, toisin kuin esimerkiksi NFC.

Eräs palvelun tavoitteista on luoda ympäristö, jossa käyttäjän kontaktilista sisältää vain ne kontaktit, joiden kanssa palvelun käyttäjä haluaa oikeasti olla tekemisissä. Jottei käyttäjän kontaktilista paisuisi muiden sosiaalisen median palveluiden tapaan, on käyttäjän kontaktilistalta voitava poistaa kontakteja, joihin käyttäjä ei enää pidä yhteyttä. Tämän toiminnallisuuden toteuttamiseksi sovellus käyttää sisäiseen SQLite-tietokantaan tallennettujen viestien aikaleimatietoja kontaktilistan automaattiseen siivoamiseen.

4 Palvelun web-rajapinta

Käyttäjän puhelimella ajettavan sovelluksen sekä palvelimella ajettavan taustajärjestelmän välinen tiedonsiirto tapahtuu palvelun web-rajapinnan välityksellä. Rajapinnan suunnittelussa on otettu huomioon muutamia eri toteutusvaihtoehtoja, joista on valittu palvelun kannalta sopivin.

4.1 Vaihtoehtoiset tiedonsiirtokanavat

Tiedonsiirto sovelluksen ja palvelimen välillä olisi voitu toteuttaa usealla eri tavalla, joista jokaisella on omat hyvät ja huonot puolensa. Mikäli sovelluksen ja palvelimen

välillä olisi ollut tarve liikuttaa jatkuvaa tietovirtaa, olisi kiinteän TCP-yhteyden käyttö ollut HTTP-protokollaa järkevämpää. Sovelluksen rakenne ja käyttötapaukset kuitenkin suosivat satunaisten, lyhyiden yhteydenottojen avaamista, jollaiseen HTTP-protokolla on yksinkertaisin ratkaisu. HTTP-protokollan käytön yksinkertaisuus johtuu siitä, että Android-alusta tarjoaa vakiona ohjelmistokomponentteja HTTP-yhteyksien muodostamiseen ja HTTP-protokollaa tukevia palvelinohjelmistoja on helposti saatavilla. HTTP-protokollan käyttö yksinkertaistaa myös sovelluksen ja palvelimen välisen tiedon suojaamista, sillä yhteydenotot on mahdollista vaihtaa käyttämään suojattua HTTPS-protokollaa. Jatkuvien TCP-yhteyksien käyttö rajapinnan liikenteeseen asettaisi myös palvelimen rakenteelle vaatimuksia, joiden takia palvelinohjelmisto pitäisi ohjelmoida alusta alkaen itse sen sijaan, että käyttäisi valmiiksi saatavilla olevia palvelinohjelmistoja.

HTTP-protokollan käyttö rajapinnan tiedonsiirrossa tarjoaa myös palvelun jatkokehityksen kannalta mielenkiintoisia mahdollisuuksia, sillä sen julkaiseminen ulkopuolisille sovelluskehittäjille ei vaatisi erityisen suurta työmäärää. Rajapinnan julkaiseminen tarjoaisi kolmannen osapuolen kehittäjille mahdollisuuden rakentaa kokonaan oma käyttöliittymänsä palvelun käyttöön, joka puolestaan antaisi käyttäjille enemmän valinnan varaa. Web-pohjaisten rajapintojen julkaiseminen ulkopuolisille sovelluskehittäjille on nykyisten mobiilikehittäjien keskuudessa suhteellisen suosittu idea, joista esimerkkitapauksina voidaan mainita esimerkiksi Telegram Messenger ja Twitter.

4.2 HTTP-protokolla

Sovelluksen ja taustajärjestelmän välinen web-rajapinta on toteutettu HTTP-protokollaa hyväksikäyttäen. HTTP-protokolla eli Hypertext Transfer Protocol on TCP/IP-mallin sovelluskerroksen toteuttaja ja on tämän vuoksi sovelluskehityksen kannalta oleellisin TCP/IP-mallin kerros. HTTP-protokolla on alun perin Tim Berners-Leen 1989 ehdottamasta, World Wide Web-projektista alkunsa saanut protokolla, jonka ensimmäinen dokumentoitu versio julkaistiin 1991 [Cailliau 2011]. HTTP-protokolla kehitettiin 90-luvun aikana, ja sen viimeisin 1999 julkaistu versio 1.1 on nykyisen World Wide Webin tiedonsiirtoon käytetty perusta.

REST-arkkitehtuurityylin asettamia rajoitteita noudattamalla luotu HTTP-protokolla tarjoaa palvelimen ja asiakkaan välille yhdenmukaisen viestintärajapinnan määrittelemällä siirrettäville viesteille ennalta määrätyn rakenteen. HTTP-protokollan viesti voi olla joko asiakkaan pyyntö palvelimelle tai palvelimen vastaus asiakkaan pyyntöön. Protokollan mukaisen keskustelun aloittavana osapuolena on aina asiakas, joka esittää pyynnön haluamalleen palvelimella sijaitsevalle resurssille käyttäen Uniform Resource Identifier (URI) -merkkijonoa sekä määrittelemällä resurssille suoritettavan toimenpiteen käyttämällä HTTP-protokollan metodeja.

Sovelluksen tapauksessa palvelimelta haettavat resurssit eivät ole kiinteitä tiedostoja vaan ne luodaan pyynnön saapuessa palvelimelle. Palvelimelta haettaville resursseille suoritettavia toimenpiteitä kuvaavia HTTP-protokollan metodeja on yhteensä kahdeksan, joista palvelussa hyödynnetään pääasiassa kahta. GET-metodin avulla sovellus voi hakea palvelimelta tietoja, kuten käyttäjän ystävälistauksen. Palvelimen vastaus GET-metodin avulla suoritettuihin resurssikyselyihin sisältää metatietoa sekä itse sisältöosan, joka ystävälistauksen tapauksessa olisi JSON-muotoon pakattua tekstiä. Mikäli sovellus haluaa lähettää tietoa palvelimelle, tulee sovelluksen käyttää HTTP-protokollan POST-metodia, jolloin palvelimelle lähetettävä tieto voidaan pakata JSON-muotoisena tekstinä lähetettävän viestin sisältöosaan, toisin kuin GET-metodin kohdalla, jolloin viestiosa jää tyhjäksi.

4.3 Web-rajapinnan esittely

Palvelun web-rajapinta mahdollistaa JSON-muotoon pakatun tiedon välittämisen palvelimen ja sovelluksen välillä. Sovelluksen palvelimelle lähettämät pyynnöt noudattavat alla kuvattua URI-syntaksia.

```
https://<API-osoite>/<operaatio>?<parametrit>
```

Pyynnön saatuaan palvelin rakentaa sille sopivan vastauksen ja lähettää sen sovellukselle paluuviestinä. JSON-muotoon pakattu vastaus sisältää rakenteellisesti aina kaksi kenttää, joista sovellus voi päätellä onnistuiko pyyntö vai ei. Ensimmäinen jokaiseen paluuviestiin pakattu kenttä on result, josta sovellus näkee, onnistuiko kutsun suoritus vai ei. Toinen kiinteä kenttä on responseCode, joka sisältää numeroarvon, jonka perusteella sovellus saa tietoonsa, miksi kutsun suoritus epäonnistui ja on näin

kykenevä näyttämään käyttäjälle sopivan virheilmoituksen. Mikäli käyttäjä esimerkiksi kirjoittaa käyttäjätunnuksensa salasanan väärin, palvelin palauttaa tässä kentässä arvon, jonka perusteella sovellus voi ilmoittaa käyttäjälle salasanan olevan väärä. Kutsun suorittamisen onnistuessa kentän arvoksi asetetaan nolla, jolloin sovellus voi jatkaa paluuviestin tietojen purkamista. Näiden kahden paluuviestikentän lisäksi viestit sisältävät itse operaatiossa haetun tiedon, joka voi olla joko yksinkertainen merkkijono tai monikerroksinen listaus esimerkiksi käyttäjän profiilitiedoista. Taulukossa 1 on listattuna muutamia esimerkkitapauksia mahdollisista rajapintakyselyistä. Listatuissa esimerkkitapauksissa operaatio tarkoittaa rajapintakyselyn sisältämää merkkijonoa, jonka perusteella palvelin tietää, mitä sovellus haluaa ja mitä parametreja se osaa odottaa.

Taulukko 1. Esimerkkejä palvelun web-rajapintakyselyistä.

Operaatio	**Parametrit**	**Vastaus esimerkki**
auth	accountname, password	{"result":"success","responseCode":0,"token":"ABCDEFG123456"}
signup	email, password, accountname	{"result":"success","responseCode":0,"token":"ABCDEFG123456"}
friends	token	{"result":"success","responseCode":0,"data":["jackd","johnnyw"]}

5 Palvelun taustajärjestelmä

Sosiaalisen median mobiilipalveluissa taustajärjestelmä tarkoittaa käyttäjän käyttämästä sovelluksesta erillistä, palvelimella ajettavaa ohjelmistokokonaisuutta, johon palvelun käyttäjien puhelimet ovat yhteydessä internetin välityksellä. Taustajärjestelmän tarkoitus on palvelussa esiintyvän tiedon tallennus ja välitys sitä tarvitseville käyttäjille. Tämän lisäksi taustajärjestelmä vastaa palvelun käyttäjien tekemien toimenpiteiden hallinnasta, kuten suhteiden luonnista ja uusien tapaamisilmoitusten lisäämisestä.

Rakenteellisesti palvelun taustajärjestelmä koostuu useasta eri komponentista, jotka keskustelevat toistensa kanssa ja jotka olisi mahdollista sijoittaa usealle eri

palvelimelle. Palvelussa hyödynnettyjä komponentteja ei kuitenkaan tämän sovelluksen kohdalla ole sijoitettu usealle eri palvelimelle, sillä vastaava hajautus olisi palvelun alustavan version kokoon nähden tarpeetonta. Palvelun taustajärjestelmässä hyödyntäviä komponentteja on pääasiassa kolme, ja ne tarjoavat taustajärjestelmälle tiedon talletukseen sopivaa toiminnallisuutta, sekä yhdistävät taustajärjestelmän internetiin. Komponenteista ensimmäinen on Apache-palvelinohjelmisto, johon saatavilla olevan kolmannen osapuolen Phusion Passenger-lisäosan avulla on mahdollista ajaa Ruby On Rails-ohjelmistokehyksellä rakennettuja web-sovelluksia Apache-palvelinohjelmiston päällä. Muut taustajärjestelmässä hyväksi käytetyt komponentit tarjoavat taustajärjestelmälle kahta erityylistä tiedontalletusvarastoa.

Käyttäjätunnuksiin ja istuntoihin perustuvissa palveluissa käyttäjän istunto voidaan tunnistaa sisään kirjautumisen yhteydessä luodun avaimen avulla. Istunnon voimassaolon tarkistaminen avaimen perusteella on jokaisen käyttäjän suorittaman toimenpiteen yhteydessä kuitenkin hidasta ja palvelua rasittavaa, mikäli istunnon avain on taustajärjestelmässä tallennettuna normaaliin tietokantaan. Tähän ratkaisuksi on palvelussa otettu käyttöön Redis-niminen, muistissa toimiva avain-arvopalvelinohjelmisto, joka mahdollistaa käyttäjien istuntoavainten nopean ja kevyen noutamisen sekä tallentamisen. Avain-arvopalvelin ei kuitenkaan ole soveltuva ratkaisu suurien tietomäärien, kuten käyttäjän profiilitietojen tallentamiseen, jonka vuoksi palvelu hyödyntää toisena tiedontalletusvarastonaan MariaDB-tietokantapalvelinta.

5.1 Ruby On Rails-ohjelmistokehys

Ruby On Rails eli Rails-ohjelmistokehys on tanskalaisen David Heinemeier Hanssonin vuonna 2004 julkaisema, Ruby-ohjelmointikielen päälle rakennettu ohjelmistokehys. Rails-ohjelmistokehyksen suosio huipentui vuonna 2006, kun Apple ilmoitti sisällyttävänsä Rails-ohjelmistokehyksen Mac OS X-käyttöjärjestelmään. Tämän jälkeen Rails on käynyt läpi useita eri versioita, joiden mukana sen rakenne on muuttunut huomattavasti ja siihen on lisätty paljon uutta toiminnallisuutta. Vuonna 2008 Rails projektiin yhdistettiin toinen, kilpaileva ohjelmistokehys nimeltä Merb, joka johti Railsin 3.0 version julkaisuun. Tällä hetkellä Railsin kehitys on edennyt kesällä 2013 julkaistuun 4.0 versioon, joka lisää ohjelmistokehykseen tuen reaaliaikaisten tietovirtojen hallinnalle. [Ruby on Rails History A Look Back 2011; Ruby On Rails 2014]

Rails on niin sanottu täydenpinon ohjelmistokehys, joka tarkoittaa sitä, että se sisältää kaiken tarvittavan toiminnallisuuden web-sovelluksen rakentamiseksi, niin käyttöliittymästä tietokantaan. Rails-ohjelmistokehys rakentuu sisäisesti komponenteista, jotka ovat vastuullisia eri toiminnallisuuksien toteuttamisesta. Näistä komponenteista ensimmäinen on ActiveRecord, joka luo pohjan Railsin oliopohjaisen tiedon välittämiselle tietokantaan. Toisena komponenttina Rails tarjoaa ActiveResource-paketin, joka mahdollistaa oliopohjaisen tiedon välittämisen REST-arkkitehtuurityyliä noudattavien tietolähteiden välillä. Viimeiset kolme komponenttia ovat kutsujen reitityksen hoitava ActionPack, apukirjastoja tarjoava ActiveSupport sekä sähköpostipalveluja tarjoava ActionMailer. [Welcome to Rails 2014]

Rails-ohjelmistokehys on rakennettu hyödyntäen MVC-arkkitehtuuria (Model-View-Controller). MVC-arkkitehtuurin mallia (model) vastaava kerros on Railsissä toteutettu ActiveRecord-komponenttia käyttäen ja se mahdollistaa Ruby-olioiden välityksen tietokannan ja Rails-sovelluksen välillä. Ohjainkerros (controller) Railsissä hallinnoi sovellukseen sisään tulevat HTTP-kutsut, sekä välittää vastauksena tarvittavat tiedot mallikerroksesta. Railsin näkymäkerros (view) koostuu HTML-kuvauskielellä luoduista tiedostoista, joihin on sisällytetty Ruby-koodia ja jotka Rails muuntaa käyttäjän verkkoselaimen luettavaan muotoon, ActionView-komponettia käyttäen. [Welcome to Rails 2014]

MVC-arkkitehtuurin lisäksi Rails korostaa eri ohjelmistokehityksessä käytettäviä periaatteita, kuten Don't repeat yourself (DRY), joka Railsin kohdalla ilmenee ActiveRecord-komponentin yhteydessä, sillä sen käyttö vähentää saman tiedon kirjoittamista useaan paikkaan. Tämän lisäksi Rails korostaa myös Convention over Configuration (CoC) -periaatetta, joka tarkoittaa, että sovelluksen kehittäjän ei tarvitse tehdä ylimääräistä työtä yksinkertaisten ja yleisten tapausten ratkaisemiseksi, vaan Rails kykenee tekemään oletuksia kehittäjän koodin perusteella ja näin vähentämään kehittäjän työmäärää. Käytännössä CoC-periaate näkyy Railsissä, kun sovelluksen kehittäjä luo uuden mallikerroksen, joka kuvaa jotain yksittäistä asiaa kuten käyttäjää. Rails olettaa, että kehittäjä haluaa todennäköisesti tallentaa useita käyttäjiä yhden sijaan ja luo automaattisesti tietokantaan taulun nimellä käyttäjät. [Welcome to Rails 2014]

5.2 Rails-ohjelmistokehyksen valinta

Sopivan ohjelmistokehyksen valinta mobiilipalvelun taustajärjestelmää kehitettäessä on useasta eri tekijästä riippuvainen päätös. Palvelun tulevaisuuden ja taustajärjestelmän jatkokehityksen kannalta onkin olennaista, että valittu ohjelmistokehys mahdollistaa palvelun laajentamisen helposti, sillä palvelun julkaisun jälkeinen ohjelmistokehyksen vaihtaminen on epäkäytännöllistä ja hankalaa. Ensimmäinen palvelun ohjelmistokehyksen valintaan vaikuttaneista tekijöistä oli vaatimus web-pohjaiselle kommunikoinnille sovelluksen ja taustajärjestelmän välillä. Käytännössä tämä tarkoitti sitä, että käyttöön oli otettava jokin web-sovellusten rakentamiseen tarkoitettu ohjelmistokehys. Palvelun laadun vuoksi eräs tärkeä valintakriteeri oli valittavan ohjelmistokehyksen maksuttomuus, joka ei kuitenkaan osoittautunut ongelmaksi, sillä käytännössä kaikki suositut ohjelmistokehykset ovat avoimen lähdekoodin projekteja.

Maailmalla suosittuja, moderneja ohjelmistokehyksiä tutkiessa on mahdollista havaita tietty kolmikko, joka esiintyy huomattavan usein internetin keskustelupalstoilla, blogeissa ja uutisartikkeleissa. Nämä kolme ovat Ruby-kieltä hyödyntävä Ruby On Rails, Python-kieltä hyödyntävä Django sekä Javascript-kieltä käyttävä Express.js. Näistä kolmesta sovelluksen kannalta mielenkiintoisimpia vaihtoehtoja ovat Ruby On Rails sekä Express.js käyttämiensä kielien vuoksi. Rails- ja Express.js-ohjelmistokehyksiä vertailtaessa tulee ensimmäisenä vastaan kutakin kehystä hyödyntävän sovelluksen pystyttämisen monimutkaisuus. Tässä tapauksessa Rails tarjoaa huomattavasti Express.js-kehystä paremman alustan, sillä uusi, vasta luotu Rails-projekti sisältää vakiona lähes kaiken perustoiminnallisuuden, jonka avulla sovellukseen pääsee käsiksi internet-yhteyden avulla. Express.js kehys sen sijaan vaatii alleen Node.js-alustan asennuksen sekä hieman alustavaa koodia.

Ohjelmistokehyksiä vertailtaessa eräänä tärkeä seikkana toimii myös tietokantayhteyksien hallinta. Tietokannan hallinta on Rails-kehyksessä toteutettu hyödyntämällä ActiveRecord-komponettia, joka automatisoi huomattavan osan ohjelmoijan työstä ja tekee tietokannan käytöstä lähes näkymätöntä. Express.js-kehys sen sijaan vaatii tietokannan hallintaan liittyvän koodin kirjoittamista käsin eikä tältä osin helpota ohjelmoijan työtä. Tietokannan hallintaan liittyvien toimintojen lisäksi, Rails-kehys tarjoaa muutamia Express.js-kehystä parempia toiminnallisuuksia sovelluksen kehittäjän näkökulmasta, sillä Rails-kehystä hyväksikäyttävää sovellusta ei tarvitse esimerkiksi uudelleen käynnistää sen koodiin tehtävien muutosten yhteydessä

ja sitä hyödyntävien sovellusten testaaminen on tehty yksinkertaiseksi rakentamalla kehykseen sisään rakennettu tuki useiden eri testityyppien kirjoittamiselle. Web-rajapinnan toteuttajaksi rakennettavaa sovellusta hyödyttää myös Rails-kehyksen ActiveSupport-komponentin tarjoamat työkalut JSON-muotoisen tiedon käsittelyyn.

5.3 Rails-ohjelmistokehyksen käyttö palvelussa

Palvelun taustajärjestelmässä Rails-ohjelmistokehystä hyödynnetään järjestelmän pääomaisen toiminnallisuuden toteuttamiseksi. Apache-palvelinohjelmiston Phusion Passenger-lisäosan päällä ajettava Rails-sovellus on luotu kolmannen osapuolen RailsAPI-pakettia hyväksikäyttäen. RailsAPI-paketti karsii muita, vakiona Rails-sovelluksen mukana tulevia paketteja, jotka ovat pääsääntöisesti rajapinnan tuottamiseen keskittyvien projektien kohdalla turhia. Näitä ovat esimerkiksi Railsin MVC-mallin näkymäkerroksen (view) toteuttavat komponentit sekä HTML-sivustoja käsittelevät paketit. Rajapintaprojektien kannalta turhien pakettien karsiminen on hyödyllistä siksi, että se keventää sovelluksen rakennetta ja näin vähentää sovelluksen tarvetta palvelinresursseille. Turhien pakettien karsiminen ei kuitenkaan aiheuta sovelluksen jatkokehityksen vaikeutumista, sillä poistetuista paketeista on mahdollista valita ja aktivoida uudelleen sovelluksen kannalta tarpeelliset, mikäli sovelluksen tarkoitusta tai rakennetta halutaan jatkossa muuttaa.

Rakenteellisesti Rails-ohjelmistokehystä hyväksikäyttävä, taustajärjestelmän sovellus on yksinkertainen ja sen toiminnallisuus on jaettu yhteen ohjainluokkaan (controller) nimeltä ApiController sekä useaan malliluokkaan (model). ApiController-luokka toimii sovelluksen ytimenä, ja se sisältää tarvittavan koodin kaikkien taustajärjestelmälle saapuvien rajapintakyselyiden palvelemiseksi. Rajapintakyselyiden REST-arkkitehtuurimallin mukaiset URI-osoitteet on sovelluksessa saatu aikaan hyödyntämällä Rails-kehyksen reititysmekanismia (routes), jonka avulla voidaan määrittää, mihin ApiController-luokan metodiin kukin sovellukselle saapuva kysely ohjataan. Rails-kehyksen reititysmekanismin avulla voidaan myös erotella eri HTTP-protokollan metodit niin, että esimerkiksi GET- ja POST-metodeja hyödyntävät kutsut voidaan ohjata eri päätepisteisiin, vaikka ne saapuisivat sovellukselle samaa URI-osoitetta käyttäen. Sovelluksen mallikerroksen luokkia hyödynnetään ApiController-luokan kautta, ja ne tarjoavat kunkin sovelluksen osa-alueen, kuten esimerkiksi käyttäjien ja tapaamisten kannalta oleelliset toiminnallisuudet. Mallikerros toimii myös

sovelluksen kanavana taustajärjestelmän tietokantaratkaisujen kanssa keskustelemiseen ja sisältää siksi kaiken tietokantakyselyihin liittyvän logiikan.

Sovelluksen hyödyntämän RailsAPI-paketin vuoksi MVC-mallin mukainen, Rails-kehyksen ActionView-komponenttiin perustuva näkymäkerros on poistettu sovelluksesta kokonaan. Näkymäkerroksen sijaan tiedon välittäminen sovelluksesta ulospäin on toteutettu suoraan ohjainkerroksesta käyttämällä Rails-kehyksen ActionController-luokan tarjoamia metodeja. Palvelun Android-sovelluksen odottama tieto tulee olla JSON-muotoon pakattua, jonka vuoksi Rails-sovelluksen ohjainkerroksesta ulospäin virtaava tieto tulee pakata sopivaan muotoon ennen sen lähettämistä. Vastausten pakkaaminen JSON-muotoon hoituu Rails-kehyksessä automaattisesti, mikäli ActionController-luokan metodeille annetaan sopivat parametrit.

5.4 Redis-palvelimen käyttö

Palvelun käyttäjän istuntoavain on taustajärjestelmän Rails-sovelluksessa jatkuvassa käytössä, jonka vuoksi avainten tallennuspaikaksi on taustajärjestelmässä varattu Redis avain-arvopalvelin. Palvelun alustavassa versiossa Redis-palvelinohjelmistoa ajetaan samalla palvelinkoneella kuin Rails-sovellustakin, mutta tarpeen tullen se olisi mahdollista siirtää ulkoiselle palvelimelle ilman suurta työmäärää. Rails-kehys ei vakiona sisällä tukea Redis-palvelimen käytölle, jonka vuoksi projektissa on otettu käyttöön kolmannen osapuolen RedisRB-paketti. Kyseinen paketti on sisällytetty sovelluksen mallitason SessionToken-luokkaan, joka sisältää tarvittavat metodit käyttäjien istuntojen hallintaa varten.

Redis-palvelinohjelmisto sisältää sisäänrakennetun tuen tallennettujen avain-arvoparien elinajan määrittämiseksi. Tämän ominaisuuden avulla käyttäjille luodut istuntoavaimet saadaan automaattisesti vanhenemaan halutun ajan kuluessa käyttäjän sisään kirjautumisesta. Teknisesti istuntoavainten uusiminen on toteutettu niin, että avaimen vanhentumisen jälkeen se poistuu Redis-palvelimelta, jolloin sovellus ei löydä vanhaa avainta. Mikäli vanhaa avainta ei löydetä, Rails-sovellus pyytää käyttäjän Android-sovellukseen tallennettua salasanatiivistettä, jonka varmentamisen jälkeen käyttäjälle luodaan uusi istuntoavain. Uuden istuntoavaimen luonnin jälkeen käyttäjän suorittamat toimenpiteet onnistuvat niin kauan kunnes Redis-palvelimelle tallennettu avain taas vanhenee.

Rails-sovelluksen rakennusvaiheessa käyttäjien istuntoavainten tallennus oli toteutettu MariaDB-tietokantaa hyväksikäyttäen ja avainten hallinta oli käsin ohjelmoitu Rails-sovelluksen mallikerrokseen. Istuntoavainten vaihtaminen Redis-palvelinta hyväksikäyttävään ratkaisuun nopeutti ja yksinkertaisti istuntojen hallintaa. Tämä myös mahdollisti Rails-sovelluksen SessionToken-luokan uudelleenkirjoittamisen muotoon, jonka rivimääräinen koko pieneni yhteen viidesosaan MariaDB-tietokantaa hyödyntävästä versiosta.

5.5 MariaDB-tietokantapalvelimen käyttö

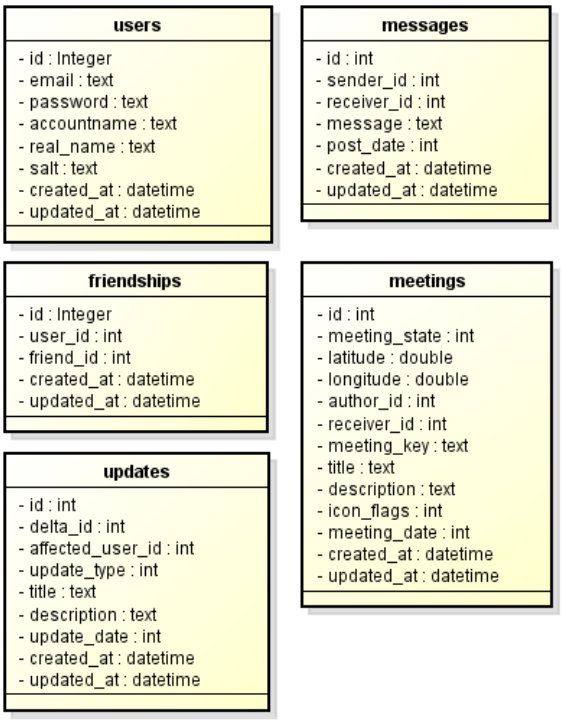
Rails-kehystä käyttävää projektia luodessa on mahdollista ottaa käyttöön useita eri tietokantavaihtoehtoja, joista oletusvaihtoehtona Rails tarjoaa kehityskäyttöön soveltuvaa SQLite-tietokantaa. Tuki eri tietokantavaihtoehtojen käyttöönotolle on toteutettu Rails-kehyksen tukemien pakettien avulla. Itse Rails-kehyksen mukana tulevat paketit mahdollistavat tuen muutamille yleisimmiten käytetyille tietokantajärjestelmille, kuten MySQL ja PostgreSQL. Rails-projektin käyttämän tietokannan konfiguraatio on mahdollista määrittää erikseen tuotanto-, kehitys- sekä testiympäristöille, mikä mahdollistaa eri tietokantojen käytön eri ajoympäristöjen kanssa. Tämä Rails-kehyksen ominaisuus helpottaa sovelluksen kehittämistä, sillä tuotantoympäristössä ajettavan sovelluksen tietokantaan tallennetut tiedot eivät häiritse muita ajoympäristöjä.

Palvelun pääasiallinen tiedontalletus on toteutettu avoimen lähdekoodin MariaDB-tietokantapalvelinta käyttäen. MariaDB-palvelimen olemuksen vuoksi se on täysin yhteensopiva MySQL-tietokantajärjestelmän kanssa, mikä sovelluksen kehityksen kohdalla mahdollistaa Rails-kehyksen mukana tulevan MySQL-ajurin käytön MariaDB-palvelimeen yhdistämiseksi. Rails-kehyksen tietokantatoiminnallisuudesta huolehtiva ActionRecord-komponentti piilottaa kaiken tietokantakohtaisen toiminnallisuuden. Tällöin sovelluksen kehittäjän ei tarvitse tehdä sovelluksen koodiin muutoksia, vaikka alla oleva tietokantapalvelin vaihdettaisiin esimerkiksi MariaDB-palvelimesta PostgreSQL-palvelimeen. Rakenteellisesti sovelluksen tietokantaan liittyvä koodi sijaitsee Rails-projektin mallikerroksen luokissa, ja se hyödyntää ActiveRecord-komponentin metodeja tietokantakyselyiden rakentamiseen SQL-lauseiden sijasta.

MariaDB-tietokantapalvelin on palvelun alustavassa versiossa sijoitettu samalle palvelimelle muiden taustajärjestelmän komponenttien kanssa. Tämä tarkoittaa sitä, ettei Rails-sovelluksen tarvitse yhdistää tietokantapalvelimeen internet-yhteyden välityksellä, jolloin yhteys tietokannan ja sovelluksen välillä on lähes välitön. Palvelun jatkokehityksen kannalta MariaDB-palvelimen käyttö on myös hyödyllistä, sillä palvelinohjelmistosta on saatavilla myös hajautettujen tietokantajärjestelmien rakentamista varten suunniteltu Galera Cluster versio. MariaDB Galera Cluster-ohjelmiston vaihtaminen palvelun yhden paikallisen tietokantapalvelimen tilalle on MariaDB-dokumentaation mukaan suhteellisen yksinkertainen toimenpide ja tulevaisuutta ajatellen myös välttämätön, mikäli palvelun suosio kasvaisi sen julkaisun jälkeen. [Getting started with MariaDB Galera Cluster 2014]

5.6 Taustajärjestelmän tietokantarakenne

Rails-kehystä hyödyntäviä sovelluksia luotaessa mallikerroksen luokat voidaan luoda joko käsin tai Rails-kehyksen mukana tulevia työkaluja hyödyntäen. Sovelluksen mallikerroksen yhdistäminen tietokantaan vaatii Rails-kehyksen ActiveRecord-komponentin hyödyntämistä. Käytännössä tämä on toteutettu niin, että kunkin tietokantataulun suunnittelun jälkeen mallille luodaan migraatioluokka, jossa määritellään tietokantatauluun halutut kentät. Itse tietokantataulujen luonti suoritetaan komentorivin kautta käyttäen Ruby-kielellä luotua Rake-työkalua. Palvelun kannalta Rails-kehyksen tapa tietokantataulujen luontiin ja hallintaan on hyvin sopiva, sillä se mahdollistaa tulevaisuudessa tehtävien muutosten päivittämisen tietokantaan erittäin helposti.



Kuva 8. Palvelun taustajärjestelmän tietokantataulut.

Palvelun alustava versio on toiminnallisuutensa kannalta niin rajoittunut, että sen taustajärjestelmän vaatima tietokantatoteutus on voitu pitää yksinkertaisena. Kuvassa 8 näkyvä toteutus sisältää kaikki palvelun alustavan version vaatimat tietokantataulut, lukuun ottamatta käyttäjien istuntoavainten tallennukseen luotua taulua, sillä kyseinen toiminnallisuus saatiin siirrettyä käyttämään Redis-järjestelmää. Tietokantatoteutuksen kaikkia tauluja yhdistävänä tekijänä on Rails-kehyksen automaattisesti tarjoama, kahdesta kentästä, created_at ja updated_at, koostuva aikaleima, jonka avulla pidetään kirjaa tietokantamerkintöjen luonnista sekä päivityksistä. Muutamien taulujen kohdalla, kyseisten aikaleimojen lisäksi tauluun on tallennettu merkinnän luontihetken kertova aika. Tieto on siis tallennettu kahteen kertaan, mutta erimuotoisena. Syy tähän on yhdenmukaisuuden hakeminen palvelun Android-sovelluksen ja taustajärjestelmän välillä sekä tietokantaan tallennetun tiedon jälkikäsittelyn vähentäminen, koska aikaleimaa ei tarvitse muuntaa erikseen Android-sovellukselle sopivaan muotoon.

Taulujen väliset, relaatiotietokannoissa yleisesti käytössä olevat suhteet ovat Rails-kehyksen käytön vuoksi sisällytetty mallitason luokkiin, joissa ne on määritelty ActiveRecord-komponentin tarjoaman toiminnallisuuden avulla. Suhteiden määrittely sovelluksen mallitasolla mahdollistaa niiden osittaisen piilottamisen ohjainkerroksen koodilta, joka tekee niiden käytöstä vaivatonta ja selkeää.

6 Palvelun testaus

6.1 Mobiilipalvelun testaus

Laajojen ohjelmistoprojektien hallinta ilman loogista testausjärjestelmää on usein lähes mahdotonta, varsinkin mikäli projektin parissa työskentelee useita kehittäjiä. Kehittäjien määrästä huolimatta useista komponenteista koostuvien projektien testaus on lähes poikkeuksetta hyödyllistä ja jatkokehityksen kannalta käytännössä elintärkeää. Projektien laajuus sekä jakautuminen useisiin eri komponentteihin tekee järkevän testijärjestelmän rakentamisesta haastavaa, sillä jokaiselle projektin komponentille täytyy rakentaa omanlaisena testit.

Sopivien testimenetelmien valinta kullekin projektin komponentille on riippuvaista komponentin tyypistä ja käytettyjen ohjelmistokehysten tarjoamista työkaluista. Projektissa rakennetun palvelun testaaminen vaatii sopivien testimenetelmien löytämisen jokaiselle palvelun kolmesta pääkomponentista, joita ovat itse Android-sovellus, web-rajapinta sekä palvelun taustajärjestelmä. Testimenetelmien rakentaminen lähdettiin toteuttamaan palvelun Android-sovelluksesta, sillä se sisältää projektin komponenteista ainoan loppukäyttäjälle näkyvän käyttöliittymän. Käyttöliittymien testaaminen on usein haasteellista, sillä niiden ulkoasu on usein muuttuvassa tilassa ja käyttäjä voi liikkua eri näkymien välillä usein eri tavoin. Android-alustan tapauksessa käyttöliittymien testaamiseen on kuitenkin saatavilla työkaluja, jotka helpottavat prosessia huomattavasti.

Palvelun web-rajapinta on projektin kolmesta pääkomponentista ainoa, jonka testaamiseen ei ole valmista toteutusta, mutta yksinkertaisuutensa vuoksi tämä ei aiheuta ongelmaa. Palvelun kuten muidenkin HTTP-protokollaa käyttävien web-rajapintojen tapauksessa testaamiseen on mahdollista rakentaa yksinkertaisia työkaluja tai vaihtoehtoisesti käyttää internetin tarjoamia palveluja. Web-rajapinnasta poiketen Rails-kehystä hyödyntävän taustajärjestelmän testaamiseen on Android-sovelluksen tapaan tarjolla valmiita työkaluja, joiden käyttö on vielä Android-alustankin tarjoamia työkaluja helpompaa.

6.2 Mobiilisovelluksen testausmenetelmät

Palvelun mobiilisovelluksen testausta lähdettiin lähestymään miettimällä, mitä sovelluksesta halutaan testata ja kuinka testaus suoritetaan. Mobiililaitteella ajettavan sovelluksen testauksella voidaan haluta selvittää useita eri asioita, kuten sovelluksen aiheuttama rasite laitteen akulle, erityyppisten laitteiden käyttö sovelluksen alustana tai eri resurssien, kuten internet-yhteyden saatavuuden vaikutus sovelluksen toimintaan. Sosiaalisen median palvelun näkökulmasta tärkeimmät testikohteet ovat sovelluksen kyky suoriutua tilanteista, joissa se uhkaa kaatua sekä käyttöliittymän toimivuus loppukäyttäjän näkökulmasta. Molempien tekijöiden testaamiseen on tarjolla laajasti käytössä olevia työkaluja ja runsaasti ohjeistusta niiden käyttöön.

Sovelluksen kaatumisherkkyyttä voidaan testata usein eri menetelmin, joista kuitenkin eräs yleisesti käytössä oleva on yksikkötestaaminen. Yksikkötestaaminen on sovelluksen jatkokehityksen kannalta myös hyvä tapa havaita uutta toiminnallisuutta lisätessä syntyneitä virheitä, jotka saattavat aiheuttaa ongelmia sovelluksen toiminnallisuudessa. Android-alusta tarjoaa sovellusten yksikkötestaamiseen JUnit-ohjelmistokehyksestä johdetun AndroidTestCase-laajennuksen. Tämän lisäksi sovelluksessa käytettyjen Android-alustasta riippumattomien komponenttien testaaminen voidaan hoitaa pelkkää JUnit-kehystä käyttäen. Sovelluksen rakenteen vuoksi Android-alustasta riippumattomia luokkia on vain yksi, jonka yksinkertaisuuden vuoksi testaaminen pelkkää JUnit-kehystä käyttäen unohdettiin. Tämän sijaan AndroidTestCase-laajennuksella testattavia luokkia on huomattava määrä ja niiden testaaminen katsottiin hyödylliseksi. [Testing Fundamentals 2014]

Sovelluksen käyttöliittymätestauksen oleellisuus asetti sopivan testaustyökalun valinnalle vaatimuksia, joiden täyttämiseksi päädyttiin valitsemaan kolmannen osapuolen Robotium-ohjelmistokehys. Robotium on Android-sovellusten testaamiseen luotu työkalu, jonka käyttö ei aseta vaatimusta sovelluksen lähdekoodin saatavuudelle. Robotiumin avulla voidaan siis suorittaa niin sanottua mustan laatikon testaamista, jolloin testattavan sovelluksen koodi ei ole saatavilla. Tämä on mahdollista siksi, että Robotiumin testausmenetelmänä on loppukäyttäjän toimintojen simulointi. Toisin sanoen Robotium simuloi sovelluksen käyttöliittymässä navigointia käyttäjän tavoin, mikä mahdollistaa loppukäyttäjällä testaamisen automatisoinnin. Käyttöliittymän testaamisen lisäksi Robotium tarjoaa Android-alustan yksikkötestaamiseen vaadittavat työkalut, joiden avulla sovelluksen sisäisten toimintojen testaus on suoritettu. Ilman

vastaavan työkalun käyttöä, sovelluksen testaamisesta olisi tullut huomattavasti hankalampaa. [Robotium 2014]

6.3 Web-rajapinnan testausmenetelmät

Palvelun kolmesta pääkomponentista, web-rajapinnan testaaminen on ainoa testitapaus, johon testaustyökalujen rakentaminen itse on järkevää. Tämä johtuu siitä, että testitapaukset ovat rajapinnan kohdalla hyvin yksinkertaisia ja niiden lopullinen toimivuus tullaan testaamaan joka tapauksessa Android-sovelluksen yhteydessä. Käytännössä testausta voidaan suorittaa joko käyttämällä internetistä löytyviä palveluja, joiden avulla voidaan luoda taustajärjestelmälle huomattavaa rasitetta, tai kirjoittamalla itse palvelimella ajettavia testiohjelmia. Internetistä löytyvien palveluiden käyttö rajapinnan testaamiseksi on toimiva ratkaisu useissa tapauksissa, mutta rakennetun palvelun kohdalla niiden käyttöön ei ollut riittävää tarvetta. Syynä tähän on palvelun alustavan version koko ja sen taustajärjestelmän komponenttien sijoittuminen yhdelle palvelimelle.

Palvelun kannalta käytännöllisemmäksi testausmenetelmäksi havaittiin yksinkertaisten testiohjelmien rakentaminen, sillä niiden avulla saadaan nopeasti selville kaikkien palvelun taustajärjestelmän tarjoamien toimintojen toimivuus. Käytännössä rajapinnan testaamiseen rakennettuna testimenetelmänä oli Linux-käyttöjärjestelmän komentotulkissa ajettava ohjelma, joka hyödyntää cURL-nimistä ohjelmistoa rajapintayhteyksien luomiseen. Luodun ohjelman avulla saadaan nopeasti selville palvelun taustajärjestelmän toimivuus miltä tahansa Linux-käyttöjärjestelmää käyttävältä koneelta.

6.4 Taustajärjestelmän testausmenetelmät

Palvelun taustajärjestelmän rakentuminen useista eri komponenteista, kuten tietokantapalvelimesta ja Rails-sovelluksesta, tarkoittaa testauksen suhteen sitä, että käytettävät testimenetelmät tulee keskittää kohtaan, josta kaikkien komponenttien toimivuus voidaan testata kerralla. Taustajärjestelmässä kyseinen kohta on palvelun toiminnallisuudesta vastaava Rails-sovellus. Tämä johtuu siitä, että muut taustajärjestelmän komponentit toimivat vain tietovarastoina Rails-sovellukselle, eikä niiden erillinen testaus olisi hyödyllistä.

Taustajärjestelmän Rails-sovelluksen testaus on toteutettu käyttäen Rails-ohjelmistokehykseen sisäänrakennettua testausjärjestelmää. Rails-kehys mahdollistaa kolmen erityyppisten testien rakentamisen, joista palvelun kannalta hyödyllisimpiä ovat yksikkötestit, sekä toiminnallisuustestit. Rails-kehyksen työkaluja käytettäessä, uuden malli- tai ohjainkerroksen luokan luonnin yhteydessä luodaan automaattisesti tarvittavat testikehykset niiden testaamiseksi. Testien suorittaminen kuitenkin vaatii käsiteltävää tietoa, jota Rails-kehyksen testiympäristöllä ei ole vakiona saatavilla, sillä testiympäristö käyttää eri tietokantaa kuin tuotanto ja kehitysympäristöt. Tämä on Rails-kehyksessä ratkaistu luomalla jokaiselle testattavalle mallille oma YAML-merkitänkielinen tiedosto, johon kunkin mallin tietokantatauluun sijoitettavat tiedot voidaan kirjoittaa. Palvelun testaamisen tapauksessa mallikerroksen luokkia vastaaviin YAML-tiedostoihin on listattu muutamia keksittyjä merkintöjä, kuten käyttäjätunnuksia ja viestejä, joita Rails-kehys käyttää testien ajamiseen. Rails-kehyksellä luotujen testien ajaminen tapahtuu Rake-työkalua käyttäen. Rake-työkalu mahdollistaa ajettavien testien valinnan niin kaikkien sovelluksen testien ajamisesta yksittäisen testimetodin ajamiseen. [Guide to testing rails applications 2014]

Varsinainen testikoodi on palvelussa jaettu mallikerroksen kohdalla yksikkötesteihin ja ohjainkerroksen kohdalla toiminnallisuustesteihin. Mallikerroksen testeissä varmistetaan jokaisen mallin sisältämän metodin toiminta sekä varmistetaan, että kultakin metodilta on mahdollista saada oikeantyyppinen virheilmoitus, mikäli niitä yritetään käyttää väärin. Mallikerroksen testaamisella saadaan varmistettua, että sovelluksen toiminta tietokannan kanssa on halutun tyyppistä. Laajemman toiminnallisuuden testaamiseksi sovellukseen on rakennettu toiminnallisuustestejä, joiden avulla saadaan varmistettua sovelluksen ohjainkerroksen sekä web-rajapinnan toiminta paikallisesti. Toiminnallisuustestit tarjoavat myös mahdollisuuden ohjainkerroksen JSON-tyyppisten vastausten testaamiseen, sillä ne mahdollistavat paikallisten HTTP-kutsujen suorittamisen sovelluksella. [Guide to testing rails applications 2014]

6.5 Taustajärjestelmän testaustuloksia

Eräs palvelun käyttökokemukseen eniten vaikuttavimmista asioista heti Android-sovelluksen käyttöliittymän jälkeen on palvelun taustajärjestelmän toiminnallisuus ja nopeus. Taustajärjestelmän nopea toiminta on myös tekijä, jolla on suora vaikutus

kaikkiin palvelun käyttäjiin. Tämän vuoksi taustajärjestelmän nopeuteen on kiinnitetty erityistä huomiota palvelun testaamisessa.

Taustajärjestelmää testattaessa ymmärrettävän tiedon saanti oli välttämätöntä, sillä palvelun komponenttien kehittäminen nopeammiksi ja paremmiksi ei olisi ollut muuten mahdollista. Tavallisesti testaamisessa käytetyt menetelmät tuottavat kaikki numeropohjaista tietoa testien suorituksesta, mutta suurin osa tästä tiedosta ei ole erityisen hyödyllistä esimerkiksi palvelun suorituskyvyn parantamisen kannalta. Tämän vuoksi palvelun kehittämisen osana rakennettiin eri komponenttien suorituskykyä mittaavia testejä, joiden avulla voitiin mitata eri komponentin toimintoihin kuluvia aikoja. Käytännössä nämä testit olivat lyhyitä koodinpätkiä, joiden avulla mitattiin esimerkiksi istuntoavaimen varmennukseen kuluvaa aikaa. Esimerkkinä näistä testeistä ovat taulukossa 2 esitetyt testitulokset. Tulokset osoittavat suorituskyvyn paranemisen, joka saavutettiin, kun käyttäjän istuntoavainten säilytyspaikka vaihdettiin palvelun taustajärjestelmässä käytetystä MariaDB-tietokannasta Redis-palvelimeen.

Taulukko 2. Redis-palvelimen käyttö verrattuna MariaDB-tietokantaan.

Redis: **redis_is_session_valid(token)**	**MariaDB:** **mariadb_is_session_valid(token)**
1.071912 ms	2.442313 ms
1.215142 ms	2.168154 ms
0.774434 ms	1.713389 ms
0.619739 ms	1.984619 ms
Keskiarvo: 0.92030675 ms	**Keskiarvo:** 2.07711875 ms

Taulukon 2 osoittamat tulokset saatiin kirjaamalla ylös eri tietokantatoteutusta käyttävien metodejen suoritukseen kuluvia aikoja. Testissä saadut ajat on kopioitu palvelun taustajärjestelmän Rails-sovelluksen lokitiedostoihin tehdyistä tulosteista, ja niille on laskettu keskiarvot. Näiden aikojen perusteella voidaan havaita, että Redis-palvelimen käyttöönotosta oli selvää hyötyä. Testin tuloksia tarkasteltaessa täytyy kuitenkin ottaa huomioon, että testin suorittamisen aikaan palvelussa liikkuvan tiedon ja käyttäjien määrä oli lähes olematon ja tuotantokäyttöön viedyssä palvelussa kyseiset ajat olisivat huomattavasti suuremmat. Testin perusteella määritelty 55 %:n

suorituskyvyn parannus olisi tuotantokäytössä olevassa palvelussa huomattavasti vaikuttavampi tekijä.

7 Yhteenveto

Tämän projektin tarkoituksena oli selvittää, kuinka sosiaalisen median mobiilipalveluja rakennetaan ja kuinka olemassa olevien palvelujen sisältämiä ongelmia voitaisiin ratkaista uudentyyppisellä lähestymistavalla. Työn edistyessä havaittiin, että alkuperäinen arvio kokonaisen mobiilipalvelun rakentamiseen vaadittavan työmäärän suhteen ei pitänyt paikkaansa, vaan palvelun rakentamiseen vaadittava työmäärä olikin moninkertainen. Tämä virhearvio aiheutti projektin kohdalla sen, että rakennetusta palvelusta jouduttiin karsimaan alkuperäisen suunnitelman mukaisia ominaisuuksia. Projektin alussa tehty arvio tarvittavasta ajasta oli myös virheellinen ja lopullinen projektiin käytetty aika ylitti alkuperäisen kahden kuukauden arvion yli kuukaudella. Yhteensä projektiin käytettiin aikaa yli kolme kuukautta. Projektin haasteista huolimatta sen rakennusprosessi oli opettavainen kokemus, joka johti jatkokehityksen kannalta houkuttelevaan lopputulokseen. Projektin aikana tutkimustyötä tehdessä vastaan sattui myös muutamia, vastaavantyyppisiä palveluja, joista osa on saavuttanut huomattavan suuren käyttäjäkunnan. Tämän perusteella voidaankin päätellä, että projektissa rakennetun palvelun idea on selvästi toimiva ja sen jatkokehitys on kannattavaa. Ilman tätä projektia tieto palvelun idean toimivuudesta sekä taidot useiden eri käytettyjen teknologioiden käytöstä olisivat jääneet saamatta.

Yksi projektin tavoitteista oli selvittää palvelun idean toimivuus sekä kokeilla, voidaanko käyttäjien kontaktilistojen paisumista estää katkaisemalla käyttäjien välinen suhde, mikäli käyttäjät eivät ole enää yhteydessä toisiinsa. Tämän ajatuksen testaaminen kuitenkin todettiin epäkäytännölliseksi, sillä sen toimiva testaus vaatisi palvelulle huomattavan määrän käyttäjiä, joita projektin luonteen ja palvelun keskeneräisyyden vuoksi ei ollut mahdollista saada. Ongelman ratkaisun testaamiseen voidaan kuitenkin palata jatkossa, mikäli palvelun jatkokehys johtaa palvelun käyttäjäkunnan kasvuun.

Työssä rakennetun palvelun tekeminen oli opettavainen kokemus, jonka ansiosta pääsin tutustumaan moniin mobiili- ja web-palvelujen rakennuksessa käytettyihin teknologioihin. Palvelua rakentaessa oppimiani uusia asioita olivat esimerkiksi Redis-palvelinohjelmiston käyttö Rails-ohjelmistokehyksellä luodun web-sovelluksen tukena.

Redis-palvelinohjelmistoa vastaavien ohjelmistojen käytöstä minulla ei ollut aikaisempaa kokemusta, mutta tiesin, että kyseisten ohjelmistojen käyttö on yleistä maailmalla suosittujen sosiaalisen median palvelujen rakentamisessa. Palvelun taustajärjestelmää luodessani opin myös uusia tekniikoita Rails-ohjelmistokehyksen hyödyntämiseksi, niistä merkittävimpiä ovat Rails-kehyksen muokkaaminen puhtaasti rajapintakäyttöä varten sekä Rails-kehyksen testiympäristön YAML-tiedostojen luonti ja niihin tallennettujen tietojen käyttö sovelluksen yksikkötestaamisessa. Palvelun Android-sovelluksen rakentamisesta opin Android-sovellusten testaamisessa käytettävien työkalujen käytön, josta minulla ei ollut aikaisempaa kokemusta.

Palvelun jatkokehittäminen nykyisestä tilastaan tulisi aloittaa alkuperäisen suunnitelman mukaisten toiminnallisuuksien lisäämisestä, jonka jälkeen palveluun voitaisiin alkaa lisäämään muita ominaisuuksia, kuten tuki kuvaviestien lähettämiselle. Jatkokehitys olisi kannattavaa myös siksi, että se voisi mahdollistaa tarkempien vastausten saannin projektissa esitettyihin kysymyksiin. Palvelun rakentamisesta opitut taidot ja idean toimivuuden selvittäminen tekivät projektista onnistuneen kokonaisuuden.

Lähteet

Smith, Cooper. 2014. Twitter Generates 70% Of Its Revenue From Mobile. Verkkodokumentti. Business Insider. <http://www.businessinsider.com/twitter-generates-70-of-its-revenue-from-mobile-2013-10>. Luettu 4.2.2014.

Facebook Reports Fourth Quarter and Full Year 2013 Results. 2014. Verkkodokumentti. Facebook, Inc. <http://investor.fb.com/results.cfm>. Luettu 4.2.2014.

Desktop & mobile comparison, WWW traffic monthly. 2014. Verkkodokumentti. StatCounter Global Stats. <http://gs.statcounter.com/#desktop+mobile-comparison-ww-monthly-201301-201401>. Luettu 4.2.2014.

Blodget, Henry. 2013. Apple Is Being Shortsighted -- And This Could Clobber The Company. Verkkodokumentti & kuva. Business Insider. <http://www.businessinsider.com/apple-is-being-greedy-2013-9>. Luettu 8.2.2014.

Dashboards - Platform versions. 2014. Verkkodokumentti & kuva. Android Developer. <http://developer.android.com/about/dashboards/index.html>. Luettu 12.2.2014.

Wester, Greg. 2013. Trust Without Passwords. Verkkodokumentti. Medium. <https://medium.com/p/c91146a96cef>. Luettu 16.3.2014.

Wilson, Jesse. 2011. Android's HTTP Clients. Verkkodokumentti. Blogspot. <http://android-developers.blogspot.fi/2011/09/androids-http-clients.html>. Luettu 20.3.2014.

Hollister, Sean. 2014. Starbucks admits its iPhone app stores unencrypted user passwords. Verkkodokumentti. The Verge. <http://www.theverge.com/2014/1/15/5313648/starbucks-admits-ios-app-stored-passwords-in-plain-text>. Luettu 23.3.2014.

Dropbox Core API Documentation. 2014. Verkkodokumentti. Dropbox. <https://www.dropbox.com/developers/core/docs#delta>. Luettu 23.3.2014.

Cailliau, Robert. 2011. A Little History of the World Wide Web. Verkkodokumentti. World Wide Web Consortium (W3C). <http://www.w3.org/History.html>. Luettu 26.3.2014.

Using Telegram API. 2014. Verkkodokumentti. Telegram Messenger. <https://core.telegram.org/api>. Luettu 26.3.2014.

REST API v1.1 Resources. 2014. Verkkodokumentti. Twitter, Inc. <https://dev.twitter.com/docs/api/1.1>. Luettu 26.3.2014.

Welcome to Rails. 2014. Verkkodokumentti. GitHub. <https://github.com/rails/rails>. Luettu 1.4.2014.

Ruby on Rails History — A Look Back. 2011. Verkkodokumentti. Gunner Technology. <http://www.gunnertech.com/2011/11/ruby-on-rails-history-a-look-back/>. Luettu 1.4.2014.

Ruby on Rails. 2014. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Ruby_on_Rails>. Luettu 1.4.2014.

A Guide to Testing Rails Applications. 2014. Verkkodokumentti. RailsGuides. <http://guides.rubyonrails.org/testing.html>. Luettu 9.4.2014.

Android robotium google code. 2014. Verkkodokumentti. Robotium. <https://code.google.com/p/robotium/>. Luettu 9.4.2014.

Testing Fundamentals. 2014. Verkkodokumentti. Android Developer. <https://developer.android.com/tools/testing/testing_android.html>. Luettu 9.4.2014.

Getting Started with MariaDB Galera Cluster. 2014. Verkkodokumentti. MariaDB. <https://mariadb.com/kb/en/getting-started-with-mariadb-galera-cluster/>. Luettu 9.4.2014.

Baker, Natasha. 2013. When it comes to apps, consumers have shorter attention spans. Verkkodokumentti. Reuters. <http://www.reuters.com/article/2013/08/20/apps-attention-span-idINDEE97J0DU20130820>. Luettu 9.4.2014