

Johdanto

Erityisesti mobiilialustoiden myötä (mobiili)applikaatiot ovat nousseet keskeiseksi sovellusten käyttöalueeksi. Kuitenkin, applikaatioiden käytöstä tiedetään varsin vähän. [Böhmer et al. \(2011\)](#) tutkimus on ensimmäinen näkemäni työ, jossa käytetään application sensor-termiä kuvaamaan tällaista toimintaa. Tässä työssä toteutetaan eräs implementaatio application sensorista.

Ohjelma toteutetaan Ruby on Rails (RoR) ympäristöllä, versiolla *3.1.0.rc4*. On syytä huomauttaa, että toteutuksen aikana versio *3.1.0* tuli yleiseen jakeluun, eli edessä olisi päivitystyö tämänkin sovelluksen osalta. Päivityksen ei tulisi kuitenkaan olla merkittävä.

Tehtävänanto

Ohjelmisto tarjoaa yksinkertaisen järjestelmän levittää, ylläpitää ja seurata applikaatioiden käyttöä. Kyseessä on siis yksinkertainen App Store-tyylinen sovellus, joka on esisijaisesti suunniteltu mobiiliapplikaatioiden käyttöön.

Levittämisellä tarkoitetaan mahdollisuutta ladata applikaatioita verkkosivuiden kautta. Tähän ohjelmistoon ei ole tarkoitus liittää mahdollisuutta maksullisiin toimintoihin, mutta sen sijaan applikaation arvostelu ja kommentointi ovat mielekkäitä lisäattribuutteja, joiden lisääminen on korkealla prioriteetillä sovelluksen kehittämisessä.

Applikaation **ylläpitäminen** kuvaa uusien applikaatioversioiden välittämistä kuluttajille. Käytännössä ohjelmisto ei toteuta ylläpitoprosessia, vaan tarjoaa rajapinnan, jonka kautta applikaatio voi käynnistyksen yhteydessä tarkistaa, onko se uusimmassa mahdollisessa versiossa. Applikaatiopään toteutus ei kuulu tämän työn määrittelyyn, vaan ohjelmistossa keskitytään vain yksinkertaisen infrastruktuurin luomiseen tätä tarkoitusta varten.

Applikaation **käytön seuranta** perustuu samoin kuin ylläpitäminen, applikaatiossa toteutettavaan ohjelmistologiikkaan. Mobiililaite on tunnistettavissa yksilöllisen IMEI-koodinsa perusteella, eli jokainen yksittäinen käyttäjä on havaittavissa¹. Tällöin tarjotaan yleinen rajapinta applikaatioille, jotta ne voivat raportoida käyttäjän toimintoja ylläpitäjille. Luonnollisesti toteutukseen kuuluu käyttöliittymä, jonka kautta ylläpitäjä voi visualisoida ja havainnoida applikaation käyttöä.

Ensisijaisesti kuluttaja siis käyttää ohjelmistoa applikaation kautta. Tämä valinta pyrkii turvaamaan käyttäjälle positiivisen ja saumattoman käyttökokemuksen, koska kuluttajan ei tarvitse poistua applikaatiosta arvostelun jättämiseksi tai päivityksen saamiseksi. Kuitenkin, harjoituksen vuoksi ohjelmistoon toteutetaan näkymät ohjelmistojen katsomiseksi, kommenttien näkemistä varten sekä niiden jättämistä varten.

¹ Toteutuksessa ei nyt oteta kantaa tällaisen toiminnallisuuden lainmukaisuuteen tai eettisyyteen.

Kuitenkin, ohjelmiston pääkäyttäjryhmä on applikaatioiden kehittäjät, jotka haluavat ylläpitää parempaa suhdetta kuluttajien ja kehittäjien välillä. Tämä on mahdollista päivitysmekanismin kautta sekä analysoimalla miten applikaatiota käytetään.

Sidosryhmät

Kehittäjä on applikaation yksittäinen kehittäjä. Hänelle keskeistä on pystyä lataamaan ja päivittämään applikaatiota sekä seuraamaan applikaation käyttöä. Tämän takia hänet tulee voida autentikoida yksikäsitteisesti käyttäjätunnuksella ja salasanalla, joka mahdollistaa vain hänen päästä käsiksi oman applikaationsa tietoihin. Lisäksi jokainen kehittäjä voi olla myös normaali käyttäjä, eli kuluttaja.

Kuluttaja voi selata applikaatioita sekä ladata ja kommentoida niitä.

Ylläpito on tunnistettu käyttäjäryhmäksi, mutta ei sitä ei ole vielä toteutettuna tähän palveluun. Ylläpitäjät ovat kehittäjiä, joilla on erityisoikeus muokata myös muiden applikaatioita.

Käyttötapauskuvaukset

Käyttötapaus 1: Kuluttaja lataa applikaation

Kuluttaja löytää kiinnostava applikaation ja lataa sen painamalla "Download" linkkiä.

Käyttötapaus 2: Kuluttaja kommentoi applikaatiota applikaation kautta

Applikaatiossa on sisäänrakennettuna toiminto, joka mahdollistaa kommentin kirjoittamisen. Käyttäjä käynnistää tämän toiminnon ja kirjoittaa kommenttinsa. Kommentti lähetetään applikaatiosta oikeanlaisena HTTP-viestinä ohjelmistolle, joka tallentaa sen.

Käyttötapaus 3: Kuluttaja kommentoi applikaatiota ohjelmiston kautta

Kuluttaja etsii applikaation, kirjoittaa kommentin applikaation omalla sivulla ja klikkaa "Add comment" nappainta.

Käyttötapaus 4: Kehittäjä päivittää ohjelmistoa

Kehittäjä kirjautuu sisään ja näkee kaikki applikaationsa. Hän etsii oikean ohjelman ja avaa ohjelman sivun. Kehittäjä siirtyy ohjelman muokkaustilaan edit-nappulan kautta ja lataa uuden tiedoston osaksi ohjelmaa. Lisäksi hän päivittää ohjelman versionumeroa ja tallentaa muutokset.

Käyttötapaus 5: Kehittäjä seuraa ohjelman käyttöä

Applikaatiosta on kerätty keskeisiä tietoja, kuten käyttömäärää ja keskimääräistä käyttöaikaa. Kehittäjä kirjautuu portaaliin sisään, avaa applikaation sivun ja näkee nämä tiedot statistiikkasivun osana.

Keskeiset mallit

Users

Käyttäjät ovat tässä ohjelmistossa applikaatioiden ylläpitäjiä, eivät varsinaisia kuluttajia. Mallin keskeinen rooli on siis suorittaa autentikaatio, jolloin vain applikaatioiden ylläpitäjät voivat muokata omia ohjelmiaan.

Tämän takia malli sisältää perinteiset yhteystiedot (käyttäjätunnus, nimi, sähköpostiosoite, salasana) sekä viitteitä (has-many -relaatio) applikaatioihin.

Applications

Applikaatiot ovat ohjelmia, joita käyttäjät lataavat omiin laitteisiinsa. Tässä toteutuksessa erityisesti keskitytään mobiiliapplikaatioiden lataamisen tarkoituksiin.

Kullakin applikaatiolla on nimi, kuvaus, versionumero, kuva sekä tiedosto. Lisäksi applikaatioon liittyy kommentteja sekä DataEntryjä.

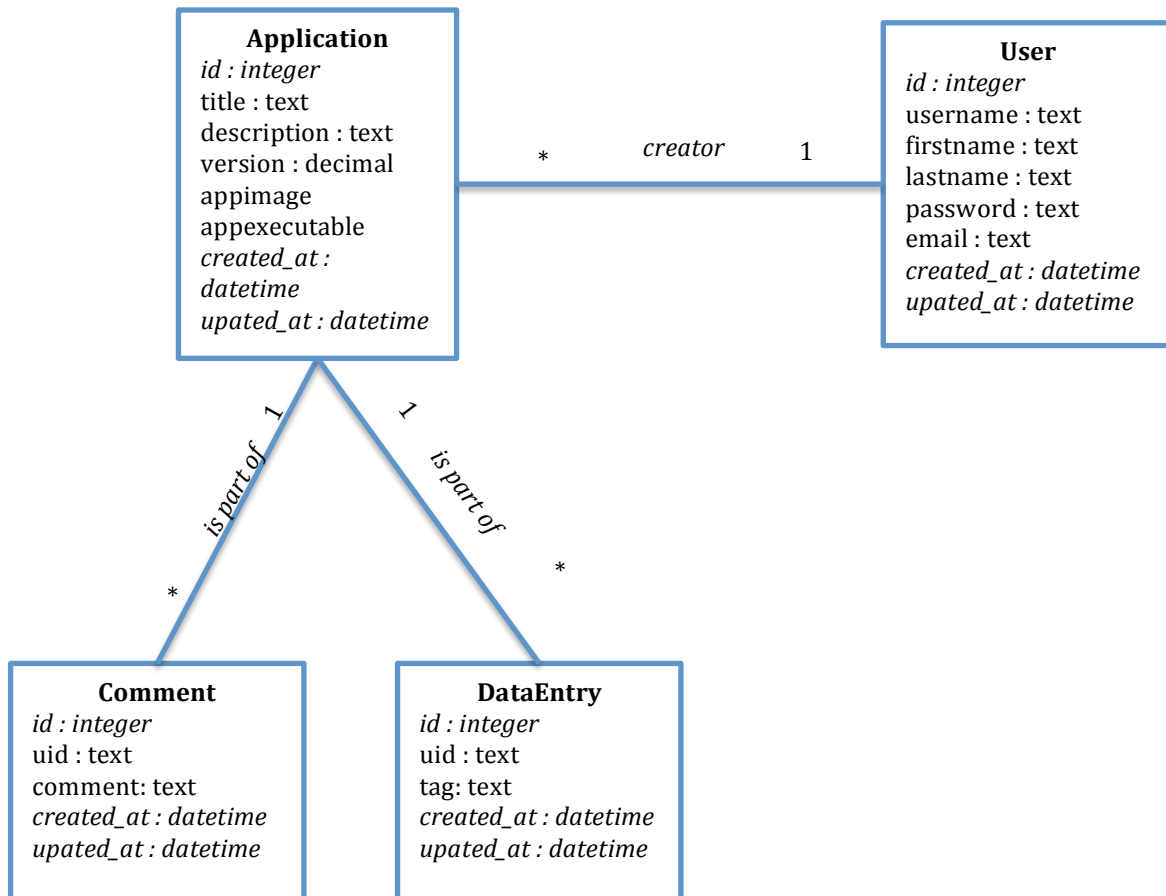
DataEntry

DataEntry kuvaa yksittäistä applikaation lähettämää tietosisältöä ohjelmistoon. Sellanen voi olla esimerkiksi tieto applikaation käynnistämisestä.

DataEntryllä on aikaleima, lähettävän laitteen ID-numero, joka tässä tapauksessa oletetaan olevan IMEI, toiminnon kuvaava termi, eli *tag*, sekä mahdollisia parametrejä.

Comment

Kommentti kuvaa yksittäistä applikaatioon annettua kommenttia. Se sisältää lähettävän laitteen ID-numeron, aikaleiman sekä varsinaisen kommentin.



Kuva 1 Ohjelmiston tietosisällöt ja niiden mallit

Käyttökokemus

Käyttökokemusta on hahmoitettu seuraavien mockup-kuvien kautta, jotka osoittavat mahdollisia kehityssuuntia.



Application 1
 A description of the application
[Download](#)



Application 2
 A description of the application
[Download](#)



Application 3
 A description of the application
[Download](#)



Application 4
 A description of the application
[Download](#)



Application 5
 A description of the application
[Download](#)



Application 6
 A description of the application
[Download](#)

[Information](#)

created with Balsamiq Mockups - www.balsamiq.com

Kuva 2 Ohjelmiston päänäköymä






Add new application


Name	<input type="text"/>
Version number	<input type="text"/>
Icon	<input type="text"/> <input type="button" value="File"/>
File	<input type="text"/> <input type="button" value="File"/>
Description	<input type="text"/>
<input type="button" value="Add application"/>	

created with Balsamiq Mockups - www.balsamiq.com

Kuva 3 Näkymä uuden applikaation lisäämiseen

Demo application

	
Application 1	
	
Application 2	
	
Application 3	
	
Application 4	
	
Application 5	



Fancy description

[Download](#)

Lorem ipsumam jne. aikaisempia kommentteja

created with Balsamiq Mockups - www.balsamiq.com

Kuva 4 Yksittäisen applikaation näkymä

Demo application statistics

Total application starts 10000

Application users 100

Application starts per user 100

Avarage application life span 5 days

created with Balsamiq Mockups - www.balsamiq.com

Kuva 5 Yksittäisen applikaation näkymä tilastojen osalta

Tekninen toteutus

Johdanto Ruby on Rails-ympäristöön

Ruby on rails perustuu model-view-controller arkkitehtuuriin ja sen soveltamiseen verkkopalveluiden tuottamisessa. MVC-arkkitehtuuri perustuu näkymän (view) eroittamiseen toimintalogiikasta (controller) ja toimintalogiikan käyttämästä tietosisällöstä (model).

Lisäksi Ruby on Rails noudattaa verkkokehityksessä yleistä REST-paradigmaa (representational state transfer), joka on yleisesti käytössä verkko-ohjelmien kehityksessä. Sen keskeisiä ajatuksia on käyttää HTTP-protokollan keskeistä neljää toimintoa (GET, POST, DELETE ja PUT) viestien välittämiseen. Idea on siis, että erilaisilla toimintoja (usein näistä käytetään nimitystä verbi) on erilaisia merkityksiä.

Verbi	Merkitys
GET	Tietosisältöjen nouto niitä muokkaamatta
POST	Uuden tietosisällön asettaminen
PUT	Tietosisällön päivittäminen
DELETE	Tietosisällön tuhoaminen

Idea siis on, että tiettyyn tietosisältöön (resurssiin) liitetään erilaisia verbejä. Esimeinä tällaisesta voisi olla

```
GET http://www.example.com/applications
GET http://www.example.com/applications/1
PUT http://www.example.com/applications/1
DELETE http://www.example.com/applications/1
```

Tässä ohjelmassa käytössä on seuraavat toiminnallisuudet

```
GET      /applications/:id/statistics(:format)
POST     /applications/:application_id/data_entries(:format)
GET      /applications/:application_id/data_entries/new(:format)
GET      /applications/:application_id/comments(:format)
POST     /applications/:application_id/comments(:format)
GET      /applications/:application_id/comments/new(:format)
GET      /applications/:application_id/comments/:id/edit(:format)
GET      /applications/:application_id/comments/:id(:format)
PUT      /applications/:application_id/comments/:id(:format)
DELETE   /applications/:application_id/comments/:id(:format)
GET      /applications(:format)
POST     /applications(:format)
GET      /applications/new(:format)
GET      /applications/:id/edit(:format)
GET      /applications/:id(:format)
PUT      /applications/:id(:format)
DELETE   /applications/:id(:format)
POST     /users/login(:format)
GET      /users/logout(:format)
POST     /users/:user_id/applications(:format)
GET      /users/:user_id/applications/new(:format)
GET      /users/:user_id/applications/edit(:format)
GET      /users/:user_id/applications(:format)
PUT      /users/:user_id/applications(:format)
DELETE   /users/:user_id/applications(:format)
GET      /users(:format)
```

```

POST    /users(:format)
GET     /users/new(:format)
GET     /users/:id/edit(:format)
GET     /users/:id(:format)
PUT     /users/:id(:format)
DELETE  /users/:id(:format)
GET     /

```

Modelit

Keskeiset mallit on kuvattu aikaisemmin tässä dokumentissa (katso kuva 1). Näihin malleihin liittyvät tiedot (attribuutit) on kuvattu myös samassa kuvassa. Erityistä Ruby on Railssin malleissa on mahdollisuus liittää validaattoreita (validators) ja metodeita yksittäisiin malleihin.

Validaattorit varmistavat, että tietosisällöt täyttävät niille asetetut ehdot. Käytännössä nämä siis estävät vääränlaisen sisällön asettamisen. Tässä projektissa on käytetty validointeja.

Kuten monissa muissakin kielissä, RoR sallii julkiset ja yksityiset metodit. Julkisia metodeita on käytetty keskeisten tietojen helpompaan muokaamiseen, kuten käyttäjän nimen esitysasuun tai kommenttien lukumäärän tarkasteluun.

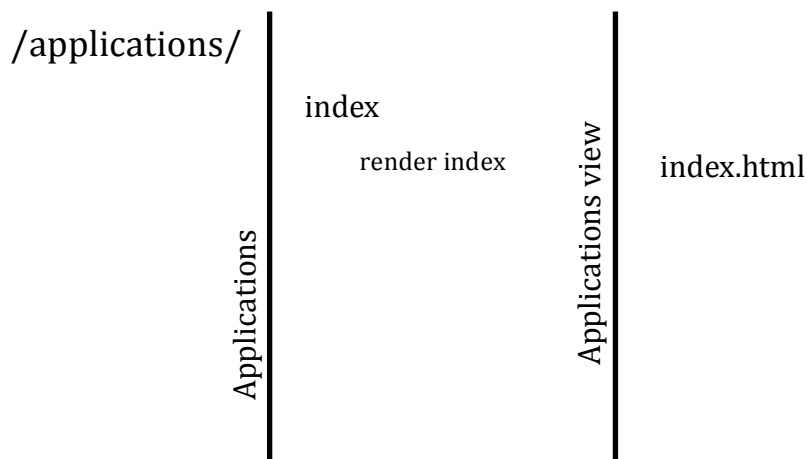
Controllerit ja viewit

RoRin logiikka on, että pyyntö välitetään controllerille, joka muodostaa siitä vastauksen erilaisia näkymiä käyttämällä. Kuva 6 selkeyttää tätä toimintamallia.

On syytä huomauttaa kaksi erityistä mallia, joita on käytetty yleisesti. Ensimmäinen on `application_controller`, joka on saatavissa käyttöön käytössä olevasta controllerista riippumatta. Sen keskeiset metodit ovat

`current_user`, joka suoritetaan ennen kaikkia näkymiä – asettaa `@current_user` –muuttujaan nykyisen käyttäjän session perusteella.

`is_logged_in` toimii yleisenä tarkastimena sille, onko käyttäjä kirjautunut sisään ja jos käyttäjä ei ole kirjautunut sisään, hänet ohjataan tyhjälle sivulle.



Kuva 6 Controller-view -toiminta

Sessio

Sessio on erityinen tallentamisrakenne, joka mahdollistaa yksittäisen käyttäjän tunnistamisen applikaation logiikassa. Käytännössä session muodostaa sellainen jakso, jonka aikana käyttäjä ei sammuta selaintaan. Sessiota käytetään tässä sovelluksessa tallentamaan tieto sisäänkirjautuneesta käyttäjästä eri näkymien välillä.

Ohjelmiston käyttöönotto

Ympäristön asennusohjeet

Toimiakseen Ruby on Rails vaatii ruby-kielen (mieluiten versio 1.9.2) ja Gemfiles-tiedostossa määritellyt asennuspaketit (gemit).

SQL skeema

Jostain syystä tehtävänannossa vaadittiin SQL-skeemaa, enkä keksinyt sille parempaa paikkaa kuin tässä. Eli, SQL-skeema, jolla tuon tietokannan saisi joku, jos haluaa niin pystyyn. Tämän sijaan kannattaa käyttää rake db:migratea, jonka mainitsen seuraavassa kappaleessa.

```
CREATE TABLE "applications" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "title" varchar(255), "description" text, "version" decimal, "created_at" datetime, "updated_at" datetime, "appimage" varchar(255), "appexecutable" varchar(255), "user_id" integer);

CREATE TABLE "comments" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "comment" text, "created_at" datetime, "updated_at" datetime, "application_id" integer, "uid" varchar(255));

CREATE TABLE "data_entries" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "time" datetime, "key" varchar(255), "created_at" datetime, "updated_at" datetime, "application_id" integer, "uid" varchar(255));

CREATE TABLE "schema_migrations" ("version" varchar(255) NOT NULL);

CREATE TABLE "users" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "username" varchar(255), "password" varchar(255), "firstname" varchar(255), "lastname" varchar(255), "email" varchar(255), "created_at" datetime, "updated_at" datetime);

CREATE UNIQUE INDEX "unique_schema_migrations" ON "schema_migrations" ("version");
```

Asetukset

Asetukset sijaitsevat config/ -kansiossa. Keskeistä on database.yml-tiedosto, jossa määritellään tietokantayhteydet sekä environments/-kansiossa määritellyt ympäristöt.

Tällä hetkellä käytössä on sqlite-tietokantaympäristö ja development-ympäristö, joka käynnistää sovelluksen porttiin 3000.

Ohjelman ensimmäinen käyttökerta

Ohjelma on tietokantasovellus, jonka takia ennen käynnistystä tietokantataulut on luotava.

Tätä varten aja komento

```
rake db:migrate
```


Tämä muodostaa tarvittavan tietokantarungon (skeeman) tietokantaan. Tämän jälkeen voit käynnistää palvelin komennolla

```
rails server
```

jonka jälkeen ohjelman tulisi käynnistyä aikaisemmin määrittelemään porttiisi. Ohjelman voi sammuttaa komennolla Control-C.

Myöhemmät käynnistyskerrat

Koska tietokanta on jo luotu, niin nyt on tarpeen vain käynnistää palvelin komennolla

```
rails server
```

ja sen voi sammuttaa komennolla Control-C.

Raportointi

Tehtävä	Päivämäärä	Aika (tunneissa)
Suunnitteludokumentin kirjoittaminen	20.7.2011	3
Automaattinen koodin generointi	22.7.2011	1
Käyttöliittymäsuunnittelu	23.7.2011	3
Suunnitteludokumentin kirjoittaminen	Ja tähän tämä kirjanpito päättyykin :D	