Metropolia

Matti Koskinen

REST-arkkitehtuurimallia noudattavan yhteisöpalvelu Web-ohjelmointirajapinnan toteuttaminen PHP:llä

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikka
Insinöörityö
20.4.2012

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Matti Koskinen REST-arkkitehtuurimallia noudattavan yhteisöpalvelu Web-ohjelmointirajapinnan toteuttaminen PHP:llä 46 sivua + 1 liite 20.4.2012
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja(t)	chief technology officer Kimmo Valtonen lehtori Olli Hämäläinen

Tämän insinöörityön tarkoitus oli tuottaa REST-arkkitehtuurimallia noudattava Web-ohjelmointirajapinta PHP-ohjelmointikieltä käyttäen. Tilaajayrityksenä toimi M-Brain Insight OY, joka tarvitsi yhteisöpalveluominaisuuksia osaksi laajempaa palvelukokonaisuutta. Työssä tutustuttiin REST-arkkitehtuurimalliin ja kaikkiin niihin työvaiheisiin, joita Web API-palvelun kehittämiseen ja käyttöönottoon kuuluu. REST-arkkitehtuurimallin myötä Web-palvelut saatetaan muiden palveluiden käytettäviksi. Palvelua hyödyntämään voidaan kehittää eri asiakaskäyttöliittymiä ja palvelukokonaisuuksia.

Työtä varten otettiin käyttöön palvelin, jossa oli esiasennettuna Ubuntu Linux –käyttöjärjestelmä. Palvelimelle asennettiin työtä varten Lighttpd-web-palvelin, MongoDB-tietokanta, Apache Subversion –versionhallintajärjestelmä sekä tarvittavat liitännäiset näiden käyttöönottamiseksi. Ohjelmointityössä noudatettiin MVC-arkkitehtuurin periaatteita ohjelmakoodin strukturoinnissa. Toteutettu Web API –palvelu käsittelee pyyntöjen ja vastauksien runkoina JSON-muotoista tietoa.

Työssä määriteltiin User-, UserGroup- ja Notification-resurssit, joihin kaikki tuetut pyynnöt tehdään. User-resurssi määrittää todellisen käyttäjän, joka todennetaan Web-palvelun käyttäjäksi. UserGroup-resurssi on käyttäjän perustama ryhmä, johon muut käyttäjät voivat liittyä tai omistaja voi kutsua. Notification-resurssi toimii väliaikaisresurssina, johon käyttäjä tarpeen mukaan reagoi riippuen muiden käyttäjien tekemistä pyynnöistä liittyä ryhmään tai käyttäjän kontaktiksi. User- ja UserGroup-resursseille on työssä määritelty kolme eri yksityisyysasetusta, joiden perusteella määritellään näkyvyys ja kanssakäymistaso muiden palvelun käyttäjien kesken.

Työtä voidaan suoraan käyttää itsenäisesti tai osana omaa Web-palvelua tai -sovellusta. Se toimii myös esimerkkinä kuinka nykyaikaisia REST-arkkitehtuurimallia noudattavia Web-ohjelmointirajapintoja toteutetaan PHP-ohjelmointikielellä. Työssä ratkaistaan useita tilanteita ja ongelmia, joita REST-arkkitehtuurimallia noudattavan Web API-palvelun kehittäjä kohtaa.

Avainsanat	Web API, REST, yhteisöpalvelu, PHP, web-palvelu

Metropolia

Abstract

Author Title Number of Pages Date	Matti Koskinen Implementing a RESTful community web API using PHP 46 pages + 1 appendix 20 April 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Kimmo Valtonen, Chief Technology Officer Olli Hämäläinen, Senior Lecturer

The purpose of this bachelor's thesis was to develop and implement a RESTful Web API using the PHP programming language. The client for this work was a company called M-Brain Insight OY. The client needed a community service for their collection of services. During the project, the REST architecture and all the necessary steps required to develop a fully functioning Web API service from scratch were studied. In general, the REST architecture allows Web services to be used by other services. Utilizing this so-called mashup service it is possible to develop and implement different types of client user interfaces and service packages.

For this project a server with a preinstalled Ubuntu Linux operating system was adopted. On this Lighttpd web server, a MongoDB database, an Apache Subversion version control system and all necessary plugins to make these work together were installed. The produced service follows the MVC architecture in code and file structure. All request and response bodies are handled as JSON-formatted data.

The service defines User, UserGroup and Notification resources which handle all supported requests. The User resource defines a real user who is authenticated as a Web API user. The UserGroup resource is a user-created group which allows other users to join in. It is also possible for the owner to send invites to other users to join the specified group. The Notification resource acts as a temporary resource which the user reacts to depending on requests made by other users to join a group or become a contact to the user. The User and the UserGroup resources also define three different privacy settings which set the level of visibility and interaction between other users of the service.

The service can be used as such or as a part of one's own Web service or application. It is an example of how modern RESTful Web APIs are developed using PHP. This thesis solves many of the situations and problems that a software developer implementing a RESTful Web API will face.

Keywords	Web API, REST, community, PHP, web service

Sisällys

Lyhenteet

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
PHP	Hypertext Preprocessor
REST	Representational State Transfer
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

1 Johdanto

Insinöörityössä tuotetaan yhteisöpalvelun Web-ohjelmointirajapinta (Application programming interface, API) käyttäen REST-arkkitehtuurimallia. Toteutettava palvelu ottaa kantaa käyttäjiin, käyttäjäryhmiin sekä näiden keskinäisiin toimintoihin. REST-arkkitehtuurimallia toteuttava Web API mahdollistaa toisistaan riippumattomien palveluiden lähettää ja vastaanottaa pyyntöjä käyttäen HTTP-protokollaa. Vuonna 2000 Roy Fieldingin väitöskirjassa esitelty ja määritelty Representational state transfer (REST) –arkkitehtuurimalli on yhä enenevämmässä määrin Webiin toteutettavien palveluiden pohjana.

Insinöörityön tarkoituksena on tutustua REST-arkkitehtuuriin sekä suunnitella ja toteuttaa käyttäjien ja käyttäjäryhmien osalta palvelu PHP-ohjelmointikielellä. Työhön kuuluu myös WWW-palvelinohjelman ja tietokannan asentaminen ja käyttöönotto sekä projektin liittäminen Subversion-versiohallintajärjestelmään. Tavoitteena on saada syvällinen tietotaito arkkitehtuurimallista, toteuttaa rajapinnat noudattaen REST-arkkitehtuurimallin suunnitteluperusteita ja esittää Web API –palvelun toteutuksen työnkulku alusta alkaen. Insinöörityö tuotetaan M-Brain Insight Oy:lle osaksi laajempaa palvelukokonaisuutta.

M-Brain Insight Oy on verkkoviestinnän seurantaan erikoistunut tietopalveluyritys. Palvelussa yhdistyy itse kehittämä hakuteknologia ja ihmisen tekemä uutisten arviointi, muokkaus ja analyysi.

M-Brain Insightin teknologia on lähes kokonaan oman kehitystyön tulosta ja perustuu uusimpiin hakukonemenetelmiin ja tutkimustuloksiin koneoppimisen alalla. Teknologian kulmakivenä on ajatus käyttää automaattisesti luotua sisällön merkityksen kuvausta avainkäsitteenä.

2 Työn taustaa

Tilaajayrityksen suunnitteilla olevia tuotteita varten haluttiin lisätä yhteisöpalveluominaisuuksia, jotka mahdollistavat tuotteiden käyttäjien keskinäisen kanssakäymisen sekä uuden tietämyksen luomisen ja jakamisen. Aikaisessa vaiheessa määriteltiin, että lopullinen sovellus koostuisi useista itsenäisistä moduuleista ja yksittäiset palvelut tehtäisiin mahdollisimman yleisiksi noudattaen REST-arkkitehtuurimallia. Insinöörityötä varten toteutettiin palvelu, joka määrittää käyttäjät, käyttäjien väliset toiminnot, käyttäjäryhmät ja käyttäjien ja käyttäjäryhmien väliset toiminnot. Lisäksi toteutettu palvelu määrittää kolme eri yksityisyystasoa käyttäjille ja käyttäjäryhmille.

Yrityksen jo olemassa olevat palvelut koostuvat useista REST-arkkitehtuurimallia noudattavista palveluista. Yksittäiset tiedonkäsittelytoiminnot on pilkottu omiksi palveluikseen. Asiakasrajapinnassa toimiva palvelu kokoaa tarvittavat tiedot eri palveluista käyttöliittymää varten. Jokainen toiminto on näin mahdollisimman läpinäkyvä tehden jatkuvasti lokimerkintöjä, jotta virhe- ja poikkeustilanteet ovat helposti selvitettävissä.

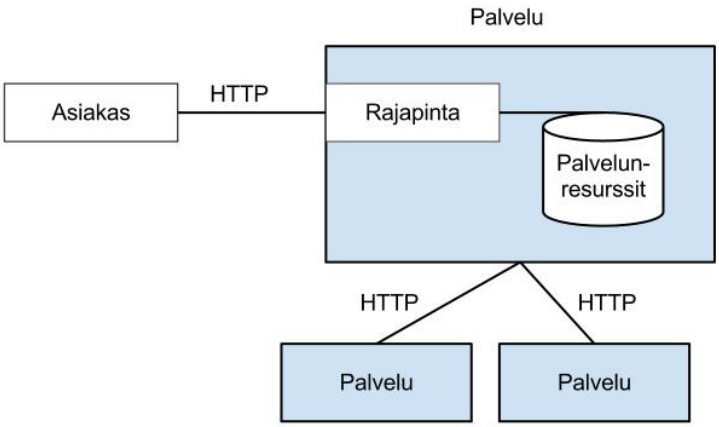
Yksittäiset palvelukokonaisuuden osat ovat myös helposti siirrettävissä palvelimelta toiselle ja monistettavissa. Näin prosessori- tai muisti-intensiivisiä tehtäviä voidaan tarvittaessa jakaa eri palveluinstansseille palvelinten sen hetkisen kuorman mukaan.

Palveluita voidaan käyttää osana muitakin palvelukokonaisuuksia, joten toteutus ei ole täysin riippuvainen sen hetkisestä palvelutarpeesta tai käyttötapauksesta. Varsinainen työ koostuukin eri palveluiden käyttöönottamisesta ja integroimisesta uudeksi kokonaisuudeksi ja käyttöliittymän tarjoamisesta asiakkaalle.

3 Ohjelmoitava Web

Webin alkuperäinen tarkoitus oli mahdollistaa tutkijaryhmien välinen tiedon jakaminen ja dokumenttien toisiinsa linkittäminen visuaalisesti. 90-luvulta saakka jatkuvasti kasvanut käyttäjämäärä on kasvattanut Webissä olevan tiedon ja palveluiden määrää valtavasti. Web-sovellusten lisäksi kehittäjät rakensivat palveluita, joita toiset palvelut pystyivät käyttämään. [2, s. 8.]

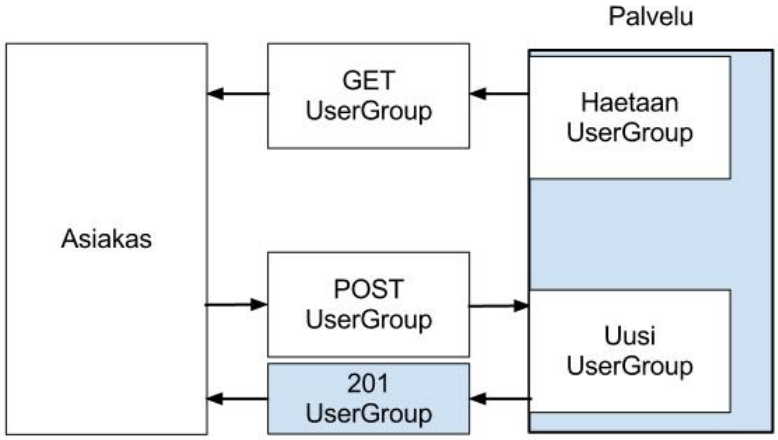
Ohjelmoitava Web tarkoittaa teknologioita ja työkaluja, jotka auttavat keskenään keskustelevien palveluiden kehittämisen Webiin. Kuviossa 1 on esiteltynä rakennekaavio Web API-palvelusta, joka hyödyntää muita palveluita. Web-palvelut käyttävät HTTP-protokollaa tiedonsiirtoon ja formaatteja, kuten XML ja JSON, tiedon esitykseen. Selaimessa ajettava AJAX-tekniikka on tehnyt Web-sovelluksista käyttäjälle entistä interaktiivisempia, nopeampia ja käyttäjäystävällisempiä. Ennen AJAXia Webin selaaminen tapahtui täysin tilattomasti. [2, s. 10.]



Kuvio 1. Web-palvelun rakennekaavio. Palvelu voi koostua useista pienemmistä palveluista ja se voi hyödyntää muita palveluita, jolloin se toimii itse asiakkaana.

HTTP-protokolla tiedonsiirron lisäksi välittää tiedon vastauksen tilasta sekä siitä, missä muodossa tieto sisältyy pyynnössä ja vastauksessa. HTTP-protokolla on normaalisti tilaton, mikä tarkoittaa, että jokainen pyyntö suoritetaan itsenäisesti ilman tietoa aiemmin tapahtuneista pyynnöistä. Tiedonesitysformaatit, kuten XML ja JSON, sisältävät palvelukohtaisen ja palvelun toimintaan liittyvän tiedon. [2, s. 11.]

Webin tieto- ja palvelumäärän hallitseminen on suuri haaste kelle tahansa kehittäjälle. Sen sijaan että toteuttaisi itse palvelun jokaisen osa-alueen ja mahdollisesti käyttäisi useita tunteja virheiden etsimiseen ja korjaamiseen, voisi olla tehokkaampaa integroida jokin olemassa oleva ja hyväksi havaittu palvelu osaksi omaa palvelua. Hyvä esimerkki tästä voisi olla yhteisöpalvelu, joka REST-arkkitehtuurimallia käyttäen hyödyntäisi Flickr-kuvapalvelua uusien valokuvien lisäämiseen ja jakamiseen sekä Yahoo! Weather RSS Feediä käyttäjän paikallissään näyttämiseen. [2, s. 16.]



Kuvio 2. Esimerkkikaavio asiakkaan ja palvelun välisestä pyyntöjen välityksestä.

Kuviossa 2 on esiteltynä asiakkaan ja palvelun välinen pyyntöjen välitys, jossa palveluun tehdään ensin hakupyyntö UserGroup-resurssin kokoelmaan, jonka jälkeen luontipyyntö. Palvelin vastaa luontipyyntöön vastauskoodilla, jonka avulla asiakas voi todentaa pyynnön onnistumisen.

4 Ympäristö ja työkalut

4.1 Työkalujen esittely

Insinöörityön varsinainen toteutus koostuu useista eri komponenteista, jotka yhdessä muodostavat toimivan kokonaisuuden. Työ tehtiin kokonaisuudessaan Linux-palvelimelle ja sen ohjelmointiin käytettiin Mac OS X –käyttöjärjestelmälle kehitettyä Coda-webkehityssovellusta. Palvelun testaaminen suoritettiin Chrome-selaimeen saatavalla Advanced Rest Client Application –liitännäisellä, jolla voidaan tehdä kaikki tarvittavat HTTP-pyynnöt ja tarkistaa palvelimen palauttama vastaus. Kuvioissa 3, 4, 5 ja 6 on esiteltynä Advanced Rest Client Application –liitännäisen eri näkymiä.

Kuvio 3. User-resurssiin GET-pyyntö Advanced Rest Client –liitännäistä käyttäen.

Pyynnössä määritellään URL, johon pyyntö halutaan tehdä, metodi jota halutaan käyttää ja mahdollisesti tarvittavat otsakkeet. Metodista riippuen pyyntö voi sisältää myös rungon.

Kuvio 4. Palvelimen palauttama HTTP-vastauskoodi, pyynnössä kestänyt aika millisekunteina ja otsakkeet. Aikaleimat ovat RFC 2822 –standardin määrittämässä muodossa.

Palvelin vastaa pyyntöön palauttamalla HTTP-vastauskoodin, otsakkeet ja mahdollisesti rungon. Asiakkaan kannalta vastauskoodi ja otsakkeet ovat jopa tärkeämpiä kuin vastauksen runko.


```
{
 - "data": [size(1)
    - {
        "_id": "4e1ebc42eca4884019000001",
        "api_key": "api_4e1ebc42460662.00295338",
      - "contact": [size(3)
            "4f26bc3eeca488d225000000",
            "4f26cee5eca488cd25000000",
            "4f4bc1a5eca488ea52000000"
        ],
        "email": "test@test.fi",
        "name": "Testi Ukko",
      - "notification": [size(7)
            "4f1eb59ceca488ce49000000",
            "4f1eb822eca488cb49000000",
            "4f26bc96eca488d425000000",
            "4f26cf4aeca488d225000001",
            "4f392b3eeca4887132000000",
            "4f3d2f75eca4887932000000",
            "4f4bc1cfeca488ec52000000"
        ],
        "privacy": "private"
    }
  ]
}
```


Kuvio 5. Palvelimen palauttama vastauksen runko JSON-muodossa.

Vastauksen runko sisältää tietorakenteen ”data”, joka sisältää yhden tai useamman JSON-olion. Virhetilanteessa rungossa palautettava ”status”-olio sisältää muuttujat vastauskoodille ja selventävälle viestille.


```
{
 - "status": {
      "status_code": 401,
      "status_message": "Unauthorized: Please provide valid email & password."
  }
}
```


Kuvio 6. Palvelimen palauttama vastauksen runko virhetilanteessa JSON-muodossa.

4.2 Representational State Transfer

4.2.1 Arkkitehtuurimallin esittely

Representational State Transfer (REST) on vuonna 2000 Roy Fieldingin väitöskirjassa esitelty ja määritelty arkkitehtuurimalli jaettujen hypermediajärjestelmien palveluiden toteuttamiseen. Tunnetuin jaettu hypermediajärjestelmä on World Wide Web. REST-arkkitehtuurimalli ei ole standardi tai protokolla. Sen periaatteisiin kuuluu, että palvelu määrittelee selkeästi resurssit, antaa jokaiselle resurssille yksilöllisen osoittimen, toteuttaa käytössä olevat metodit, on tilaton, tallennettavissa välimuistiin ja noudattaa olemassa olevia standardeja tai suosituksia. Se perustuu asiakas-palvelin-periaatteeseen ja on kehitetty rinnakkain HTTP 1.1 –spesifikaation kanssa noudattaen sitä. REST-arkkitehtuurimalli ei kuitenkaan ole sidottu HTTP-protokollaan. Pääpiireittäin REST-arkkitehtuurimalli valvookin kaikkien osapuolten eli palvelimen, yhdyskäytävän, välityspalvelimen ja asiakkaan toiminnan oikeellisuutta ilman että asettaisi rajoituksia yksittäisiin toimijoihin. [5, s. 1.]

REST-arkkitehtuurimallin suurimmat edut ovat sen yksinkertaisuus ja olemassa olevien standardien hyödyntäminen. Yksi alkuperäisistä pääsuunnittelupäämääristä oli toteuttaa arkkitehtuurimalli, jota noudattamalla palvelut skaalautuisivat hyvin Internetiin. Kun kehittäjä joutuu määrittelemään resurssit tarkasti, tulee palvelusta läpinäkyvämpi ja helpommin hahmotettavissa oleva. Tilattomuus tarkoittaa, että jokaisen pyynnön tulee sisältää kaikki tieto, minkä palvelin tarvitsee, että se voi toteuttaa pyynnön kokonaisuudessaan ja palauttaa vastauksen asiakkaalle. REST-arkkitehtuurimalli ei ota kantaa varsinaiseen toteutukseen tai siinä käytettyyn ohjelmointikieleen. Palvelun voi toteuttaa noudattamaan REST-arkkitehtuurimallia niiltä osin, kuin on palvelulle tarpeellista.

Resurssien selkeä määrittäminen helpottaa REST-palvelun käyttämistä, koska resurssista nähdään suoraan mistä on kyse. Kun toimintoina käytetään HTTP-metodeja, ei tarvitse keksiä tai määrittää mitään ylimääräistä, sillä HTTP-metodit voidaan suoraan määrittää CRUD-operaatioihin. CRUD tulee sanoista Create, Read, Update ja Delete eli luonti, luku, päivitys ja poisto. Taulukossa 1 on esiteltynä HTTP-metodit ja niitä vastaavat CRUD-operaatiot.

Taulukko 1. HTTP-metodit CRUD-operaatioina

Metodi	Toiminto
GET	Luku
PUT	Päivitys
DELETE	Poisto
POST	Luonti

REST-arkkitehtuurimalli madaltaa näin kynnystä luoda palvelu käyttäen yksinkertaisia ja ymmärrettäviä standardeja. Pyyntöjen vastaukset sisältävät vastauksen rungon (body) ja HTTP-vastauskoodin (HTTP-status code). Sovelluksen tai käyttöliittymän, joka jäsentää palvelimelta saadun vastauksen, tulisi ensimmäiseksi tarkastaa vastauskoodi ja sen jälkeen runko mahdollisia lisätietoja ja toimintoja varten.

HTTP-pyynnössä asiakkaan tulee ensin päättää, mitä asiakas haluaa tehdä ja mihin resurssiin. Toiminto määrittää käytettävän HTTP-metodin ja resurssi URL:n. Esimerkeissä sana, jota edeltää dollarimerkki ($), esittää kuvattua muuttujaa ja esimerkkipyynnöt tehdään user-nimiseen resurssiin.

Kuviossa 7 esitellyssä tapauksessa ”data”-tietorakenne voi olla vain yhden tai useamman elementin kokoinen kokoelma, eli se sisältää useita käyttäjiä. Kuviossa 8 ”data”-tietorakenne on yhden elementin kokoinen, eli se sisältää vain yhden käyttäjän.

```
-> GET http://$palvelin/api/user
<- 200 OK; {"data":[{"_id": "$user_id", "name": "$nimi"}, {...}]}
```

Kuvio 7. Hakupyyntö resurssin kokoelmaan.

```
-> GET http://$palvelin/api/user/$user_id
<- 200 OK; {"data":[{"_id": "$user_id", "name": "$nimi", "email": "$email", "..."}]}
```

Kuvio 8. Hakupyyntö yksilöityyn resurssiin.

Kuviossa 9 User-resurssia käytetään tehtaana, jossa luodaan uusi esitysmuoto. Kun palvelin on suoriutunut tehtävästä, palauttaa se HTTP-vastauskoodin ja HTTP-otsakkeessa tiedon, mistä kyseinen käyttäjä löytyy.

```
-> POST http://$palvelin/api/user; Body: {"name": "$nimi", "email": "$email", "..."}
<- 201 Created; Location: http://$palvelin/api/user/$user_id
```

Kuvio 9. Luontipyyntö resurssitehtaaseen.

Kuvion 10 toiminto on kuin uuden käyttäjän luominen, mutta päivityksessä yksilöidään resurssi, jonka esitysmuoto halutaan päivittää. Kuviossa 11 pyyntö poistaa käyttäjätunnuksella määritetyn käyttäjän.

```
-> PUT http://$palvelin/api/user/$user_id; Body: {"name": "$nimi", "email": "$email", "..."}
<- 201 Created; Location: http://$palvelin/api/user/$user_id
```

Kuvio 10. Päivityspyyntö yksilöityyn resurssiin.

```
-> DELETE http://$palvelin/api/user/$user_id
<- 200 OK
```

Kuvio 11. Poistopyyntö User-resurssiin.

4.2.2 Resurssit ja esitysmuodot

Palvelun määrittelyn ollessa valmis, voidaan siitä etsiä substantiivit, joilla kuvataan tietojoukkoja. REST-arkkitehtuurimallia noudattavissa palveluissa substantiivit ovat resursseja, joiden arviointiin tulee käyttää riittävästi aikaa, koska resurssit ovat palvelun ydin. Asiakkaan näkökulmasta resurssi on URI:lla osoitettu käsitteellinen olevainen. Esitysmuoto taas on konkreettinen, koska sitä käytetään ja siihen tehdään toimenpiteitä asiakkaalla ja palvelimella. Jokaiseen resurssiin tulee määrittää oma URI ja HTTP-metodi jokaista loogista toimintoa kohden. Yhdellä resurssi-URI:llä voi olla useita määriteltyjä HTTP-metodeja riippuen toiminnosta. [1, s. 45; 2, s. 78.]

Esitysmuoto on paljon enemmän kuin pelkästään tietoa esitettynä tietyssä muodossa. HTTP-protokollassa esitysmuodon metatieto tallennetaan otsakkeisiin nimi-arvo-pareina. Otsakkeet ovat vähintään yhtä tärkeitä kuin esitysmuodon tieto, niiden avulla muun muassa varmistetaan näkyvyys, löydettävyys, välimuistiin tallentaminen ja HTTP-protokollan oikea toiminnallisuus. [1, s. 46.]

4.2.3 HTTP-metodit

HTTP-protokolla on sovelluskerroksen protokolla, joka määrittää toiminnot esitysmuotojen siirtämiseen asiakkaan ja palvelimen välillä. Tässä protokollassa metodit, kuten GET, POST, PUT ja DELETE, ovat toimintoja resurssiin ja näin eliminoivat tarpeen itse keksiä toimintoja. [1, s. 1.]

HTTP-protokollassa turvallisuus ja idempotenssi ovat metodien ominaisuuksia, joiden implementoinnista palvelin on vastuussa. Turvallisuus tarkoittaa, että pyynnön ei oleteta aiheuttavan mitään sivuvaikutuksia. Asiakas voi toistaa pyynnön tietäen ettei se muuta resurssin tilaa. Nämä tulee implementoida lukutoimintoina. Idempotenssi tarkoittaa, että asiakas voi toistaa pyynnön ja sillä on sama vaikutus, kuin että pyynnön tekisi yhden kerran. Idempotenssillä on suurin merkitys verkko- tai palveluongelma tilanteissa. Idempotentit metodit vastaavat ohjelmoinnissa settereitä. Taulukko 2 näyttää mitkä metodit ovat turvallisia ja idempotentteja. [1, s. 10.]

Taulukko 2. HTTP-metodien turvallisuus ja idempotenssi.

Metodi	Turvallinen?	Idempotentti?
GET	On	On
HEAD	On	On
OPTIONS	On	On
PUT	Ei	On
DELETE	Ei	On
POST	Ei	Ei

GET-metodia käytetään turvallisiin ja idempotentteihin ”tiedon haku”-pyyntöihin. POST-metodia käytetään, kun halutaan luoda uusi resurssi, muokata vanhaa resurssia tai suorittaa toimintoja, jotka eivät ole turvallisia tai idempotentteja ja joihin muut metodit eivät ole sopivia. PUT-metodia voidaan käyttää uusien resurssien luomiseen ja vanhojen resurssien päivittämiseen. Ero POST-metodiin on, se että asiakas voi päättää uutta resurssia luodessa URI:n. Palvelun kuvauksesta käy ilmi jos tätä toimintoa tuetaan. Lisäksi PUT-metodi on idempotentti, joten asiakas voi huoletta lähettää pyynnön useita kertoja ja voi olla varma siitä, että lopputulos on sama. DELETE-metodilla poistetaan

yksittäinen resurssi ja sen esitysmuoto. HEAD-metodi on GET-metodiin verrattava toiminto, mutta palauttaa vain resurssin metadataa. OPTIONS-metodi palauttaa Web-palvelimen tukemat HTTP-metodit kyseiselle resurssille. [1, s. 13–23.]

4.2.4 HTTP-otsakkeet

Otsakkeet ovat tärkeä osa resurssin esitysmuotoa. Taulukossa 3 on esiteltynä toteutuksen pyynnöissä ja vastauksissa välitettäviä HTTP-otsakkeita. Jos palvelu vaatii HTTP-autentikoinnin tulee ”Authorization”-otsakkeessa välittää tarvittavat parametrit. Kun esitysmuoto sisältää vastausrungon, on hyvä sisällyttää otsakkeita, kuten ”Content-Type”, ”Content-Encoding” ja ”Last-Modified”. Uusien resurssien luonnissa tai uudelleen ohjauksessa ”Location”- ja ”Content-Location”-otsakkeet ovat välttämättömiä. Välityspalvelimet toimivat tehokkaasti, kun ne pystyvät tarjoamaan mahdollisimman monta vastausta ilman, että alkuperäiseen palvelimeen tarvitsee ottaa yhteyttä. Käyttäen ”Cache-Control”- ja ”Expires”-otsakkeita palvelu ja välityspalvelin tietävät, kuinka tulee toimia oikein, eikä vastauksissa välitetä vanhentunutta tietoa. [1, s. 17 ja s. 46 ja s. 148.]

Taulukko 3. Pyynnöissä ja vastauksissa välitettäviä HTTP-otsakkeita. [3; 1, s. 46 ja s. 49; 4.]

Otsake	Kuvaus
Authorization	HTTP-todennus tarvittavat tunnukset. Käytettävä ”basic”- tai ”digest”-tyyppistä ja niiden vaatimia parametreja palvelimen vastauksessa antamien ohjeiden mukaisesti.
Date	Päivämäärä, jolloin pyyntö lähetettiin.
Content-Type	Esitysmuodon mediatyyppi mahdollisesti sisältäen mediatyypin muita parametreja, kuten ”charset”-parametrin. Toteutuksessa käytetty ”application/json”-mediatyyppi ei määritä yhtään lisäparametria. ”application/json” on aina merkistömuodoltaan tyyppiä UTF-8, UTF-16 tai UTF-32. Jos UTF-16- tai UTF-32-tyyppiä käytetään, täytyy käyttää ”Content-Transfer-Encoding”-otsaketta arvolla ”binary”.
Content-Encoding	Otsaketta tulee käyttää, jos esitysmuoto siirretään pakatussa muodossa, esimerkiksi ”gzip”, ”compress” tai ”deflate”. Näin voidaan optimoida kaistan käyttöä ja nopeuttaa tiedonsiirtoa. Vastaanottajan tulee ohjeen mukaisesti purkaa paketti ennen jäsentämistä. Jos ei ole tiedossa, että palvelin tai asiakas tukee kyseistä pakkausmuotoa on tätä vältettävä.
Last-Modified	Pelkästään vastauksessa välitettävä tieto. RFC 2822-standardin mukainen aikaleima kertoo, koska viimeiset muutokset resurssin esitysmuotoon on tehty.

Cache-Control	Määrittää ohjeet, joita kaikkien pyyntö-vastaus –ketjussa olevien välityspalvelinten ja -mekanismien on noudatettava. ”must-revalidate”-ohje vaatii välityspalvelinten tarkistaa palvelimelta esitysmuodon tuoreus ennen tarjoamista. ”max-age”-ohje kertoo, kuinka kauan esitysmuoto säilyy tuoreena. Arvo annetaan sekunteina.
Expires	Päivämäärä, jolloin esitysmuoto muuttuu vanhentuneeksi.
Location	Tätä käytetään uudelleenohjauksessa, tai kun uusi resurssi on luotu kertomaan resurssin esitysmuodon sijainti.
Content-Location	Vaihtoehtoinen resurssin esitysmuodon sijainti. Esimerkiksi kun pyynnön seurauksena luodaan väliaikaisresurssi, voidaan Content-Locationilla osoittaa sen sijainti.

4.2.5 HTTP-vastauskoodit

HTTP-pyyntöjen vastauksena saadaan aina vastauskoodi, joka kertoo, kuinka pyynnön kävi palvelimella. Vastauskoodit on jaettu viiteen luokkaan: Informational, Success, Redirection, Client Error ja Server Error. Taulukossa 4 käydään läpi yleisimmät HTTP-vastauskoodit. Palvelun tulee huolehtia oikeiden vastauskoodien generoinnista ja niiden välittämisestä vastauksessa. On erittäin tärkeää, että oikeat vastauskoodit lähetetään oikeissa tilanteissa, jotta HTTP-protokollaa tarkasti implementoivat palvelut osaavat toimia oikein.

Success-luokan koodeilla (koodit 200–226) kerrotaan pyynnön onnistumisesta. Esimerkiksi onnistunut haku- tai luontipyyntö. Client Error- ja Server Error-luokat ovat virheitä kuvaavia koodeja. Hyviin tapoihin kuuluu välittää virhetilanteessa vastauskoodin lisäksi viesti osana vastausta, jotta asiakas osaa mahdollisesti seuraavaa pyyntöä varten tehdä oikeita muutoksia tai odottaa virhetilan raukeamista. [6.]

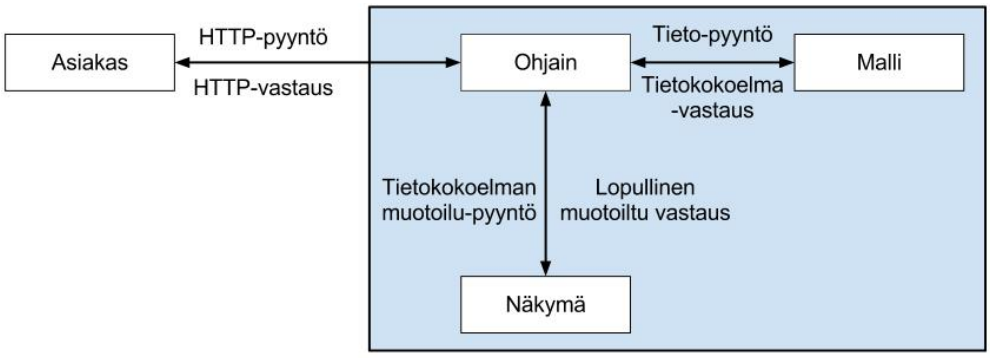
Client Error-koodit (koodit 400–499) kertovat asiakkaan pyynnön virheellisyydestä. Näitä voivat olla esimerkiksi syötteen väärä sisältö, puutteelliset oikeudet tai pyydetyn resurssin puuttuminen. Server Error –koodeja (500–510) käytetään, kun pyyntö epäonnistuu jonkin palvelimella sijaitsevan virheen takia. Tämän kaltaisia tilanteita ovat esimerkiksi virhe palvelun ohjelmakoodissa tai yhteysvirhe tietokantaan. [1, s.71–72.]

Taulukko 4. Yleisimpiä HTTP-vastauskoodeja. [7.]

Vastauskoodi	Kuvaus
200	OK. Onnistunut HTTP-pyyntö.
201	Created. Uusi resurssi luotiin onnistuneesti.
202	Accepted. Pyyntö hyväksyttiin, mutta vaatii lisäkäsittelyä.
204	No Content. Onnistunut HTTP-pyyntö, mutta palvelin ei palauta runkoa.
303	See Other. Resurssi jota pyydettiin löytyy otsakkeessa määritellystä sijainnista.
400	Bad Request. Pyyntöä ei voitu täyttää virheellisen syötteen takia.
401	Unauthorized. Autentikointi epäonnistui tai ei ole annettu. WWW-Authenticate-otsake sisältää tietoa vaadittavasta autentikoinnista.
403	Forbidden. Pyyntö oli aito ja onnistunut, mutta palvelin ei suostu vastaamaan siihen. Viittaa pyynnön tekijän puutteellisiin oikeuksiin.
404	Not Found. Pyydettyä resurssia ei löytynyt.
405	Method Not Allowed. Pyyntö tehtiin käyttäen HTTP-metodia mitä kyseinen resurssi ei tue.
500	Internal Server Error. Yleinen virhe palvelimella.
501	Not Implemented. Palvelin ei ymmärrä pyyntöä tai ei kykene täyttämään sitä.

4.3 Model-view-controller

MVC-arkkitehtuurin idea on erottaa sovellus kolmeen eri päätekijäänsä. Sovelluksen käyttöliittymä on suoraan yhteydessä Controlleriin eli ohjaimeen. Ohjain toimii näkymän ja mallin välissä reagoiden käyttöliittymältä tuleviin pyyntöihin välittämällä ne mallille sekä näkymälle tarpeen mukaan. Model on sovelluksen malli, joka vastaa sovelluksen tiedon tallentamisesta, hakemisesta ja ylläpidollisista toiminnoista. View on sovelluksen näkymä, se on vastuussa mallilta saadun tiedon jäsentämisestä ja lopullisen esitysmuodon valmistamisesta. Kuvio 12 esittää MVC-arkkitehtuurin periaatteen yhtenä kaaviona. [8, s. 1.]



Kuvio 12. MVC-toimintakaavio.

MVC-arkkitehtuurin noudattamisella on useita hyviä puolia. Se kannustaa kirjoittamaan uudelleenkäytettävää standardisoitua ohjelmakoodia ja järjestämään tiedostot selkeästi omiin osa-alueittensa määrittämiin kansioihin. MVC:n käyttö helpottaa ulkopuolisen suorittamaa koodiarviointia ja muutosten tekemistä määrättyihin sovelluksen osiin ja helpottaa rinnakkaisen sovelluksen kehittämisen. [8, s. 3.]

Web-palveluita ohjelmoitaessa MVC-arkkitehtuurista käytetään yleensä vain mallia ja ohjainta. Tilanteissa, joissa palvelu tukee useita eri tiedonsiirtomuotoja, voi kuitenkin olla hyvä irrottaa vastauksen lopullinen muodostaminen näkymään, jotta lisätään selkeyttä ja skaalautuvuutta.

4.4 PHP: Hypertext Preprocessor

PHP: Hypertext Preprocessor eli PHP on ilmainen, avoimen lähdekoodin, yleiskäyttöinen, PERLin kaltainen ohjelmointikieli, joka sopii erinomaisesti sovellusten ja palveluiden kehittämiseen. PHP-koodiin voi liittää suoraan HTML:ää. PHP:n syntaksi ottaa mallia muilta kieliltä, kuten C, Java ja PERL. PHP on joustava ja voimakas kieli, joka käyttää omaa sisäänrakennettua muistialuetta, jolla nopeutetaan prosessointinopeutta ja vähennetään palvelinrasitusta. PHP:n saa toimimaan suurimmilla alustoilla, kuten Windows, Linux, Unix ja Mac OS X. Se on yhteensopiva lähes kaikkien Web-palvelinten kanssa. [9; 10; 11.]

PHP:n myötä palveluiden kehittämisestä on tullut helposti lähestyttävää. PHP integroituu tiukasti Web-palvelimen kanssa ja taipuu hyvin ketterään kehittämiseen. Pitää ottaa huomioon kuitenkin, että ohjelmointikielien vertailu on aina epätäydellistä ja suuris-

sa määrin subjektiivista. Kieltä valittaessa tuleekin tarkastella, mitä halutaan saavuttaa, mihin tarkoitukseen kieli on kehitetty, mitä sisäänrakennettuja ominaisuuksia siinä on ja miten laajasti ja aktiivisesti se on tuettu laajennusten suhteen. [12.]

4.5 JavaScript Object Notation

JavaScript Object Notation eli JSON on kevyt, tekstipohjainen, kieliriippumaton ja avoin tiedonsiirtomuodon standardi. Se on alun perin johdettu ECMAScript-ohjelmointikielen standardista. Standardoitu JavaScript noudattaa ECMAScriptiä. JSONin rakenteellisen tiedon esitystapa on johdettu suoraan JavaScriptin tietorakenteista ja assosiaatiotauluista. [3.]

JSON kuvaa neljä primitiivistä ja kaksi rakenteellista tietotyyppiä. Primitiivitietotyypit ovat merkkijono, numerot, boolean ja null. Rakenteelliset tietotyypit ovat oliot eli assosiaatiotaulut ja tietorakenteet. JSON teksti tulee olla Unicode-koodattua ja oletusarvoisesti se on UTF-8-muotoista. [3.]

JSON:n ja XML:n välillä on jo muutaman vuoden ajan ollut kädenvääntö Webin tiedonsiirtomuodon herruudesta. Tällä hetkellä näyttää siltä, että JSON on vienyt pidemmän korren. JSON on yksinkertaisempi kuin XML, ei määritä dokumenttia toisin kuin XML, ja tarjoaa paremman tietorakenne-esitysmuodon, joka soveltuu suoraan käytettäväksi ohjelmoinnissa suurimmalle osalle kielistä. JSON soveltuu parhaiten tiedonsiirtoformaatiksi ja XML dokumenttiensiirtoformaatiksi, eli oikeaa työkalua tulee käyttää siihen tarkoitettuun työhön. Koodilistauksessa 1 on esiteltynä yksinkertainen JSON-olio, joka sisältää standardin mukaiset tietotyypit. Koodilistauksessa 2 JSON-olio, jossa tietorakenteessa kaksi oliota. [13; 14.]

```
{
    ”Olio”:{
                    ”Numero”: 101.2,
                    ”Merkkijono”: ”Esimerkki 1”,
                    ”Boolean”: true
    },
      ”Tietorakenne”: [1, 2, 3, 4]
}
```

Koodilistaus 1. Esimerkki JSON-oliosta.

```
{
    ”Tietorakenne”: [
        {”Merkkijono”: ”Esimerkki 1”, ”Numero”: 101},
        {”Merkkijono”: ”Esimerkki 2”, ”Numero”: 65.5}
    ]
}
```

Koodilistaus 2. Esimerkki JSON-oliosta, jossa tietorakenteessa kaksi oliota.

4.6 MongoDB

MongoDB on skaalautuva, hyvän suorituskyvyn ja avoimen lähdekoodin NoSQL-tietokanta, jossa tieto tallennetaan JSON-tyylisinä dokumentteina. MongoDB tukee indeksointia, replikointia ja hajauttamista. Suurimmalle osalle ohjelmointikielistä löytyy suorat laajennukset, joilla saadaan täysi MongoDB-tuki. [15.]

MongoDB:n mukana tulee interaktiivinen JavaScript shell, jolla voidaan tehdä tietokantakomentoja suoraan komentoriviltä. Shell on hyödyllinen apuväline kun halutaan tarkastella tietokannan sisältöä, kokeilla pyyntöjä, luoda indeksejä ja suorittaa ylläpitoskriptejä. [16.]

4.7 Lighttpd

Lighttpd on avoimen lähdekoodin kevyt ja nopea HTTP-palvelin. Lighttpd on hyvä vaihtoehto Apachelle, jos ei ole tarvetta laajalle ominaisuuskirjolle vaan halutaan näppärä Web-palvelin. Lighttpd sisältää kehittyneitä ominaisuuksia kuten FastCGI, SSL/TLS-tuki, URL:ien uudelleenkirjoitus, LDAP-palvelimen kautta tunnistautuminen, hakemistojen salasanasuojaaminen ja HTTP-pakkaus. Lighttpd:n muistinkäyttö on erittäin pientä verrattuna muihin Web-palvelimiin, ja se hallitsee prosessorikuormaa tehokkaasti. [17; 18.]

4.8 Apache Subversion

Apache Subversion eli svn on versiohallintajärjestelmä, joka mahdollistaa tiedostojen ja niiden sisällön muokkaamisen hajautetusti tietoverkkossa. Toiminta perustuu keskuspalvelimeen, joka pitää versioarkistoa. Jokainen keskuspalvelinta käyttävä osapuoli pyytää itselleen työkopion. Työkopiota käyttäjä muokkaa ja päivittää muokkaukset keskuspalvelimelle ja näin saattaa muutokset muiden osapuolien saataville.

Apache Subversion on kaupallista laatua oleva järjestelmä, joka on edelleen niin pienten kuin suurtenkin projektien suosiossa. Svn pitää yllä säännönmukaisia versionumeroita, joten projektin etenemistä voidaan suoraan seurata aikajanalla. Svn:n versioarkistosta on mahdollista pyytää pelkästään tietty alipuu. [19; 20.]

Apache Subversionin käytöstä hyötyvät yhden henkilön projektitkin, koska palvelin toimii samalla varmuuskopiointitarkoitukseen. Versioarkistosta on helppo pyytää uudelleen koko projekti tai sen osa versioarkistosta halutulla versionumerolla haarauttamista tai uudelleenmuokkausta varten, jos huomataankin, että työkopio ei vastaakaan toivottua. Palvelun tuotantokäyttöön ottaminen onnistuu myös halutulle palvelimelle pyytämällä viimeisin versio.

5 Palvelun kuvaus

5.1 Toiminta lyhyesti

Toteutuksessa on määritelty User-, UserGroup- ja Notification-resurssit. Viimeksi mainittu on niin sanottu väliaikaisresurssi, jonka kautta käyttäjien sekä käyttäjien ja käyttäjäryhmien väliset hyväksymistä vaativat toiminnot suoritetaan. Jokaista reaalimaailman käyttäjää kohden tulee olla yksi määritelty User-resurssi. Asiakkaan todentuessa pyyntöjä varten tämän User-resurssin esitysmuodosta tarkistetaan oikea sähköpostiosoite ja API-avain.

Toteutuksessa kaikki resurssit tukevat GET-, HEAD-, OPTIONS-, POST- ja DELETE-metodeilla tehtyjä pyyntöjä. User- ja UserGroup-resurssit tukevat lisäksi PUT-metodia esitysmuodon päivittämiseen. User-resurssin omistaja on todennettu asiakas ja vain hänellä on kyky päivittää henkilökohtaisia tietoja. UserGroup-resurssin omistaja on todennettu resurssin luonut asiakas, ja vain hänellä on täysi hallintaoikeus resurssiin.

User- ja UserGroup-resurssit tukevat yksityisyysmääritystä. Haluttu taso määritellään resurssin esitysmuotoon luonnin yhteydessä ja se voidaan päivittää milloin vain omistajan toimesta. Yksityisyysasetuksella voidaan rajoittaa resurssin ja sen tietojen näkymistä julkisesti.

5.2 Resurssien ja metodien määritykset

Taulukoissa 5, 6 ja 7 kuvattujen resurssien ja metodien lisäksi jokainen resurssi tukee OPTIONS HTTP-metodia. Tällä pyynnöllä palvelin kertoo kyseisen resurssin tukemat HTTP-metodit. Taulukoissa sana, jota edeltää dollarimerkki ($), esittää kuvattua muuttujaa.

Taulukko 5. User-resurssi

URI	Metodi	Resurssi	Toiminto	Kuvaus
/user	GET	user	haku	Haetaan käyttäjät.
/user	POST	user	luonti	Lisätään uusi käyttäjä.
/user/$user_id	GET	user	haku	Haetaan käyttäjä.
/user/$user_id	PUT	user	päivitys	Päivitetään käyttäjä.
/user/$user_id	DELETE	user	poisto	Poistetaan käyttäjä.
/user/$user_id	HEAD	user	haku	Haetaan käyttäjän metatiedot.
/user/$user_id/contact	GET	user	haku	Haetaan käyttäjän kontaktit.
/user/$user_id/contact	POST	user	luonti	Pyydetään käyttäjä omaksi kontaktiksi. Käyttäjä tekee pyynnön ilman runkoa. Tämä pyyntö luo uuden Notification-resurssin.
/user/$user_id/contact	DELETE	user	poisto	Poistetaan käyttäjä kontakteista.
/user/$user_id/notification	GET	user	haku	Haetaan käyttäjän notifikaatiot.
/user/$user_id/usergroup	GET	user	haku	Haetaan käyttäjän käyttäjäryhmät.
/user/$user_id/api_key	GET	user	haku	Haetaan käyttäjän API-avain. API-avain on ainutlaatuinen merkkijono, joka toimii salasanana.
/user/$user_id/api_key	POST	user	luonti	Luodaan uusi API-avain. API-avain on ainutlaatuinen merkkijono, joka toimii salasanana.

Taulukko 6. UserGroup-resurssi

URI	Metodi	Resurssi	Toiminto	Kuvaus
/usergroup	GET	usergroup	haku	Haetaan käyttäjäryhmät.
/usergroup	POST	usergroup	luonti	Lisätään uusi käyttäjäryhmä.
/usergroup/$usergroup_id	GET	usergroup	haku	Haetaan käyttäjäryhmä.
/usergroup/$usergroup_id	PUT	usergroup	päivitys	Päivitetään käyttäjäryhmä.
/usergroup/$usergroup_id	DELETE	usergroup	poisto	Poistetaan käyttäjäryhmä.
/usergroup/$usergroup_id	HEAD	usergroup	haku	Haetaan käyttäjäryhmän metatiedot.
/usergroup/$usergroup_id/contact	GET	usergroup	haku	Haetaan käyttäjäryhmän kontaktit.
/usergroup/$usergroup_id/contact	POST	usergroup	luonti	Pyydetään päästä osaksi käyttäjäryhmää. Käyttäjä tekee pyynnön ilman runkoa. Omistaja voi kutsua muita käyttäjiä lisäämällä pyyntöön käyttäjätunnus. Tämä pyyntö luo uuden Notification-resurssin
/usergroup/$usergroup_id/contact	DELETE	usergroup	poisto	Poistetaan käyttäjä ryhmästä. Käyttäjä poistaa itsensä ryhmästä.
/usergroup/$usergroup_id/contact/$user_id	DELETE	usergroup	poisto	Poistetaan käyttäjätunnuksella määritelty käyttäjä ryhmästä. Ryhmän omistaja voi poistaa jonkun muun käyttäjän.

Taulukko 7. Notification-resurssi

URI	Metodi	Resurssi	Toiminto	Kuvaus
/notification	GET	notification	haku	Haetaan notifikaatiot.
/notification/$notification_id	GET	notification	haku	Haetaan notifikaatio.
/notification/$notification_id	POST	notification	päivitys	Asetetaan notifikaation tila. Tila voi olla ”accepted” tai ”declined”.
/notification/$notification_id	DELETE	notification	poisto	Poistetaan notifikaatio. Omistaja voi poistaa vain jos muuttuja ”status” on ”waiting”. Kohde voi poistaa vain jos muuttuja ”status” on ”declined”.
/notification/$notification_id	HEAD	notification	haku	Haetaan notifikaation metatiedot.

5.3 Pyynnöissä asiakkaan välittämät JSON-tietorakenteet

Luonti- ja päivityspyynnöissä tulee välittää JSON-olio, joka sisältää seuraavassa esitetyistä tietorakenteen tiedoista tarpeen mukaan kaiken tai osan. Sana, jota edeltää dollarimerkki ($), esittää kuvattua muuttujaa.

User-resurssi

```
1.  {
2.          ”name”: ”$nimi”,
3.          ”email”: ”$email”,
4.          ”privacy”: ”moderate”
5.  }
```

Koodilistaus 3. User-resurssin JSON-olio.

Koodilistauksessa 3 esitellyn JSON-olion muuttujat ”name”, ”email” ja ”privacy” ovat muodoltaan merkkijonoja. ”privacy”-merkkijono odottaa arvokseen joko ”public”-, ”moderate”- tai ”private”-arvon. Tällä JSON-tietorakenteella voidaan luoda tai päivittää User-resurssia. Päivitys voidaan tehdä kaikella tiedolla tai vain halutulla osalla. Palvelin palauttaa vastaavanlaisen tietorakenteen sisältäen lisätietoa api-avaimesta, kontakteis-

ta ja notifikaatioista. Tiedot Api-avaimesta ja notifikaatioista eivät näy muille kuin käyttäjälle itselleen. Käyttäryhmistä saadaan erikseen tieto tekemällä pyyntö user-resourssiin usergroup-lisämääreellä.

UserGroup-resurssi

```
1. {
2.         ”name”: ”$nimi”,
3.         ”privacy”: ”moderate”
4. }
```

Koodilistaus 4. UserGroup-resurssin JSON-olio.

Koodilistauksessa 4 muuttujat ”name” ja ”privacy” ovat muodoltaan merkkijonoja. ”privacy”-merkkijono odottaa arvokseen joko ”public”-, ”moderate”- tai ”private”-arvon. Tällä JSON-tietorakenteella voidaan luoda tai päivittää UserGroup-resurssia. Päivitys voidaan tehdä kaikella tiedolla tai vain halutulla osalla. Palvelin palauttaa vastaavanlaisen tietorakenteen sisältäen lisätietoa kontakteista ja omistajasta.

Käyttäjäryhmän omistaja voi kutsua ryhmään muita käyttäjiä tekemällä ryhmään pyynnön contact-lisämääreellä.

```
1. {
2.         ”contact”: ”$user_id”
3. }
```

Koodilistaus 5. Käyttäjäryhmän ulkopuolisen käyttäjän kutsupyynnön JSON-olio.

Koodilistauksessa 5 muuttuja ”contact” on muodoltaan merkkijono, ja se odottaa arvokseen oikeellisen käyttäjätunnuksen.

Notification-resurssi

Notification eroaa kahdesta aiemmin kuvaillusta resurssista siten, että palvelin vastaa sen luomisesta. Notification-resurssi on niin kutsuttu väliaikaisresurssi, jonka tarkoitus on pitää liittymispyynnön tila kummankin osapuolen tiedossa.

```
{
        ”status”: ”accepted”
}
```

Koodilistaus 6. Notifikaation hyväksymisen JSON-olio.

Notifikaation kohde voi hyväksyä henkilökohtaisen kontakti- tai käyttäjäryhmään liittymispyynnön määrittämällä ”status”-muttujan arvolla ”accepted”, kuten koodilistauksessa 6 on esitelty. Vastaavasti pyynnön voi hylätä tekemällä pyynnön ”declined”-arvolla.

5.4 Vastauksissa palvelimen välittämät JSON-tietorakenteet

Haku-, luonti-, päivityspyynnöissä ja virhetilanteissa palvelin palauttaa yleensä JSON-olion, joka asiakkaan tulee tarkistaa vastauskoodin ja otsakkeiden lisäksi. Sana, jota edeltää dollarimerkki ($), esittää kuvattua muuttujaa.

User-resurssi

```
{
     ”data”: [
       {
       ”_id”: ”$user_id”,
       ”api_key”: ”$api_key”,
        ”contact”: [
                     ”$user_id1”,
                     ”$user_id2”,
                     ”...”
                     ],
       ”email” : ”$email”,
       ”name”: ”$nimi”,
       ”notification”: [
                     ”$notif_id1”,
                     ”$notif_id2”,
                     ”...”
                     ],
       ”privacy”: ”moderate”
       }
       ]
}
```

Koodilistaus 7. User-resurssin JSON-olio.

Pyynnöstä riippuen palvelin palauttaa ”data”-tietorakenteen, jossa yksi tai useampia käyttäjiä. Yksittäisen käyttäjän pyyntö palauttaa koodilistauksessa 7 esitellyn kaltaisen JSON-olion. Käyttäjälle itselleen vain näytetään ”api_key”- ja ”notification”-kentät. ”notification”-kentän id-arvoa tulee käyttää Notification-resurssiin tehtävässä pyynnössä. Yksityisyysasetus ja kontaktius määrittävät, kuinka paljon tietoa käyttäjästä muille näytetään.

UserGroup-resurssi

```
{
    ”data”: [
        {
      ”_id”: ”$usergroup_id”,
      ”contact”: [
                    ”$user_id1”,
                    ”$user_id2”,
                    ”...”
                    ],
      ”name”: ”$nimi”,
      ”owner_id” ”$user_id”,
      ”privacy”: ”moderate”
      }
      ]
}
```

Koodilistaus 8. UserGroup-resurssin JSON-olio.

Kuten User-resurssissakin koodilistauksessa 8 esitellyn UserGroup-resurssin JSON-olion ”data”-tietorakenne sisältää yhden tai useamman ryhmiä. Yksittäisestä ryhmästä yksityisyysasetuksista riippuen näytetään koodilistauksen 8 kaltaiset tiedot. ”contact”-tietorakenne sisältää ryhmään kuuluvat henkilöt, ”name” on ryhmän nimi merkkijonona, ”owner_id”-merkkijono kertoo, kuka ryhmän on luonut ja kuka sitä hallitsee.

Notification-resurssi

```
{
        ”data”: [
        {
        ”_id”: ”$notif_id”,
        ”owner_id”: ”$user_id”,
        ”resource”: ”user”,
        ”resource_id”: ”$user_id1”,
        ”target_id”: ”$user_id1”,
        ”status”: ”waiting”,
        ”status_message”: ”Request to resource  user is waiting response.”
        }
        ]
}
```

Koodilistaus 9. Notification-resurssin JSON-olio.

Notification-resurssi palauttaa todentamisen perusteella käyttäjän notifikaatiot kokoelmana. Koodilistauksessa 9 esitellyn yksittäisen notifikaation ”data”-tietorakenteessa ”owner_id” on notifikaation aiheuttajan käyttäjätunnus, ”resource”-muuttuja kertoo, mitä resurssia notifikaatio koskee, ”resource_id” määrittää tämän resurssin, ”target_id” on käyttäjä, jonka tulee reagoida notifikaatioon, ”status”-merkkijono kertoo notifikaation tilan ja ”status_message” kuvaa tilaa ymmärrettävämmin.

Virhetilanne

```
{
        "status": {
                        "status_code": 401,
                        "status_message": "Unauthorized: Please provide valid email
    & password"
        }
}
```

Koodilistaus 10. Virhetilanteen JSON-olio.

Koodilistauksen 10 status-olio sisältää ”status_code”-kokonaisluvun, joka vastaa HTTP-vastauskoodia ja ”status_message”-viestin, jossa selvennetään virhetilannetta ja mahdollisesti suositellaan tarvittavia toimenpiteitä.

5.5 Todennus

Jotta pyyntöjä pääsee tekemään, on asiakas ensin todennettava. Palvelu tukee HTTP Basic- ja Digest-todennustapoja, joista toinen tulee määrittää palvelussa käyttöön. Basic-todennuksessa lähetetään käyttäjänimi ja salasana käytännössä selväkielisenä Authorization-otsakkeessa. HTTP 1.1:ssä spesifioitu Digest-todennus on suunniteltu turvallisemmaksi. HTTP Digest-todennuksessa salasana muutetaan hash-muotoiseksi ennen sen lähettämistä palvelimelle. [21.]

Todennu ei ole sama asia kuin auktorisointi. Todennuksen kautta järjestelmä tietää kuka pyyntöjä tekee. Auktorisointi varmistaa, että kyseisellä käyttäjällä tai järjestelmällä on oikeus tehdä niitä pyyntöjä, joita hän yrittää tehdä. Auktorisointi siis edellyttää todennusta. [22.]

HTTP-protokollaan määriteltyjen todennusmenetelmien käyttäminen ei aina sovellu kaikkiin tilanteisiin. Tällainen tilanne voi tulla vastaan, kun joudutaan sallimaan palvelun todentua käyttäjän puolesta, jotta palveluiden välinen kommunikointi jatkuisi ilman käyttäjän suoranaista vaikuttamista ilman että tarvitsee antaa omia tunnuksia pyyntöjä tekevän palvelun käyttöön. Yksi tunnetuimmista menetelmistä on OAuth. Tämä avoin standardi perustuu poletteihin (token), joilla tunnistetaan palvelut ja käyttäjä. Poletti antaa palvelulle oikeuden toimia määritellyllä sivustolla tietyssä resurssissa ennalta rajatun ajan verran. [23.]

OAuthia on kritisoitu turhan monimutkaiseksi ja esimerkiksi Amazon Web Services, joka yksi maailman suurimmista ja käytetyimmistä Web API:sta, ei tue tätä ollenkaan. Amazonin todennus perustuu julkiseen käyttäjätunnisteeseen ja salaiseen avaimeen, jota käytetään ainoastaan HMAC-SHA1-salatun allekirjoituksen luomiseen. [24.]

Basic-todennus

Basic-todennuksen hyötyinä voidaan pitää sitä, että se on yksinkertainen ja kaikkialla tuettu. Basic-todennus ei kuitenkaan tarjoa yhtään suojaa, koska käyttäjänimi ja salasana lähetetään helposti palautettavassa muodossa. Käyttäjänimi ja salasana yhdistetään yhdeksi kaksoispisteellä erotetuksi merkkijonoksi. Tämä merkkijono enkoodataan Base64-järjestelmällä ja lähetetään Authorization-otsakkeessa. Basic-todennuksen ongelmat ratkeavat kuitenkin, kun käytetään TLS/SSL-salausprotokollaa. [25.]

Digest-todennus

Digest-todennus tarjoaa vahvemman suojauksen, koska salasanaa ei käytetä suoraan, vaan osana avaimen laskentaa. Kun asiakkaan tulee todentua, palvelin palauttaa WWW-Authenticate-otsakkeessa tiedot, joita käyttämällä asiakas voi rakentaa oikeellisen Authorization-otsakkeen. Digest-todennus on kuitenkin haavoittuva niin kutsutulle ”man-in-the-middle”-hyökkäykselle. Taulukoissa 8 ja 9 esitellään asiakkaan ja palvelimen otsakkeissa välittämät tiedot.

Taulukko 8. Digest-todennuksen palvelimen palauttamat tiedot WWW-Authenticate-otsakkeessa [26.]

Muuttuja	Kuvaus
realm	Kertoo mille alueelle ollaan kirjautumassa, yleensä järjestelmän lyhyt kuvaus.
qop	Quality Of Protection kertoo, kuinka hash tulee luoda. Arvo voi olla ”auth” tai ”auth-init”.
nonce	Yksilöllinen merkkijono, jota käytetään hashin luontiin ja lähetetään takaisin palvelimelle.
opaque	Voidaan käyttää istuntomuuttujana. Jos tämä muuttuu, selain kirjaa käyttäjän ulos.

Taulukko 9. Digest-todennuksen asiakkaan välittämät tiedot Authorization-otsakkeessa [26.]

Muuttuja	Kuvaus
username	Käyttäjänimi.
realm	Sama realm, minkä palvelin oli antanut asiakkaalle.

nonce	Sama nonce, minkä palvelin oli antanut asiakkaalle.
uri	Kirjautumisen vaativa URI.
qop	Sama Quality Of Protection, jonka palvelin oli antanut asiakkaalle.
nc	Nonce-count. Pyynnön laskurinumero. Jokaisen pyynnön yhteydessä numeroa tulisi kasvattaa.
cnonce	Yksilöllinen tunniste, jonka asiakas luo.

response	Varsinainen hash, jolla varmistetaan asiakkaan oikeellisuus. Esimerkkinä qop auth arvolla oikean hashin generointi seuraavanlainen (pseudokoodia):
	
	A1 = MD5(username:realm:password)
	A2 = MD5(request-method:uri) //request-method on käytettävä HTTP-metodi
	response = MD5(A1:nonce:nc:cnonce:qop:A2)
	
	Palvelin luo omilla tiedoilla vastaavan response-hashin ja tekee vertailun oikeellisuudesta.

5.6 Yksityisyys

Eri yksityisyysasetuksilla voidaan rajoittaa palvelussa käyttäjän tai käyttäjäryhmän näkyvyyttä ja mahdollisuutta vuorovaikutukseen toisten käyttäjien kanssa. Toteutus tukee kolmea eri vaihtoehtoa yksityisyydelle: ”Public”, ”Moderate” ja ”Private”. Vaihtoehdot ovat käytössä niin User-resurssille kuin UserGroup-resurssillekin ja ne on esitelty taulukossa 10.

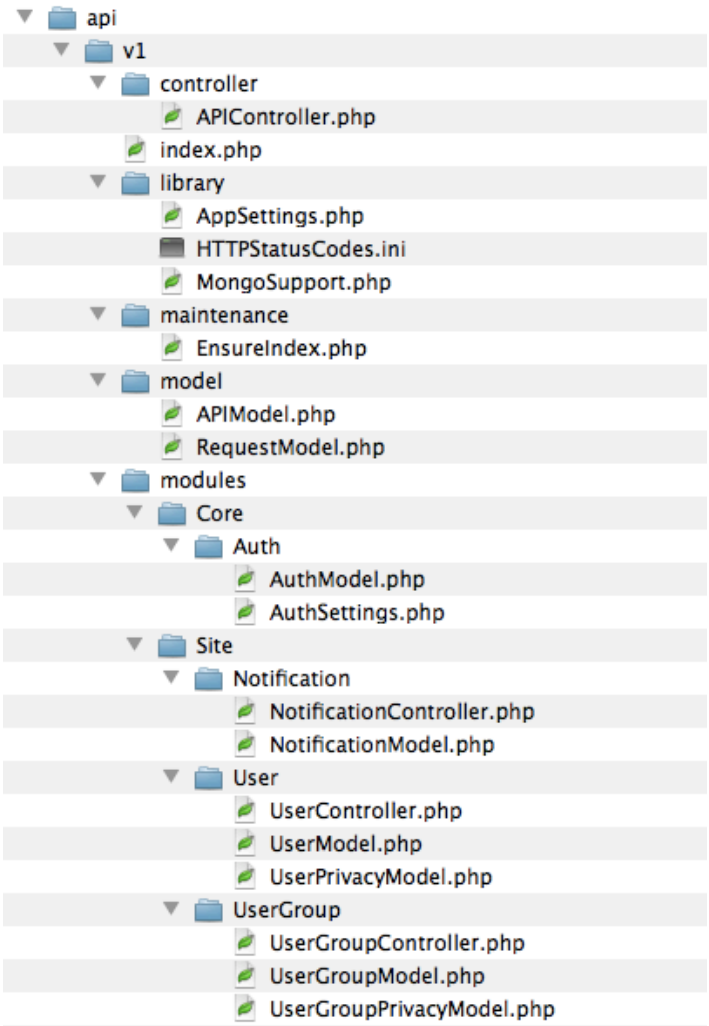
Taulukko 10. Yksityisyysasetukset User- ja UserGroup-resurssille

Muuttuja	Kuvaus
Public	Kaikki tieto näytetään muille käyttäjille. Käyttäjän tapauksessa API-avainta ja notifikaatioita ei näytetä millään asetuksella. Käyttäjä ja käyttäjäryhmä on löydettävissä. Käyttäjään tai käyttäjäryhmään tehty kontaktipyyntö hyväksytään automaattisesti.
Moderate	Pelkästään id ja nimi näytetään käyttäjille, jotka eivät ole käyttäjän tai käyttäjäryhmän kontakteja. Käyttäjä ja käyttäjäryhmä on löydettävissä. Käyttäjään tai käyttäjäryhmään tehty kontaktipyyntö vaatii omistajan hyväksynnän.
Private	Mitään ei näytetä muille käyttäjille, ellei käyttäjä ole käyttäjän tai käyttäjäryhmän kontakti. Käyttäjä ja käyttäjäryhmä ei ole löydettävissä. Käyttäjään tai käyttäjäryhmään ei voi tehdä kontaktipyyntöjä. Käyttäjä voi pyytää muita ei ”Private”-asetuksella varustettuja käyttäjiä kontakteiksiin. Ryhmän omistaja voi kutsua muita käyttäjiä ei ”Private”-asetuksella varustettuja käyttäjiä ryhmän kontakteiksi.

6 Toteutus

6.1 Kansio- ja tiedostohierarkia

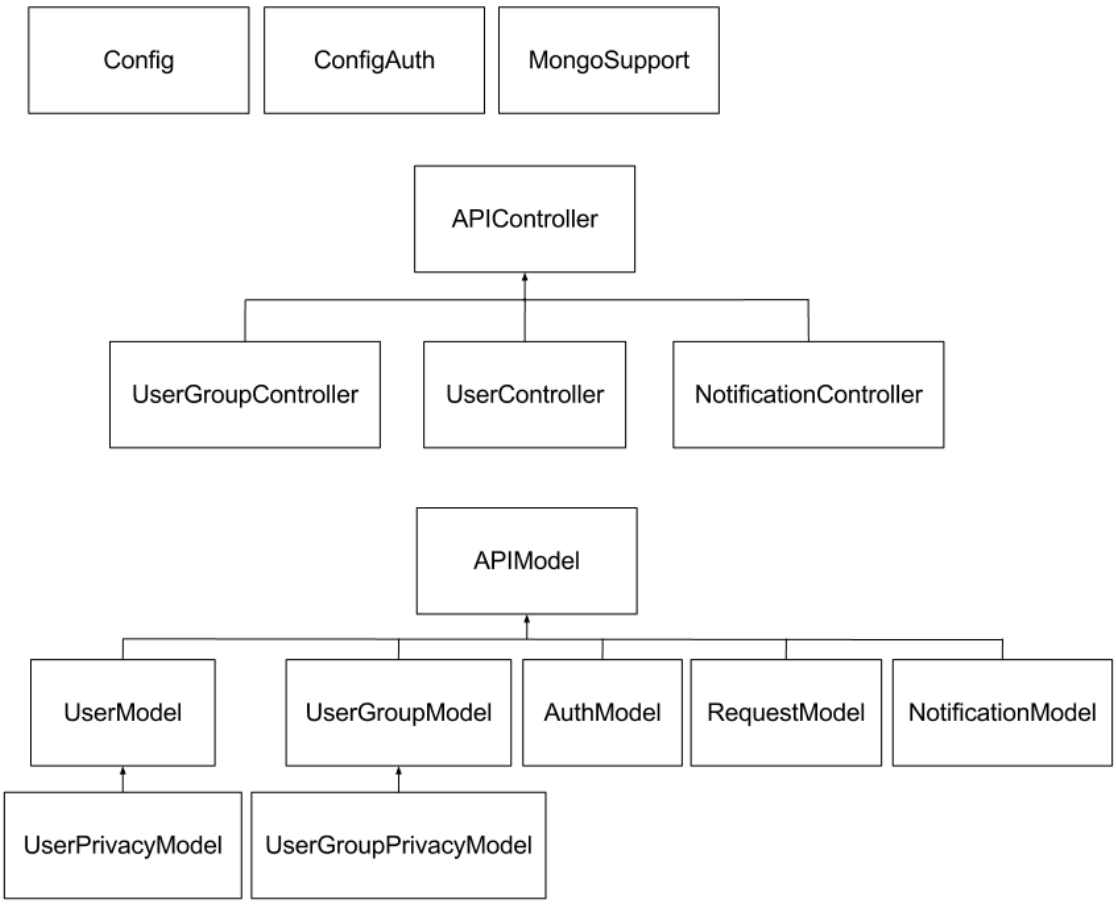
Kansio- ja tiedostohierarkia noudattaa MVC-mallia, joka esitetään kokonaisuudessaan kuviossa 13. Library-, Controller- ja Model-kansiot sisältävät palvelun yleisen toiminnan kannalta oleellisimmat tiedostot. Modules-kansioon on hajautettu varsinaista toiminnallisuutta. Core-kansio sisältää Web API –palvelun ydintoiminnallisuutta. Site-kansio sisältää palvelun käyttötapauksen moduuleja, joita voidaan helposti muutta tai tuottaa lisää.



Kuvio 13. Kansio- ja tiedostohierarkia.

6.2 Luokat

Toteutus koostuu useista luokista, joilla jokaisella on oma tehtävänsä palvelussa. Kuviossa 14 on esitelty toteutuksen luokkakaavio. Config-luokka sisältää palvelun kannalta oleellisia määriteltyjä vakioita. Yhtä lailla ConfigAuth-luokka pitää sisällään todennukseen liittyviä määritettäviä vakioita, joita asettamalla saadaan esimerkiksi haluttu todennusmenetelmä käyttöön. MongoSupport-luokassa toteutetaan tietokantayhteys ja tarvittaessa muutetaan MongoDB-olioita tietorakenteiksi, jotta ne ovat käytettävissä ohjelmassa.



Kuvio 14. Luokkakaavio.

APIController-luokka on toteutuksen pääohjain. Varsinaiset resurssimoduulit kyetään pitämään siisteinä niin että ne sisältävät pelkästään tarpeelliset metodit hyödyntämällä APIControlleria. Resurssikohtaiset ohjaimet UserController, UserGroupController ja NotificationController toteuttavat pääasiassa vain HTTP-metodeita vastaavat pyyntömeto-

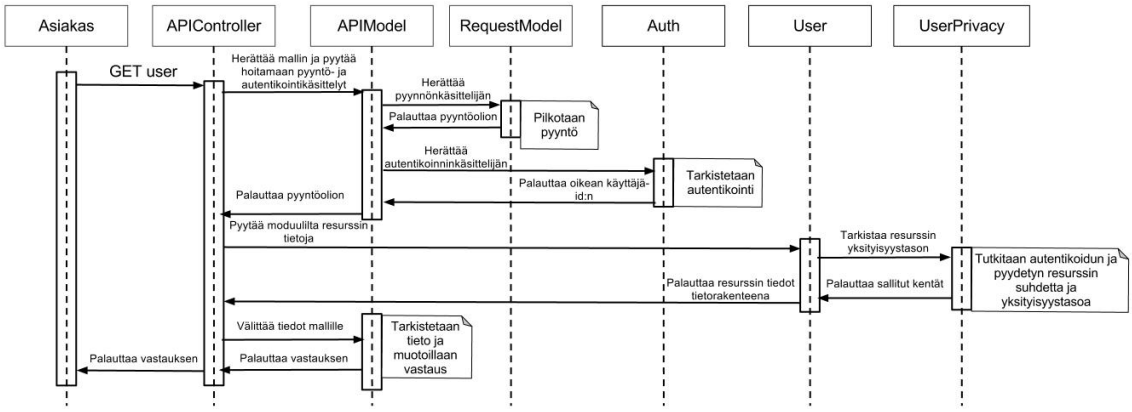
dit, jotta APIController osaa välittää pyynnön oikein mallille ja tehdä mahdollisesti pyyntöön liittyviä tarvittavia toimenpiteitä.

APIModel-luokka on toteutuksen päämalli. Luokka käsittelee RequestModel-luokan avulla pyynnön käsittelyn ja AuthModel-luokan avulla käyttäjien todennuksen. Resurssimoduuleissa UserModel, UserGroupModel ja NotificationModel tehdään varsinainen resurssikohtainen pyyntöjen tarkistus, käsittely, tietokantapyynnöt ja vastausten palautus. User- ja UserGroup-resursseihin GET- ja POST-metodeilla tehdyillä pyynnöillä pitää ottaa huomioon yksityisyysasetukset. Näiden tarkistaminen hoidetaan kunkin resurssin PrivacyModel-luokassa. APIModel-luokassa muodostetaan lopullinen oikeellinen vastaus, joka sisältää vastauskoodin, otsakkeet ja mahdollisen rungon.

6.3 Ohjelmakoodin työnkulku

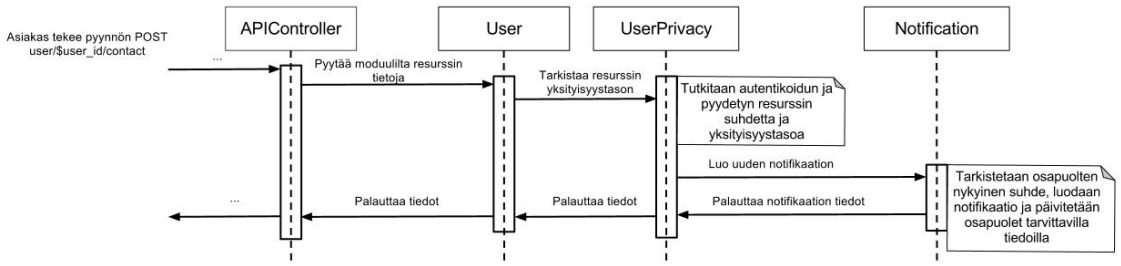
Asiakkaan pyyntö otetaan vastaan index.php-tiedostossa, eli pyyntö tulee http://$palvelin/api/v1/$resurssi-polkuun. Kuviossa 15 ja 16 esitellään ohjelmakoodin kulku vuokaavioina. Index.php:ssä luodaan uusi APIController, joka lataa palvelun kannalta kriittisen APIModelin, palveluasetukset ja rekisteröityjen moduulien ohjaimet. APIControllerin tehtävä on ottaa vastaan asiakkaan pyyntö, välittää se APIModelille jäsennettäväksi ja tämän perusteella siirtää pyyntö oikealle resurssinkäsittelijälle, pyytää APIModelia tarkistamaan todennus ja viimeiseksi palauttaa APIModelin oikeellisesti muokkaama vastaus asiakkaalle.

APIModel huolehtii HTTPStatusCodes.ini-tiedostossa sijaitsevien HTTP-vastauskoodien jäsentämisestä, lataa RequestModelin pyynnön jäsentämistä varten ja ydinmoduuleihin kuuluvan AuthModelin, jossa tarkistetaan todennus. Auth-moduulin asetuksissa voidaan määrittää, käytetäänkö Basic- vai Digest-todennusta. APIModel lataa rekisteröityjen moduulien mallit. APIModelin tärkein tehtävä on muodostaa oikeellinen vastaus, jonka APIController palauttaa asiakkaalle. APIController toimii moduuleille ”interfacena”. Moduulien ohjaimissa määritellään tuetut metodit ja annetaan APIControllerin hoitaa välitys moduulin mallille ja mallilta saadun vastauksen esikäsittely ja tarkistus ennen välittämistä APIModelille lopullista muodostamista varten.



Kuvio 15. Ohjelmakoodin esimerkkivuokaavio GET-pyynnöstä User-resurssiin.

Toteutuksen resurssimoduulit sisältävät ohjaimet ja mallit. User- ja UserGroup-resurssissa malli on vielä pilkottu yksityisyystarkistusten osalta omiin privacy-malleihinsa. Malleissa tehdään tarvittaessa tietokantakyselyt. Resurssien privacy-mallissa validatePrivacy-funktio tarkistaa pääsääntöisesti luonnin ja päivittämisen yhteydessä, että määritelty yksityisyysasetus on oikeellinen. CheckPrivacy-funktio tarkistaa GET-metodilla tehtyihin hakupyyntöihin palautettavat resurssin esitysmuodon ja POST-metodilla tehtyihin luontipyyntöihin uuden notifikaation luonnin tarpeellisuuden.



Kuvio 16. Ohjelmakoodin esimerkkivuokaavio POST-pyynnöstä yhden User-resurssin kontaktiksi. Kuviossa esitettynä kohta vain notifikaation luomisesta. Pyynnön käsittely ja autentikoinnin tarkistaminen jätetty pois, koska ne esiteltiin aiemmassa kuviossa.

Toteutuksessa oletetaan, että käyttäjäryhmiä tulee olemaan vähemmän kuin käyttäjiä, joten yksittäisen käyttäjän käyttäjäryhmät on helppo ja nopea käydä pyytämässä UserGroup-resurssilta. Kun User-resurssiin ei tallenneta käyttäjäryhmiä suoraan, säästytään ylimääräiseltä kirjanpidolta. Notification-resurssien määrän uskotaan olevan moninkertainen, joten ne on helppo sisällyttää User-resurssiin. User-resurssin kontaktit täytyy myös sisällyttää resurssiin itseensä tietorakenteena.

6.4 Palvelimen käyttöönotto

Toteutus tehtiin Linux-palvelimelle, jossa käyttöjärjestelmänä Ubuntu 4.4.3. Seuraavia ohjeita noudattamalla saadaan palvelin täysin toimintakuntoon palvelua varten. Ohjeet ovat kelvolliset myös yleisiksi ohjeiksi palvelimen käyttöönottoon Web-palvelun kehitystä varten.

6.4.1 Lighttpd:n ja PHP5-CGI:n asentaminen

```
1. $ apt-get install lighttpd
2. $ apt-get install php5-cgi
3. $ /etc/init.d/lighttpd start
4. $ /etc/init.d/lighttpd stop
```

Koodilistaus 11. Lighttpd:n ja php5-cgi:n asennus onnistuu helpoiten apt-gettiä käyttämällä.

Lighttpd käyttää FastCGI-rajapintaa. Koodilistauksessa 11 esitellään Lighttpd:n ja php5-cgi:n asentaminen. Tämä mahdollistaa ulkopuolisten sovellusten käytön Web-palvelimella. Sovellukset ajetaan erillisinä prosesseina. [27.]

Seuraavassa käydään läpi lighttpd Web-palvelimen asetusten määrittäminen ja testaus. Lighttpd:n asetukset voidaan testata koodilistauksessa 12 esitellyllä tavalla. [28.]

```
1. $ lighttpd -t -f lighttpd.conf
```

Koodilistaus 12. Lighttpd-asetusten testaus.

Jos tiedostoa ei löydy ja saa virheen: ”lighttpd.conf failed: No such file or directory”, tulee ajaa koodilistauksessa 13 esitelty rivi:

```
1. $ lighttpd -t -f /etc/lighttpd/lighttpd.conf
```

Koodilistaus 13. Luodaan uusi lighttpd.conf-tiedosto.

```
server.document-root = "/var/www/"
url.access-deny = ( "~", ".inc", ".ini" )
static-file.exclude-extensions = ( ".php", ".pl", ".fcgi", "~", ".inc" )
server.port = 8080
server.modules = (
                    "mod_access",
                    "mod_alias",
                    "mod_accesslog",
                    "mod_compress",
                    "mod_fastcgi",
                    "mod_rewrite",
        #     "mod_redirect",
        #     "mod_evhost",
        #     "mod_usertrack",
        #     "mod_rrdtool",
        #     "mod_webdav",
        #     "mod_expire",
        #     "mod_flv_streaming",
        #     "mod_evasive"
        )

#include FastCGI config-file
include "incl-fastcgi-php.conf"

#include mod_rewrite config-file
include "mod_rewrite.conf"

$HTTP["url"] =~ "^/community/" {
      dir-listing.activate = "disable"
}
```

Koodilistaus 14. lighttpd.conf-tiedostoon tehdään seuraavat lisäykset tai muutokset tarpeen mukaan.

Koodilistauksessa 14 on esiteltynä lighttpd:n pääasetukset. Koodilistauksissa 15 ja 16 include-tiedostojen asetukset.

```
fastcgi.debug = 0
fastcgi.map-extensions = ( ".php3" => ".php", ".php4" => ".php" )
fastcgi.server = ( ".php" =>
            (( "bin-path" => "/usr/bin/php-cgi",
                "socket" => "/tmp/php.socket",
                "bin-environment" => (
                    "PHP_FCGI_CHILDREN" => "16",
                    "PHP_FCGI_MAX_REQUESTS" => "10000"
                ),
                "bin-copy-environment" => ( "PATH", "SHELL", "USER" ),
                "broken-scriptfilename" => "enable"
            ))
        )
```

Koodilistaus 15. incl-fastcgi-php.conf-tiedosto luodaan ja lisätään seuraavat rivit.

```
url.rewrite-once
   = ("/community/api/(v\d)/(.*)" => "/community/api/$1/index.php/$2")
```

Koodilistaus 16. mod_rewrite.conf-luodaan ja lisätään seuraava rivi.

Lighttpd:n hyvä ominaisuus on include-tiedostojen ja -asetusten käyttö. Moduuleja voi ottaa käyttöön tai poistaa käytöstä tarpeen mukaan server.module-parametreilla. Tässä fastcgi-php-asetukset ladataan erillisestä tiedostosta. Omasta ympäristöstä tulee katsoa ”which php-cgi”-komennolla incl-fastcgi-php.conf-tiedoston ”bin-path”-attribuuttiin määritettävä polku. [29.]

Lighttpd:n mod_rewrite on laajennus, jolla voidaan uudelleenkirjoittaa URL:eja. Toteutuksessa ei haluta, että pyynnön tekijä joutuu määrittämään ”index.php”:n osana URL:ia, koska se ei suoranaisesti liity API:n eikä anna lisäarvoa pyynnölle. ”url.rewrite”-säännöllä lisätään ”index.php” sisäisesti ennen pyynnön välittämistä API:lle, jotta kaikki toimii niin kuin pitää. Lisäksi sääntö pitää huolen API-versioista. Jos API:sta tulee uusi versio, pätee sama uudelleenkirjoitus edelleen. [30.]

6.4.2 MongoDB:n asentaminen

MongoDB kannattaa asentaa omasta paketistaan. 10genin paketit ovat oletusarvoisesti tuoreempia, kuin Debianin tai Ubuntun varastoista löytyvät, joten koodilistauksissa 17, 18 ja 19 on esitelty tarvittavat toimenpiteet, jotta saadaan oikeat paketit asennettua. [31.]

```
1. $ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
```

Koodilistaus 17. Apt-key komenolla autentikoidaan paketti.

```
1. deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen
```

Koodilistaus 18. Lisätään rivi ”/etc/apt/sources.list”-tiedostoon.

```
1. $ apt-get update
2. $ apt-get install mongodb-10gen
3. $ apt-get install php5-dev php5-cli php-pear
4. $ pecl install mongo
```

Koodilistaus 19. Asennetaan MongoDB ja tarvittavat PHP-ajurit. [32.]

Koodilistauksessa 20 on esiteltynä MongoDB:n käyttöönotto asennuksen jälkeen. Esiteltyä riviä voi käyttää suoraan myös Apachen omassa PHP-asetustiedostossa.

```
1. extension=mongo.so
```

Koodilistaus 20. ”/etc/php5/cgi/php.ini”-tiedostoon lisätään rivi. Jos käytössä on Apache-palvelin voi saman rivin lisätä ”/etc/php5/apache2/php.ini”-tiedostoon.

6.4.3 Apache Subversionin asentaminen

Apache Subversion vaatii Apache-web-palvelimen. Svn-versioarkiston paikka yleensä on joko /srv/svn, /usr/local/svn tai /home/svn. Koodilistauksessa 21 on esiteltynä Apache Subversionin asentaminen.

```
1. $ apt-get install subversion
2. $ apt-get install subversion libapache2-svn
3. $ addgroup subversion
4. $ adduser matti subversion
5. $ adduser www-data subversion
```

Koodilistaus 21. Asennetaan subversionin tarvitsemat paketit ja määritellään ryhmä johon muutama käyttäjä lisätään.

6.5 Tietokannan indeksointi

Indeksit parantavat hakusuorituskykyä, usein dramaattisestikin. Nyrkkisääntönä voidaan pitää, että jokaista yksilöllistä tietokantahakua varten tarvitaan indeksi. Indeksi on tietorakenne, johon kerätään tietoa kokoelmassa olevien dokumenttien indeksoitavaksi määriteltyjen kenttien arvoista. MongoDB:n hakuoptimoija käyttää tätä tietorakennetta järjestämään dokumentit kokoelmassa. Oletusarvoisesti MongoDB luo indeksin ”_id”-kentälle, joten tästä kentästä ei tarvitse huolehtia. ”_id”-kentän indeksiä ei voi myöskään tuhota. [33.]

Jos indeksiä ei ole ja tietokantakysely suoritetaan käyttäen määritellyn kentän avainta, joutuu MongoDB käymään jokaisen dokumentin läpi tarkistaen kyselyssä käytetyn kentän avaimen arvon. Kun kokoelma on indeksoitu kentän avaimelle, ovat haut nopeita. Varsinaisten indeksien luonti MongoDB:ssä helppoa. MongoDB:n ensureIndex()-funktio luo indeksin vain, jos sitä ei ole jo olemassa. [33.]

Indeksien seurauksena kokoelmiin tehdyt lisäys- tai muokkaustoiminnot hidastuvat, koska MongoDB joutuu pitämään indeksit ajan tasalla. Tulee huomioida, että yleisesti suurin osa Web-sovellusten ja palveluiden aiheuttamista tietokantakyselyistä on hakupyyntöjä, joten indeksien käyttäminen on perusteltua ja suositeltavaa. Jos kuitenkin tiedetään, että kokoelmaan tehdään selkeästi enemmän kirjoitustoimintoja, voi indeksin käyttö olla epätarkoituksenmukaista.

Toteutuksessa ”maintenance”-kansiosta EnsureIndex.php-tiedosto tarkistaa että oikeat indeksit ovat olemassa. Säännöllisesti ajettavat houlto- ja tarkistustoimenpiteet on helppo suorittaa cronissa. Nämä perustuvat suoraan käytössä oleviin tietokantakyselyihin.

```
1.  $user->ensureIndex(array('email' => 1), $options);
2.  $user->ensureIndex(array('contact' => 1, 'privacy' => 1), $options);
3.  $user->ensureIndex(array('notification' => 1), $options);
```

Koodilistaus 22. User-resurssin indeksien olemassaolo varmistetaan ja tarvittaessa luodaan ensureIndex-metodilla.

User-resurssille luodaan kolme eri indeksiä, kuten koodilistauksesta 22 on esitelty. $user-muuttuja osoittaa MongoDB:n user-kokoelmaan ja muuttujassa $options määritellään MongoDB tarkistamaan indeksoinnin onnistuminen ja se, että indeksointi suoritetaan taustalla asynkronisesti. [34.]

```
1.  $usergroup-
    >ensureIndex(array('contact' => 1, 'owner_id' => 1, 'privacy' => 1), $options)
    ;
```

Koodilistaus 23. UserGroup-resurssin indeksien olemassaolo varmistetaan ja tarvittaessa luodaan ensureIndex-metodilla.

UserGroup-resurssille luodaan yksi indeksi, kuten koodilistauksesta 23 on esitelty. $usergroup-muuttuja osoittaa MongoDB:n usergroup-kokoelmaan ja muuttujassa $options määritellään MongoDB tarkistamaan indeksoinnin onnistuminen ja se, että indeksointi suoritetaan taustalla asynkronisesti. [34.]

```
1.  $notif-
    >ensureIndex(array('owner_id' => 1, 'target_id' => 1, 'resource_id' => 1, 'res
    ource' => 1), $options);
```

Koodilistaus 24. Notification-resurssin indeksien olemassaolo varmistetaan ja tarvittaessa luodaan ensureIndex-metodilla.

Notification-resurssille luodaan yksi indeksi, kuten koodilistauksessa 24 on esitelty. $notif-muuttuja osoittaa MongoDB:n notification-kokoelmaan ja muuttujassa $options määritellään MongoDB tarkistamaan indeksoinnin onnistuminen ja se, että indeksointi suoritetaan taustalla asynkroonisesti. [34.]

6.6 Projektin liittäminen Apache Subversioniin

Palvelun liittäminen Apache Subversioon on helppoa. Koodilistauksessa 25 esitellyissä komentoriviltä ajettavissa komennoissa ”community” on projektin nimi.

```
1. $ mkdir /home/svn/repos/community
2. $ svnadmin create /home/svn/repos/community
3. $ sudo chown -R www-data:subversion /home/svn/repos/community
4. $ sudo chmod -R g+rws /home/svn/repos/community
```

Koodilistaus 25. Komentoriviltä luodaan uusi subversion projekti.

Ennen koodilistauksessa 26 esitellyn rivin ajamista tulee mennä siihen kansioon, joka halutaan siirtää versioarkistoon. Siirto tehdään kansioon nimeltä ”trunk”.

```
1. $ svn import -m "initial import" . file:////home/svn/repos/community/trunk
```

Koodilistaus 26. Projekti siirretään versioarkistoon ”svn import”-komennolla.

Koodilistauksessa 27 pyydetään versioarkistosta itselle työkopio.

```
1. $ svn co file:////home/svn/repos/community/trunk community
```

Koodilistaus 27. Versioarkistosta pyydetään työkopio omalle koneelle ”svn co”-komennolla.

Kun muutoksia on tehty työkopioon, lähetetään muutokset palvelimelle koodilistauksessa 28 esitellyllä tavalla.

```
1.  $ svn commit -m "Tein muutoksia"
```

Koodilistaus 28. Omassa kopiossa tehdyt muutokset lähetetään versioarkistoon ”svn commit”-komennolla.

Projekti voidaan haarauttaa koodilistauksen 29 esittelemällä tavalla. Tämä on hyödyllistä, jos halutaan työstää eri korjauksia tai ominaisuuksia irrallaan päähaarasta.

```
1.  $ svn copy -m
    "Projektihaara" file:///home/svn/repos/community/trunk file:///home/svn/repos/
    community/branches/joku-branch
```

Koodilistaus 29. Projektin haarauttaminen omaksi ”branchiksi”. Esimerkissä ”joku-branch”

Jos halutaan tarjoilla versioarkistoa palvelimen ulkopuolelle, tulee määrittää svnserve-asetukset. Lisätään /home/svn/repos/community/conf/svnserve.conf-tiedostoon koodilistauksen 30 rivit:

```
1.  anon-access = none
2.  auth-access = write
3.  password-db = passwd
```

Koodilistaus 30. Svnservelle voidaan määrittää projektin käyttöoikeuksia ”svnserve.conf”-tiedostoon.

Asetukset määrittävät, että anonyymit käyttäjät eivät pääse käsiksi tähän projektiin, todennetuilla käyttäjillä on kirjoitusoikeus ja käyttäjänimi-salasanaparit sijaitsevat saman kansion passwd-tiedostossa. Tähän /home/svn/repos/community/conf/passwd-tiedostoon lisätään yksi tai useita haluttuja käyttäjänimi-salasanapareja koodilistaus 31:n mukaisesti.

```
1.  svn_user = svn_salasana
```

Koodilistaus 31. Projektin ”passwd”-tiedostoon määritellään käyttäjänimi-salasana-parit.

```
1.  $ svnserve –d
2.  $ svn list svn://localhost/community
```

Koodilistaus 32. Svnserve käynnistetään daemon-tilassa ja kokeillaan että kaikki toimii niin kuin pitää.

Koodilistauksessa 32 esitellään svnserven käynnistäminen ja testaus. Muut palvelimet käyttävät localhostin tilalla palvelimen tunnettua nimeä tai ip-osoitetta.

7 Jatkokehitys

Esiteltyä toteutusta voi laajentaa ja jatkokehittää useilta osa-alueilta. Kaikki arkaluontoinen tieto tulisi salata tietokannassa. Palveluun tulisi toteuttaa kyselyparametrien (query string) tuki. Näillä voidaan esimerkiksi rajoittaa yhden kyselyn tuottamaa kokoelma-vastauksen kokoa, järjestystä tai määrittää palvelukohtainen muuttuja, jos käyttäjiä ja käyttäjäryhmiä tullaan käyttämään täysin eri kokonaisuuksissa ja halutaan pitää tiedot selkeästi erillään. User-resurssilla voisi olla palvelun näkökulmasta eri oikeuksia, jotka määräävät, mitä pyyntöjä todennettu käyttäjä saa tehdä. UserGroup-resurssilla voisi olla useita omistajia, jotka pystyisivät hallinnoimaan samaa ryhmää. Jotta käyttäjäryhmät olisivat paremmin yrityskäytössä hyödynnettäviä tulisi ryhmien välinen hierarkia tarpeeseen. Tämä tosin voisi olla toisen palvelun määritettävissä oleva asia. Yksityisyysasetusmääritykset ovat tällä hetkellä melko hankalasti laajennettavissa yhdellä kerralla koko palvelun laajuudelta, joten ACL-tyylinen tai muu keskitetty ratkaisu voisi tulla tarpeeseen.

8 Yhteenveto

Insinöörityön tarkoituksena oli toteuttaa REST-arkkitehtuurimallia noudattava yhteisöpalvelun Web-ohjelmointirajapinta PHP-ohjelmointikielellä. Varsinaisen toteutuksen lisäksi työn tarkoituksena oli tutustua REST-arkkitehtuurimalliin sekä kaikkiin niihin palvelun ympäristön pystyttämisen kannalta oleellisiin vaiheisiin, jotka tulee palvelimelle tehdä. Insinöörityö vastaa sitä työtä, jota ohjelmistokehittäjä työssään tekee.

Insinöörityön toteutus selvensi REST-arkkitehtuurimallia ja yleisiä API-rajapinnan kehittämisen periaatteita. Toteutuksen rakenteet sopivat yleiseksi kehykseksi mille tahansa REST-arkkitehtuurimallia toteuttavalle Web-palvelulle. Yhteisöpalvelu-käyttötapaus ratkaisee useita palvelun kehittämisessä mahdollisesti vastaantulevia ongelmia ja tilanteita.

Oma palvelu kannattaa toteuttaa noudattamaan REST-arkkitehtuurimallia, jotta sen integrointi muihin palveluihin käy mahdollisimman helposti ja käyttö on läpinäkyvää. REST-arkkitehtuurimallia noudattavat palvelut toimivat muille palveluille itsenäisinä moduuleina, joten palvelukokonaisuuksienkin hallitseminen helpottuu ja tehostuu. Lisäksi palvelua voi muokata ja päivittää ilman, että muut palvelut tai kokonaisuus kärsii. Palvelun Web API voi olla julkisesti avoin tai näkyä vain esimerkiksi yrityksen sisäverkkoon, jolloin sen tietoturva- ja toiminnallisuusvaatimukset ovat selkeästi erilaiset.

Lähteet

1 Subbu Allamaraju, RESTful Web Services Cookbook. O'Reilly Media / Yahoo Press. Helmikuu 2010.

2 Abeysinghe Samisa, RESTful PHP Web Services. Packt Publishing. Lokakuu 2008.

3 The application/json Media Type for JavaScript Object Notation (JSON). 2012. Verkkodokumentti. The Internet Society. <http://www.ietf.org/rfc/rfc4627>. Luettu 13.3.2012.

4 List of HTTP header fields. 2012. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/List_of_HTTP_header_fields>. Luettu 13.3.2012.

5 Guidelines for Implementation of REST. 2012. Verkkodokumentti. National Security Agency. <http://www.nsa.gov/ia/_files/support/guidelines_implementation_rest.pdf>. Luettu 13.3.2012.

6 Status Code Definitions. 2012. Verkkodokumentti. Fielding, et al. <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>. Luettu 13.3.2012.

7 List of HTTP status codes. 2012. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/List_of_HTTP_status_codes>. Luettu 13.3.2012.

8 Understanding MVC in PHP. 2012. Verkkodokumentti. Joe Stump. <http://oreilly.com/php/archive/mvc-intro.html>. Luettu 13.3.2012.

9 PHP: Hypertext Preprocessor. 2012. Verkkodokumentti. The PHP Group. <http://www.php.net/>. Luettu 13.3.2012.

10 PHP Benefits - The benefits of PHP programming language. 2012. Verkkodokumentti. Rose India. <http://www.roseindia.net/tutorial/php/phpbeginning/php-benefits.html>. Luettu 13.3.2012.

11 PHP Introduction. 2012. Verkkodokumentti. W3Schools. <http://www.w3schools.com/php/php_intro.asp>. Luettu 13.3.2012.

12 Why PHP (and not Java)? 2012. Verkkodokumentti. Dries Buytaert. <http://buytaert.net/why-php-and-not-java>. Luettu 13.3.2012.

13 JSON vs. XML. 2012. Verkkodokumentti. Klint Finley. <http://www.readwriteweb.com/hack/2010/11/json-vs-xml.php>. Luettu 13.3.2012.

14 JSON: The Fat-Free Alternative to XML. 2012. Verkkodokumentti.. <http://www.json.org/xml.html>. Luettu 13.3.2012.

15 MongoDB. 2012. Verkkodokumentti. 10gen, Inc.<http://www.mongodb.org/>. Luettu 13.3.2012.

16 Mongo – The Interactive Shell. 2012. Verkkodokumentti. 10gen, Inc. <http://www.mongodb.org/display/DOCS/mongo+-+The+Interactive+Shell>. Luettu 13.3.2012.

17 Lighttpd. 2012. Verkkodokumentti.. <http://www.lighttpd.net/>. Luettu 13.3.2012.

18 Lighttpd. 2012. Verkkodokumentti. Linux.fi. <http://linux.fi/wiki/Lighttpd>. Luettu 13.3.2012.

19 Why Subversion Still Beats Git. 2012. Verkkodokumentti. Brandon Savage. <http://www.brandonsavage.net/why-subversion-still-beats-git/>. Luettu 13.3.2012.

20 What does SVN do better than git? 2012. Verkkodokumentti. Stack Exchange Inc. < http://programmers.stackexchange.com/questions/111633/what-does-svn-do-better-than-git>. Luettu 13.3.2012.

21 Digest access authentication. 2012. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Digest_access_authentication>. Luettu 13.3.2012.

22 Authentication. 2012. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Authentication#Authentication_vs._authorization>. Luettu 13.3.2012.

23 Moving from Basic Auth to OAuth. 2012. Verkkodokumentti.. <https://dev.twitter.com/docs/auth/moving-from-basic-auth-to-oauth>. Luettu 13.3.2012.

24 Amazon Fulfillment Web Service – Developers Guide (Version 1.1). 2012. Verkkodokumentti. Amazon Web Services LLC. <http://docs.amazonwebservices.com/fws/latest/DeveloperGuide/index.html?RequestAuthenticationArticle.html>. Luettu 13.3.2012.

25 Basic access authentication. 2012. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Basic_access_authentication>. Luettu 13.3.2012.

26 HTTP Basic and Digest authentication with PHP. 2012. Verkkodokumentti. Xavier. <http://api-meal.eu/memo/149-http-basic-and-digest-authentication-with-php/>. Luettu 13.3.2012.

27 The FastCGI Interface. 2012. Verkkodokumentti. Jean-Philippe Lang. <http://redmine.lighttpd.net/projects/1/wiki/Docs:ModFastCGI>. Luettu 13.3.2012.

28 Configuration. 2012. Verkkodokumentti. Jean-Philippe Lang. <http://redmine.lighttpd.net/projects/lighttpd/wiki/TutorialConfiguration>. Luettu 13.3.2012.

29 Simplify your configfiles with includes. 2012. Verkkodokumentti. Jan. <http://blog.lighttpd.net/articles/2005/11/25/simplify-your-configfiles-with-includes>. Luettu 13.3.2012.

30 Mod rewrite. 2012. Verkkodokumentti. Jean-Philippe Lang. <http://redmine.lighttpd.net/projects/4/wiki/Mod_rewrite>. Luettu 13.3.2012.

31 Ubuntu and Debian packages. 2012. Verkkodokumentti. 10gen, Inc. <http://www.mongodb.org/display/DOCS/Ubuntu+and+Debian+packages>. Luettu 13.3.2012.

32 PHP Language Center. 2012. Verkkodokumentti. 10gen, Inc. <http://www.mongodb.org/display/DOCS/PHP+Language+Center>. Luettu 13.3.2012.

33 Indexes. 2012. Verkkodokumentti. 10gen, Inc. <http://www.mongodb.org/display/DOCS/Indexes>. Luettu 13.3.2012.

34 MongoCollection:ensureIndex. 2012. Verkkodokumentti. The PHP Group. <http://php.net/manual/en/mongocollection.ensureindex.php>. Luettu 13.3.2012.

Tuetut pyynnöt ja pyyntöjen sekä vastausten rungot

User-resurssi

URI	Metodi	Pyynnön runko	Vastauksen runko
/user	GET	-	{"data": [{"_id": "$user_id", "name": "$nimi"}, {..}]}
/user	POST	{"name": "$nimi", "email": "$email", "privacy": "moderate"}	{"data": [{"_id": "$user_id", "api_key": "$api_key", "email": "$email", "name": "$nimi", "contact": [], "privacy": "moderate", "notification": []}]}
/user/$user_id	GET	-	{"data": [{"_id": "$user_id", "api_key": "$api_key", "email": "$email", "name": "nimi", "contact": [], "privacy": "moderate", "notification": []}]}
/user/$user_id	PUT	{"name": "$nimi", "email": "$email", "privacy": "moderate"}	{"data": [{"_id": "$user_id", "api_key": "$api_key", "email": "$email", "name": "$nimi", "contact": [], "privacy": "moderate", "notification": []}]}
/user/$user_id	DELETE	-	-
/user/$user_id	HEAD	-	-
/user/$user_id/contact	GET	-	{"data": [{"_id": "$user_id", "contact": ["$user_id1", "..."]}]}
/user/$user_id/contact	POST	-	{"data": [{"_id": "$notification_id", "owner_id": "$user_id", "resource": "user", "resource_id": "$user_id1", "target_id": "$user_id1", "status": "waiting", "status_message": "Request to resource user is waiting response."}]}
/user/$user_id/contact	DELETE	-	-
/user/$user_id/notification	GET	-	{”data”: [{”_id”: ”$user_id”, ”notification”: [”$notification_id”, ”...”]}]}
/user/$user_id/usergroup	GET	-	{"data": [{"_id": "$user_id", "usergroup": ["$usergroup_id", "..."]}]}

/user/$user_id/api_key	GET	-	{"data": [{"_id": "$user_id", "api_key": "$api_key"}]}
/user/$user_id/api_key	POST	-	{"data": [{"_id": "$user_id", "api_key": "$new_api_key"}]}

UserGroup-resurssi

URI	Metodi	Pyynnön runko	Vastauksen runko
/usergroup	GET	-	{"data": [{"_id": "$usergroup_id", "name": "$nimi"}, {..}]}
/usergroup	POST	{"name": "$nimi", "privacy": "moderate"}	{"data": [{"_id": "$usergroup_id", "owner_id": "$user_id", "name": "$nimi", "contact": ["$user_id", "..."], "privacy": "moderate"}]}
/usergroup/$usergroup_id	GET	-	{"data": [{"_id": "$usergroup_id", "owner_id": "$user_id", "name": "$nimi", "contact": ["$user_id", "..."], "privacy": "moderate"}]}
/usergroup/$usergroup_id	PUT	{"name": "$nimi", "privacy": "moderate"}	{"data": [{"_id": "$usergroup_id", "owner_id": "$user_id", "name": "$nimi", "contact": ["$user_id", "..."], "privacy": "moderate"}]}
/usergroup/$usergroup_id	DELETE	-	-
/usergroup/$usergroup_id	HEAD	-	-
/usergroup/$usergroup_id /contact	GET	-	{"data": [{"_id": "$usergroup_id", "contact": ["$user_id", "..."]}]}
/usergroup/$usergroup_id /contact	POST	-	{"data": [{"_id": "$usergroup_id", "owner_id": "$user_id", "resource": "usergroup", "resource_id": "$usergroup_id", "target_id": "$user_id", "status": "waiting", "status_message": "Request to resource usergroup is waiting response."}]}
/usergroup/$usergroup_id /contact	DELETE	-	-
/usergroup/$usergroup_id	DELETE	-	-

/contact/$user_id			

Notification-resurssi

URI	Metodi	Pyynnön runko	Vastauksen runko
/notification	GET	-	{"data": [{"_id": "$notification_id", "resource": "user"}, {...}]}
/notification/$notification_id	GET	-	{"data": [{"_id": "$notification_id", "owner_id": "$user_id", "resource": "user", "resource_id": "$user_id1", "target_id": "$user_id1", "status": "waiting", "status_message": "Request to resource user is waiting response."}]}
/notification/$notification_id	POST	{"status": "accepted"}	{"data": [{"_id": "$notification_id", "owner_id": "$user_id", "resource": "user", "resource_id": "$user_id1", "target_id": "$user_id1", "status": "accepted", "status_message": "Request to resource user has been accepted."}]}
/notification/$notification_id	DELETE	-	-
/notification/$notification_id	HEAD	-	-