Jukka Pelto-aho

PATENTTI- JA REKISTERIHALLITUKSEN SÄHKÖISEN ASIAKIRJAPALVELUN TOTEUTTAMINEN

Opinnäytetyö
CENTRIA AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Kesäkuu 2015

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Yksikkö Kokkola-Pietarsaari	**Aika** Kesäkuu 2015	**Tekijä/tekijät** Jukka Pelto-aho
Koulutusohjelma Tietotekniikan koulutusohjelma		
Työn nimi PATENTTI- JA REKISTERIHALLITUKSEN SÄHKÖISEN ASIAKIRJAPALVELUN TOTEUTTAMINEN		
Työn ohjaaja Sakari Männistö		**Sivumäärä** 32
Työelämäohjaaja Timo Sorsamäki		

Patentti- ja rekisterihallitus (PRH) on työ- ja elinkeinoministeriön alainen virasto. Sen tehtäviin kuuluu muun muassa patenttihakemusten käsittely, patenttien myöntäminen ja niiden rekisteröinti.

Patenttien hakemiseen kuuluu paljon kirjeenvaihtoa hallinnon ja hakijan välillä. Patenttihakemusten käsittelyprosessi tapahtuu pitkälti sähköisesti hallinnon sisäisessä toiminnassa. Hakemuksiin liittyvä kirjeenvaihto hallinnon ja hakijan välillä tapahtui aikaisemmin vain paperikirjeiden välityksellä. Erityisesti asiamiesten ja isompien yritysten taholta oli esitetty toiveita saada kirjeet sähköisessä muodossa. Erityisesti haluttiin, että kirjeet voitaisiin siirtää automaattisesti asiakkaiden omiin sähköisiin järjestelmiin. Aikaisemmin asiakkaat joutuivat ensin skannaamaan ja syöttämään saapuneet paperikirjeet omiin tietojärjestelmiinsä. Oli tarve ratkaista asia tarjoamalla sähköinen rajapinta asiakkaille asiakirjojen hakemista varten.

Alfame Systems Oy on noin 30 henkilöä työllistävä kokkolalainen ohjelmistoalan yritys. Patentti- ja rekisterihallitus on jo pitkään ollut Alfame Systems Oy:n asiakas. Alfame Systems toteutti PRH:n tarvitseman palvelun.

Tässä työssä tutustuttiin REST-arkkitehtuurityyliin ja sen etuihin. Asiakkaan tarvitsema palvelu määritettiin ja toteutettiin. Raportissa kuvataan REST-arkkitehtuurin periaatteet sekä ohjelmistokomponentit millä käytännön toteutus tehtiin.

Asiasanat
HTTP, Java, REST, Spring framework, Web Services

Centria
UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Unit	Date	Author/s
Kokkola-Pietarsaari	June 2015	Jukka Pelto-aho
Degree programme Information technology		
Name of thesis FINNISH PATENT AND REGISTRATION OFFICE, IMPLEMENTING AN ELECTRONIC SERVICE FOR PATENT CORRESPONDENCE		
Instructor Sakari Männistö		**Pages** 32
Supervisor Timo Sorsamäki		

The Finnish Patent and Registration Office (PRH) is under the administrative branch of the Ministry of Employment and the Economy. Its responsibilities include handling patent applications, granting patents and maintaining a registry of patents.

During the patent application process there is a considerable amount of correspondence between PRH and the patent applicant. PRH internally processes applications using electronic services. The correspondence between the PRH and the applicant used to be conducted using traditional letters. Applicants that made numerous patent applications (companies and law firms) had requested for an electronic service for handling correspondence. There was a need for a service where the applicants' own electronic service could download the correspondence. Previously the applicants had to manually scan and upload the documents into their own systems. There was a need to solve the situation by offering an electronic interface for the clients to retrieve the documents.

Alfame Systems Oy is an information technology company based in Kokkola. It employs about 30 employees. The Finnish Patent and Registration Office had used the services of Alfame Systems Oy for quite some time. Alfame implemented the required new service.

During this thesis work the REST architecture style and its benefits were studied. The electronic service needed by the client was defined and implemented. This thesis work describes the REST architecture style and the software components which were used to implement the service.

Key words
HTTP, Java, REST, Spring framework, Web Services

TIIVISTELMÄ
ABSTRACT
SISÄLLYS

1 JOHDANTO 1
2 REST OHJELMISTOARKKITEHTUURI 3
2.1 Yleistä 3
2.2 Asiakas-Palvelin -rajoite (Client-Server) 3
2.3 Tilaton-rajoite (Stateless) 4
2.4 Välimuisti-rajoite (Cache) 5
2.5 Yhdenmukainen rajapinta -rajoite (Uniform Interface) 6
2.6 Kerroksittainen järjestelmä -rajoite (Layered System) 8
2.7 Suoritettavan koodin lähettäminen -rajoite (Code-on-demand) 9
2.8 REST ohjelmistoarkkitehtuurin erot RPC pohjaisiin web rajapintoihin 9
3 ASIAKIRJAPALVELUN KUVAUS 11
3.1 Toteuttajan ja Asiakkaan lyhyt kuvaus 11
3.1.1 Alfame Systems Oy 11
3.1.2 Patentti- ja rekisterihallitus 11
3.2 Toteutettavan palvelun kuvaus 12
3.2.1 Lähtökohta 12
3.2.2 Tavoite 12
3.2.3 Palvelun toiminnot 13
4 TOTEUTUS 15
4.1 Toteutusprosessin lyhyt kuvaus 15
4.2 Toteutuksen arkkitehtuuri ja käytetyt tekniikat 17
4.3 Hakurajapinnan toteutus (yhden rajapinnan toteutus esimerkkinä) 19
4.4 Tietovarasta- ja logiikkakerrosten toteutus 25
5 YHTEENVETO 31

LÄHTEET

1 JOHDANTO

Tässä opinnäytetyössä toteutettiin Patentti- ja rekisterihallinnolle sähköinen rajapinta patenttien kirjeenvaihdon välittämiseen. Työ toteutettiin käyttämällä REST-sovellusarkkitehtuuria.

Patentti- ja rekisterihallinto hoitaa mm. patenttihakemuksiin liittyvät toimenpiteet. Patenttia voi hakea joko yksityinen henkilö tai yritys. Yksityishenkilö tai yritys voi myös antaa patentin haun asiamiehen hoidettavaksi. Asiamiehenä tyypillisesti toimii patenttiasioihin keskittynyt lakiyritys.

Patenttien hakemiseen kuuluu paljon kirjeenvaihtoa hallinnon ja hakijan välillä. Patenttihakemusten käsittelyprosessi tapahtuu pitkälti sähköisesti hallinnon sisäisessä toiminnassa. Hakemuksiin liittyvä kirjeenvaihto hallinnon ja hakijan välillä tapahtui aikaisemmin vain paperikirjeiden välityksellä. Erityisesti asiamiesten ja isompien yritysten taholta oli esitetty toiveita saada kirjeet sähköisessä muodossa. Erityisesti haluttiin, että kirjeet voitaisiin siirtää automaattisesti asiamiesten omiin sähköisiin järjestelmiin. Aikaisemmin asiamiehille saapuneet paperikirjeet täytyi ensin skannata ja syöttää tietojärjestelmään.

Työssä toteutettiin tietojärjestelmä, joka tarjoaa rajapinnan, johon asiamiehet voisivat integroida omia tietojärjestelmiään. Tavoitteena oli mahdollistaa hallinnon kirjeiden siirtyminen asiamiesten järjestelmiin automaattisesti sähköisessä muodossa. Rajapinnan kautta voidaan pyytää luettelo hakijalle kuuluvista kirjeistä ja ladata kopio kirjeestä.

Toteutuksessa päätettiin käyttää REST-arkkitehtuuria. Java-ohjelmointikielen lisäksi käytettiin Spring-kehystä, Hibernate ORM -työkalua sekä RestEasyä.

Työ tehtiin työnantajani Alfame Systems Oy:n normaalina asiakkaalle tehtävänä sovellusprojektina. Alfame on kokkolalainen ohjelmistoalan yritys. Alfame tuottaa asiakkailleen räätälöityjä tietojärjestelmiä.

Tässä opinnäytetyössä kuvataan toteutuksessa käytettyä ketterää menetelmää, REST-arkkitehtuurin periaatteita sekä ohjelmiston perusarkkitehtuuria. Asiakkaalle tehdyn projektin yhteydessä toteutettiin REST-rajapinnan lisäksi myös selainpohjainen sovellus, mutta se rajattiin tämän lopputyön ulkopuolelle.

2 REST OHJELMISTOARKKITEHTUURI

2.1 Yleistä

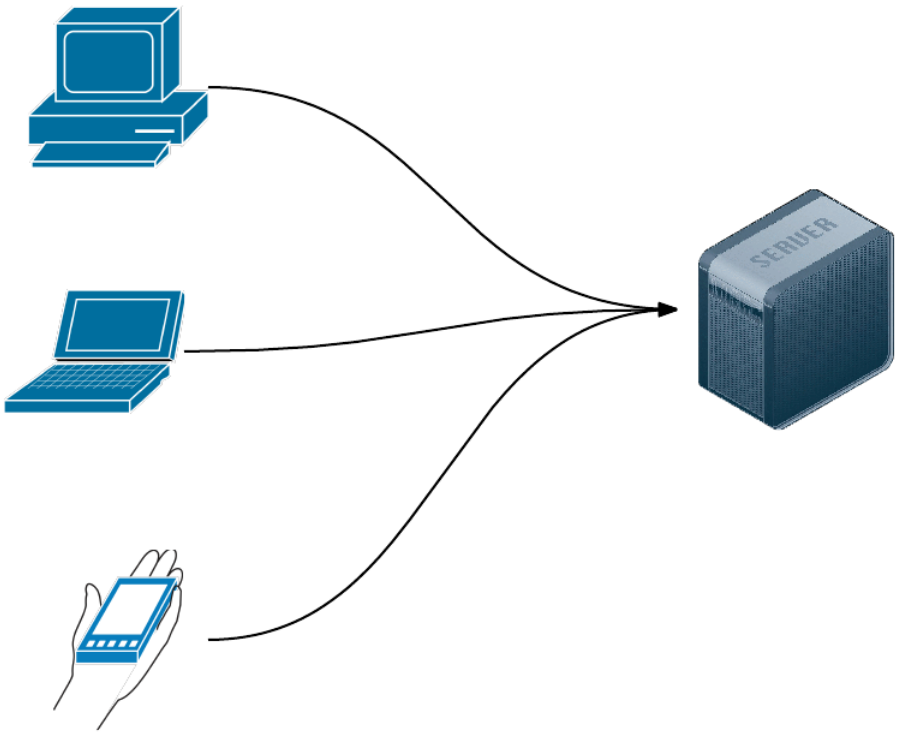
Roy T. Fielding esitteli väitöskirjatyössään kehittämänsä termin REST, joka on lyhenne sanoista REpresentational State Transfer (Fielding 2000, 76). Kyseessä on sovellusarkkitehtuurityyli hajautettujen hypermediajärjestelmien rakentamiseen. Fielding oli mukana määrittelemässä internetin HTTP 1.0 -protokollaa vuoden 1995 alussa. Internetin kehittyessä oli tarve päivittää HTTP-protokollaa, ja tässä yhteydessä Fielding halusi määrittää ja kuvata internetin korkeamman tason arkkitehtuuria. Internetistä löytyy joukko standardeja, jotka kuvaavat hyvin tarkalla tasolla kunkin yksittäisen toiminnon toimintaa. Hyvää korkeamman tason kuvausta ei kuitenkaan ollut. Fielding halusi korjata tämän puutteen, jotta protokollien päivityksen yhteydessä voitiin keskittyä parantamaan heikkoja kohtia samalla pitäen kiinni vahvuuksista.

Internetin ominaisuuksia ovat hajautettavuus ja skaalautuvuus. Toteutettaessa ohjelmistojen välisiä web-rajapintoja REST-arkkitehtuurityylin mukaisesti, vastaavat ominaisuudet on mahdollista saavuttaa. Se vaatii pitäytymistä tiettyihin REST-sovellusarkkitehtuurin rajoitteisiin, joilla rajapintoja rakennetaan. Seuraavassa luvussa kuvataan kyseiset rajoitteet lyhyesti.

2.2 Asiakas-Palvelin -rajoite (Client-Server)

Asiakas-Palvelin -rajoitteessa ohjelman toiminto jakaantuu palvelimessa ja asiakas-koneessa tapahtuviin toimintoihin. Web-palvelimen ja asiakkaan selaimen välinen työnjako on tyyppiesimerkki tästä. Palvelimella ajettava ohjelmisto huolehtii tiedon tallentamisesta ja käsittelystä sekä asiakkaan selaimelle siirrettävän HTML-koodin muodostamisesta. Asiakkaan selain puolestaan huolehtii vastaanotetun HTML-koodin esittämisestä kyseisellä päätelaitteella. (Fielding 2000, 78.)

Tämä työnjako tarjoaa useita etuja. Ensinnäkin on selvästi eroteltu vastuut. Tämä yksinkertaistaa molempien ohjelmiston rakennetta. Lisäksi, mikäli asiakkaan ja palvelimen väliset rajapinnat eivät muutu, molempia ohjelmistoja voidaan kehittää toisistaan riippumatta.



KUVIO 1. Asiakas-palvelin -malli

2.3 Tilaton-rajoite (Stateless)

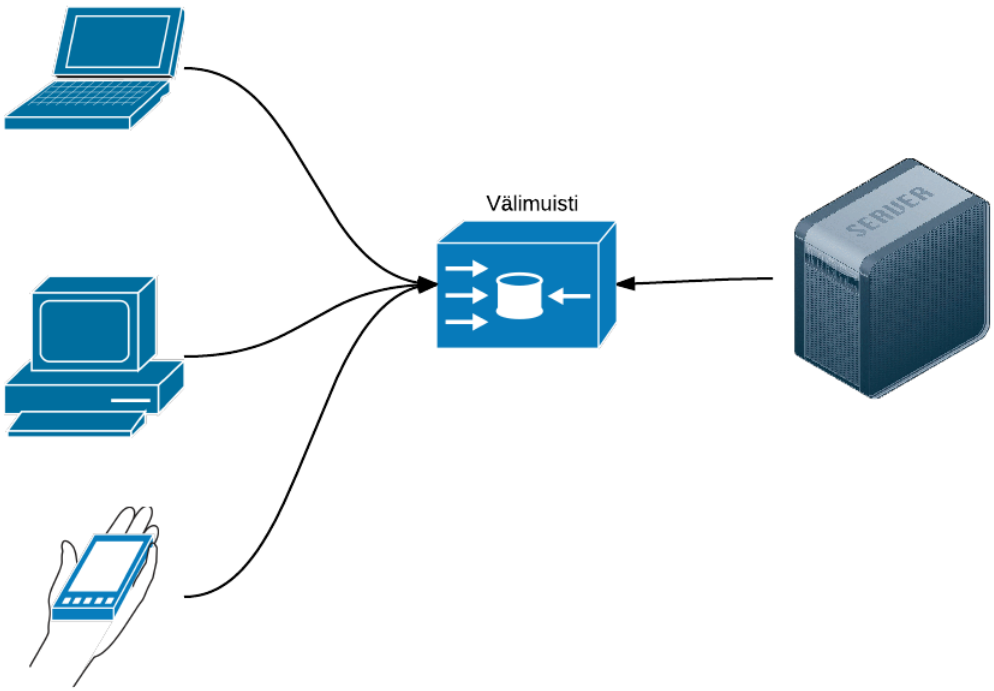
Tilattomuus tarkoittaa sitä, että kutsut asiakkaalta palvelimelle voidaan palvelimella tulkita pelkästään kutsun sisällön perusteella. Vastauksen muodostamiseen palvelimen ei tarvitse hakea tietoa esim. asiakkaalta aikaisemmin tulleista kutsuista.

Tämä parantaa sovellusten luotettavuutta ja skaalautuvuutta. Luotettavuus paranee, koska palvelin sovellus sietää paremmin tilapäisiä virhetilanteita. Virhe saattaisi

esim. pyyhkiä tiedot asiakkaan aikaisemmista kutsuista, mutta tilattomassa kutsussa on itsessään kaikki tarvittava tieto, joten virhe ei haittaa palvelinsovelluksen toimintaa. Skaalautuvuus paranee, koska palvelimen ei tarvitse tallentaa asiakkaan tilaa kutsujen välillä. Tämä vaatii vähemmän resursseja palvelimelta sekä yksinkertaistaa palvelinohjelmiston rakennetta. (Fielding 2000, 78.)

2.4 Välimuisti-rajoite (Cache)

Välimuisti-rajoite määrittelee, että kunkin palvelimen antaman vastauksen tulee sisältää tieto siitä, voidaanko vastaus tallentaa välimuistiin. Asiakkaan ja palvelimen välissä voi sijaita välimuisti. Mikäli asiakas hakee palvelimelta tietoa, johon vastaus löytyykin jo välimuistista, voidaan vastaus antaa sieltä, ja tällöin kutsua palvelimelle asti ei välity. Tämä vähentää verkon liikennettä, pienentää vastausten viiveitä ja parantaa skaalautuvuutta. (Fielding 2000, 79–80.)



KUVIO 2. Välimuisti asiakkaan ja palvelimen välissä

2.5 Yhdenmukainen rajapinta -rajoite (Uniform Interface)

Yksi REST-arkkitehtuurin keskeisiä periaatteita on yhdenmukaisten rajapintojen toteuttaminen ja käyttäminen komponenttien välillä. Tämä yksinkertaistaa järjestelmän kokonaisarkkitehtuuria. Yhdenmukainen rajapinta myös lisää komponenttien välisen kommunikoinnin ymmärrettävyyttä. Kun kukin kutsu ja vastaus pysyttäytyvät tietyissä rajoitteissa, sovelluksen toimintaa pystyy paremmin ymmärtämään. (Fielding 2000, 81.)

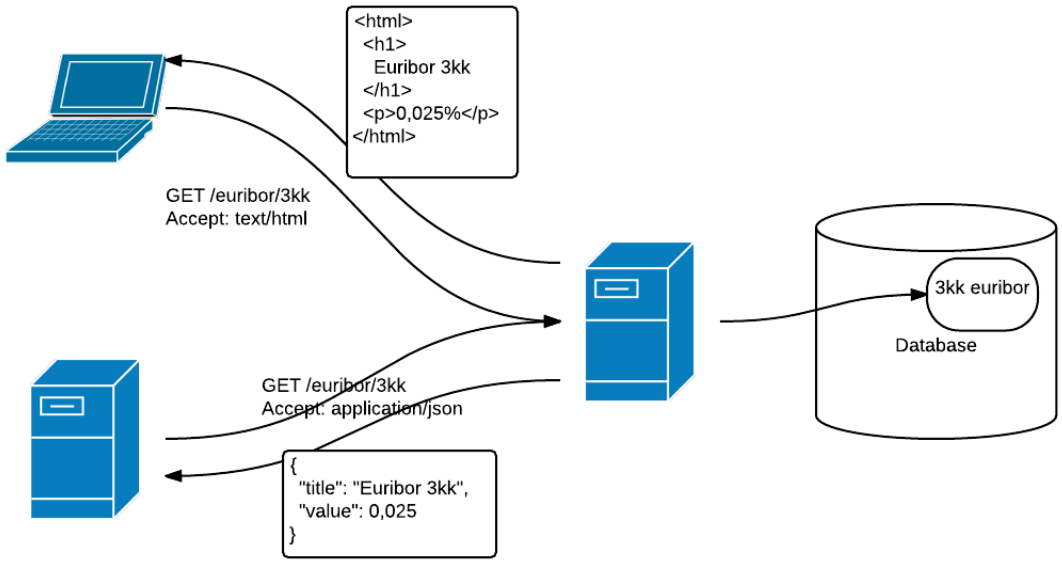
Huonona puolena yhdenmukaisen rajapinnan toteuttamisessa on se, että sovellus menettää hieman tehokkuuttaan. Kommunikointi voitaisiin toteuttaa kunkin toiminnon tarpeiden mukaan mahdollisimman tehokkaasti, mutta tällöin järjestelmä olisi vaikeampi ymmärtää, ja jouduttaisiin etsimään toisia tapoja mm. vikasietoisuuden ja skaalautuvuuden toteuttamiseksi. (Fielding 2000, 82.)

Yhdenmukainen rajapinta vaatii tiettyjen lisärajoitteiden noudattamista (Fielding 2000, 82). Rajoitteet ovat resurssien tunnistaminen, resurssien käsittely esitysten (representations) kautta, itseselitteiset viestit sekä hypermedian käyttö ilmaisemaan rajapinnan tilan muutosmahdollisuudet (Allamaraju 2010, 261-263).

Resurssi on mikä tahansa asia, mihin voidaan viitata yksilöllisellä tunnisteella. Se voi olla esim. teksti tai kuva. Resurssin sisältö voi muuttua ajankohdasta toiseen, vaikka tunniste pysyykin samana (esim. päivän sää).

Resurssi voidaan kuvata useammalla kuin yhdellä tavalla. Esimerkiksi samalla resurssitunnisteella oleva tieto voidaan esittää XML-muodossa tai HTML-muodossa. Resurssiin liitetään metadatana tieto, mikä kuvaa, missä esitysmuodossa ko. resurssi on kuvattu. Tällöin resurssi on vain yksi esitys

(representation) taustalla olevasta resurssista. REST-sovellusarkkitehtuurissa voidaan esittää toiveita, missä muodossa resurssi toimitetaan (html, xml, json yms.). Täten resursseja käsitellään aina valitun esitysten kautta (Allamaraju 2010, 262).



KUVIO 3. Sama resurssi esitetään asiakkaan toivomassa esitysmuodossa

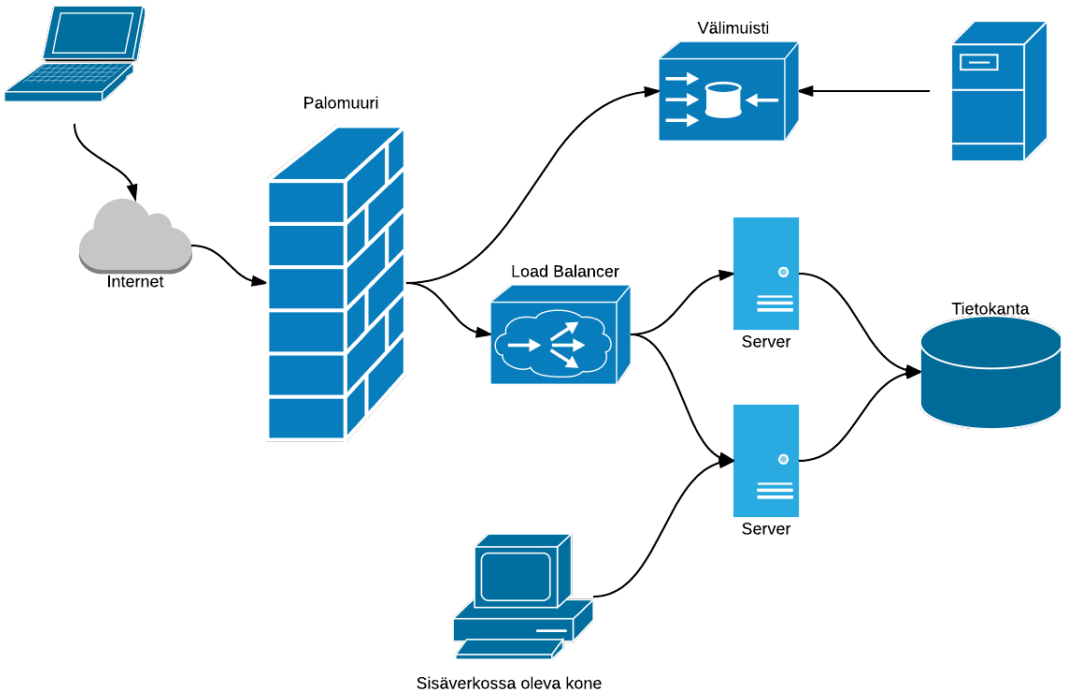
Itseselitteiset viestit ovat viestejä, jotka sisältävät kaiken tarvittavan tiedon viestin käsittelyyn. Viesti sisältää tiedon lisäksi myös metatietoa, jossa tyypillisesti kerrotaan ainakin, mitä tietotyyppiä viesti on (XML,JSON yms.) sekä voiko tiedon esimerkiksi tallentaa välimuistiin.

Jokainen vastaus, jonka asiakas saa palvelimelta, kuvaa asiakkaalle näkyvän sovelluksen tilaa. Jotta asiakas voisi muuttaa tilaa, täytyy hänen tietää minkälaisia kutsuja sovellukselle voi lähettää. REST-sovellusarkkitehtuurin yhtenä rajoitteena on hypermedian käyttö. Hypermediassa välitetään asiakkaalle sisällön mukana elementtejä, jotka kuvaavat asiakkaalle tilan muutosmahdollisuudet. Esimerkiksi HTML-kielessä nämä ovat linkkejä ja syöttölomakkeita (Allamaraju 2010, 263).

2.6 Kerroksittainen järjestelmä -rajoite (Layered System)

Kerroksittaisen järjestelmän rajoite tarkoittaa sitä, että järjestelmä koostuu hierarkkisesti järjestetyistä kerroksista, ja ohjelmistokomponenttien pääsy rajoittuu vain lähimpään kerrokseen. Komponenteilla ei ole pääsyä eikä tarvetta tietää, miten muu järjestelmä on koostunut lähimmän kerroksen takana. Tämä antaa mahdollisuuksia muuttaa järjestelmän kokoonpanoa kerroksen takana, ilman että ohjelmistokomponenttia tarvitsee mitenkään muuttaa. Lisäksi komponenteista tulee yksinkertaisempia ja itsenäisempiä. (Fielding 2000, 82–84.)

Sovelluksen jako kerroksiin lisää järjestelmän dataprosessointia, ja siten voi lisätä viiveitä. Kerroksittainen järjestelmä tosin mahdollistaa välimuistien lisäämisen eri kerroksien väliin, jolloin voidaan saavuttaa huomattavia tehokkuusparannuksia.



KUVIO 4. Kerroksittainen järjestelmä

2.7 Suoritettavan koodin lähettäminen -rajoite (Code-on-demand)

Suoritettavan koodin lähettäminen -rajoite antaa palvelimelle mahdollisuuden lähettää osa suoritettavasta koodista vastauksen osana asiakkaalle. Esimerkiksi web-sivuun voi kuulua osana javascipt koodia. Asiakasohjelmaan eli tässä tapauksessa selaimeen on toteutettu tuki javascript koodin tulkitsemiseen. Skripti-koodilla web-sivuun voidaan toteuttaa sivukohtaisia monimutkaisempia toimintoja. Selaimeen ei tällöin tarvitse toteuttaa lukuisaa joukkoa erilaisia ominaisuuksia, vaan ne voidaan ladata sivun mukana.

Suoritettavan koodin lähettäminen heikentää REST-palvelun vastausten tulkintaa. Jotta sisältö voidaan täysin tulkita, tarvitaan vastauksen lisäksi ladattavaan koodin ymmärtävä sovellus. Tämä rajoite onkin valinnainen (Fielding 2000, 84), REST arkkitehtuuria noudattavia sovelluksia voi rakentaa ilman sen käyttöä.

2.8 REST ohjelmistoarkkitehtuurin erot RPC-pohjaisiin web rajapintoihin

REST on vain yksi tapa web-rajapintojen toteuttamiseen. Toinen tapa toteuttaa rajapintoja ovat RPC-tyyppiset (Remote Procedure Call) rajapinnat (Richardson & Ruby 2007, 14). RPC eroaa REST:stä kahdella tavalla:

- tyypillisesti RPC rajapintaa kutsutaan vain yhden osoitteen kautta (endpoint)
- suoritettava toiminto kerrotaan välitettävän viestin sisällä .

Yhden osoitteen käyttö rikkoo REST-periaatteita siinä, että kukin resurssi tulisi olla yksilöllisellä viitteellä tunnistettavissa. Tämä estää välimuistien käytön, eli suorituskyvyltään RPC-tyyppiset rajapinnat eivät ole yhtä skaalautuvia.

REST:ssä suoritettava toiminto kuvattaisiin varsinaisessa kutsussa HTTP-metodilla (GET, POST yms.). Web-pohjaisen sovelluksen suorituskykyä voidaan lisätä laittamalla välimuisti asiakkaan ja palvelun väliin. Kun suoritettava toiminto on itse

kutsussa eikä viestin sisällössä, voi välimuisti päätellä asiakkaan kutsusta, että voidaanko välimuistia käyttää. Esimerkiksi mikäli kutsu on HTTP GET -tyyppinen, välimuisti voi päätellä, että resurssi voidaan hakea välimuistista. Jos kutsu taasen on muun tyyppinen (PUT, POST, DELETE) eli sellainen, joka jollain tavalla muuttaa dataa palvelimella, kutsu on välitettävä perille asti eikä välimuistia voida hyödyntää.

SOAP-protokolla on RPC-pohjainen rajapinta (Richardson & Ruby 2007, s.19). Tyypilliset SOAP toteutukset kutsuvat vain yhtä osoitetta, ja suoritettava toiminto kerrotaan täysin välitettävän viestin sisällä. SOAP-protokolla ei ole sidottu suoraan http-protokollaan. SOAP-sanomia välitetään myös esimerkiksi sanomajonojen sanomina. Siitä huolimatta valtaosa SOAP-toteutuksista käyttää HTTP-protokollaa sanomien lähettämiseen (Richardson & Ruby 2007, 303).

RPC-pohjaiset (SOAP mukaan lukien) toteutukset, jotka käyttävät HTTP-protokollaa sanomanvälitykseen, siis käytännössä hukkaavat HTTP-protokollan tuomia etuja.

3 ASIAKIRJAPALVELUN KUVAUS

3.1 Toteuttajan ja Asiakkaan lyhyt kuvaus

3.1.1 Alfame Systems Oy

Alfame Systems Oy on kokkolalainen vuonna 2004 perustettu yritys, jossa on nykyään noin 30 työntekijää. Yritys tuottaa asiakkaille räätälöityjä ratkaisuja erilaisiin tarpeisiin. Pääsääntöisinä ohjelmointialustoina projekteissa on joko avoimen lähdekoodin ympäristöt (erityisesti Java) tai Microsoft-pohjaiset alustat (Sharepoint, .Net, Visual Basic yms.). Microsoft-pohjaisia ratkaisuja käytetään tyypillisesti silloin, kun ne tarjoavat melkein valmiin ratkaisun asiakkaan tarpeisiin. Java-pohjaiset projektit puolestaan ovat käytössä silloin, kun asiakkaan tarpeet vaativat paljon omaa kehitystyötä.

3.1.2 Patentti- ja rekisterihallitus

Patentti- ja rekisterihallitus kuuluu työ- ja elinkeinoministeriön konserniin. Sillä on kaksi päätehtävää: myöntää suojia erilaisille immateriaalioikeuksille ja ylläpitää rekisteriä suomalaisista yrityksistä, yhdistyksistä ja säätiöistä. PRH on virasto, joka kattaa 95 % toiminnan kustannuksista erilaisilla toimintoihin liittyvillä maksuilla.

PRH:ssa on kolme linjaa, jotka hoitavat PRH:lle kuuluvia tehtäviä. Yritys- ja yhteisölinja huolehtii yritysten, yhdistysten ja säätiöiden rekisteröinnistä ja ajantasaisuudesta. Patentti- ja innovaatiolinja huolehtii kansallisten ja kansainvälisten patenttihakemusten käsittelystä ja patenttien rekisteröinnistä. Lisäksi patenttilinja vastaa hyödyllisyysmallioikeuksien hakemuksista ja rekisteröinnistä. Tavaramerkki- ja mallilinjan tehtäviin kuuluvat tavaramerkkien ja mallioikeusasioiden käsittely sekä tavaramerkki- ja mallirekisterin ylläpito.

Patenttihakemuksia on PRH:lle tehty vuonna 2013 1737 kpl. Patentteja on myönnetty 711 kpl. (Patentti- ja Rekisterihallituksen vuosikertomus 2013.)

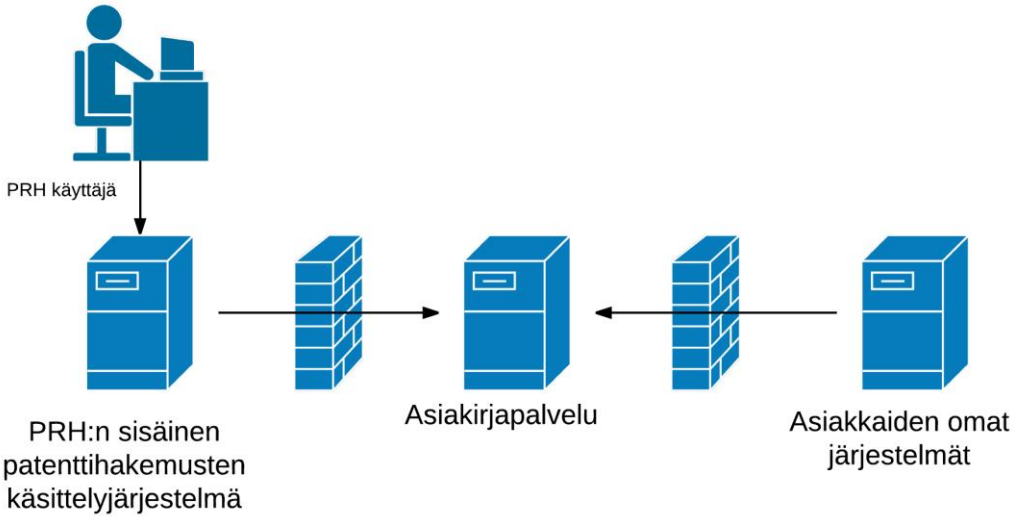
3.2 Toteutettavan palvelun kuvaus

3.2.1 Lähtökohta

Patentin hakijana voi toimia yksityinen henkilö, yritys tai patentin hakijaa edustava asiamies. Asiamies tyypillisesti on immateriaalioikeuksiin lakiyritys. Patenttien myöntämiseen liittyy tiedonvaihtoa PRH:n ja patentin hakijan välillä. PRH käyttää järjestelmää patenttihakemuksen prosessin käsittelemiseen. Tästä järjestelmästä, patenttihakemuksiin liittyvää kirjeenvaihtoa tulostettiin ja postitettiin patentin hakijoille. Lisäksi patentin hakijana toiminut yritys tai asiamies saattoi skannata ja siirtää saapuneet paperitulosteet hakijan omiin käsittelyjärjestelmiin. Tämä aiheutti turhaa työtä molemmille osapuolille, ja erityisesti isojen yritysten ja asiamiesten puolelta esitettiin toiveita kirjeenvaihdon sähköisen rajapinnan luomisesta.

3.2.2 Tavoite

Projektin tavoitteena oli toteuttaa palvelu, jossa PRH:n sisäisestä käsittelyjärjestelmästä patenttihakemusten käsittelyyn liittyvät kirjeet siirrettäisiin erilliseen palveluun, josta asiakas voisi puolestaan hakea ne omiin järjestelmiinsä. Ainakin alkuvaiheessa oli tarkoitus tarjota tämä järjestelmä vain sellaisille hakijoille, jotka tekevät paljon patenttihakemuksia (isot yritykset ja asiamiestoimistot). Palvelu tarjoaa ohjelmointirajapinnat, joita vasten asiakkaan järjestelmät voivat tehdä kutsuja, hakea kirjeitä ja siirtää ne suoraan omiin sähköisiin käsittelyjärjestelmiinsä. Koska palvelun käyttäjillä on erilaisia järjestelmiä, tuli palvelun olla yleiskäyttöinen.



KUVIO 5. Palvelun periaatekuva

Oheisesta kuvassa on esillä palvelua käyttävät tahot. PRH:n patenttihakemusta käsittelevä henkilö voi siirtää asiakkaalle tarkoitetun asiakirjan asiakirjapalveluun. Asiakirjapalvelu sijaitsee erillisellä palomuurilla suojatulla alueella. Asiakkaiden omat tietojärjestelmät voivat ottaa yhteyttä palveluun ja hakea asiakirjan omiin tietojärjestelmiinsä.

3.2.3 Palvelun toiminnot

Palvelun toimintoihin kuuluvat seuraavat asiakkaille tarkoitetut toiminnollisuudet:

- listaus asiakkaalle kuuluvista asiakirjoista, tulosten rajaaminen mahdollista hakuehtoja käyttämällä
- yhden asiakirjan haku
- asiakirjan saantitodistuksen haku
- listaus kaikista tiettyyn kansioon kuuluvista asiakirjoista .

PRH:n sisäiset järjestelmät puolestaan integroidaan asiakirjapalveluun tarjoamalle niille omat toiminnot:

- lisää asiakirjoja tietylle asiakkaalle tiettyyn kansioon
- lisää kansio
- poista asiakirja
- merkitse asiakirja poistetuksi (tällöin asiakas näkee asiakirjan listauksessa, mutta ei voi hakea varsinaista asiakirjaa)
- listaus kaikista tietylle asiakkaalle kuuluvista asiakirjoista
- listaus kaikista tietylle asiakkaalle tiettyyn kansioon kuuluvista asiakirjoista
- listaus kaikista tiettyyn kansioon kuuluvista asiakirjoista
- listaus kaikista palveluun tehdyistä tileistä ja niihin kuuluvista kansioista ja asiakirjoista
- listaus kaikista yhteen tiliin kuuluvista asiakirjoista .

4 TOTEUTUS

4.1 Toteutusprosessin lyhyt kuvaus

Toteutusprosessin aikana käytettiin ketterää menetelmää. Projektin alussa oli määritelty sähköisen palvelun tavoitteet yleisellä tasolla. Toteutuksen edetessä määrityksiä tarkennettiin ja toiminnollisuuksia lisättiin tarpeen mukaan.

Projektin aikana käytetty ketterä menetelmä oli Scrum. Scrum menetelmän pääelementit ovat:

- tuotteen kehitysjono (Product Backlog)
- sprintti (Sprint)
- sprintin suunnittelu (Sprint Planning)
- päiväpalaveri (Daily Scrum)
- scrummaster (Scrum Master)
- tuoteomistaja (Product Owner)
- tuoteversio (Increment) .

(Suomenkieliset termit: Suomenkielinen scrum-sanasto 2014.)

Asiakkaan vaatimukset, käyttötapaukset ja toiminnollisuudet kerätään tuotteen kehitysjonoon. Kehitysjonoon voi aina lisätä uusia kehitettäviä asioita ja niiden ei tarvitse olla täysin loppuun asti määriteltyjä. Kehitysjonoon lisätyille asioille annetaan työmääräarvio.

Sprintin suunnittelupalaverissa valitaan kehitysjonosta asiat, jotka halutaan sisällyttää seuraavaan sprinttiin. Sprintti on tietyn ajan kestävä kehityspanostus, joka kestää tyypillisesti kahdesta neljään viikkoa. Työmääräarviot ja märitykset yleensä tarkentuvat tässä yhteydessä. (Schwaber & Beedle 2002, 7-10).

Suunnittelupalaveriin voi osallistua enemmänkin henkilöitä, mutta varsinaiseen kehitykseen osallistuvat tyypillisesti kehittäjät, scrummaster ja tuoteomistaja. Scrummaster on projektipäällikkö, jonka vastuulla on projektin eteneminen ja ohjaaminen. Tuoteomistaja on sellainen henkilö, joka voi tehdä päätöksiä tuotteen ominaisuuksista. Parhaissa tapauksissa tuoteomistaja on asiakkaan edustaja. Scrum-menetelmässä ei pyritä kuvaamaan sovellusta täydellisesti heti projektin alkaessa, vaatimuksia muutetaan ja täydennetään kehitystyön edetessä. Tuoteomistajan rooli on tärkeä, jotta näitä muutoksia ei jouduta odottelemaan.

Scrum-menetelmällä pyritään takaamaan kehittäjille työrauha kunkin sprintin ajaksi. Samalla pyritään myös antamaan tilaa uusille kehitysideoille ja kehityskohteiden priorisoinnin muutoksille. Sprintin aikana tulleet uudet vaatimukset lisätään tuotteen kehitysjonoon, ja seuraavan sprintin suunnittelupalaverissa päätetään, mitä tehtäviä toteutetaan seuraavassa sprintissä.

Päivittäiseen tekemiseen kuuluu päiväpalaveri, johon osallistuvat vähintään kehitystiimi, scrummaster ja tuoteomistaja. Kunkin henkilön osalta käydään läpi seuraavat asiat:

- mitä on tehnyt edellisen päiväpalaverin jälkeen
- mitä aikoo tehdä seuraavaksi
- onko ongelmia, mitkä ovat esteenä .

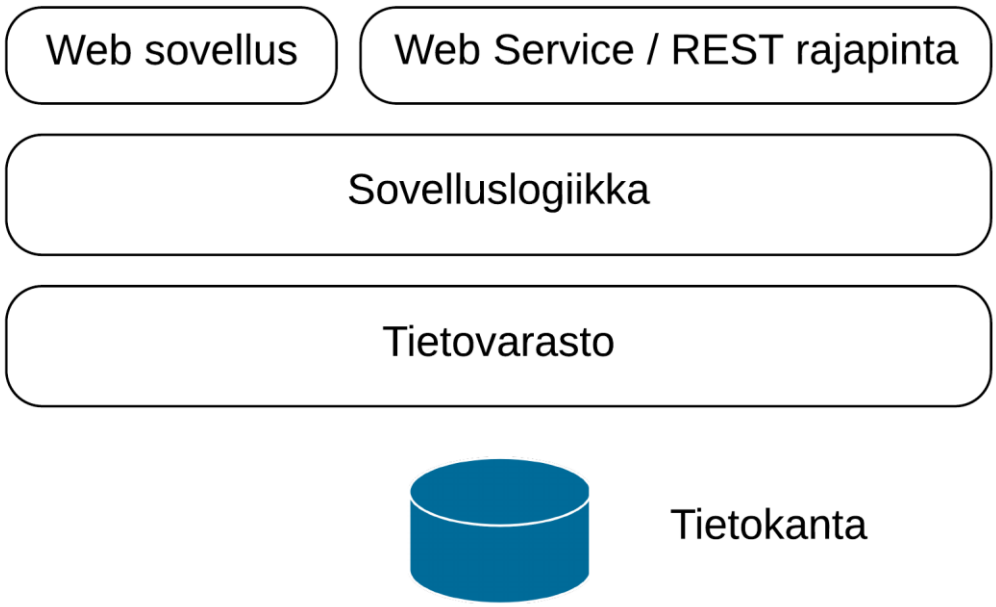
Päiväpalaverin tehtävänä on tuoda nopeasti esille mahdolliset ongelmat ja esteet, sekä löytää niihin ratkaisuja. Itse päiväpalaverissa ongelmia ei ryhdytä ratkaisemaan, vaan tarvittavat henkilöt voivat ratkoa ongelmaa keskenään palaverin jälkeen.

Kunkin sprintin lopussa tulee olla sellainen tuoteversio, mikä voidaan asentaa käyttäjille testattavaksi, ja josta löytyy sprintin suunnittelupalaverissa sovitut

ominaisuudet. Koska sprintin pituus on tyypillisesti kahdesta neljään viikkoa, asiakas pysyy hyvin perillä siitä, miten tuotteen kehitys edistyy.

4.2 Toteutuksen arkkitehtuuri ja käytetyt tekniikat

Sovellus toteutettiin Java-ohjelmointikielellä. Sovelluksen arkkitehtuuri on tyypillinen kolmikerrosarkkitehtuuri, jossa tietovarasto-, sovelluslogiikka- ja web service -kerros ovat kukin erillisiä kokonaisuuksiaan ja keskustelevat keskenään selkeiden ja erillisten rajapintojen kautta. Tämä mahdollistaa sovelluksen joustavan ylläpidon ja kehityksen jatkossa.



KUVIO 6. Kolmikerrosarkkitehtuuri

Kuvassa oleva alin kohta kuvaa tietokantapalvelinta ja siinä sijaitsevaa, tätä järjestelmää varten muodostettavaa tietokantaa. Ylempänä oleva laatikko kuvaa komponenttia, jonka tehtävän on hoitaa kaikki yhteydet tietokantaan päin. Kaikki logiikka, missä haetaan, luodaan, päivitetään tai poistetaan tietueita tietokannan taulujen riveistä, sijoitettiin tähän kerrokseen. Tietokantayhteyksien toteuttamisessa

käytettiin avoimen lähdekoodin Hibernate-komponenttia, jonka avulla tietovarastotoimintojen toteuttaminen on nopeampaa verrattuna esimerkiksi suoraan JDBC-rajapinnan ja SQL-lauseiden käyttöön.

Tietokantakerroksen yläpuolella on sovelluslogiikkakerros. Kaikki sovelluksen tietoja muuttavat tai esimerkiksi laskennat yms. sijoitettiin tähän kerrokseen. Tarkoituksena on, että tämän kerroksen yläpuolella olevat rinnakkaiset kerrokset (web, web services) ottavat kantaa vain enää itse toimintojen rajapinnan toteutukseen, jolloin järjestelmän sovelluslogiikkaa ei tarvitse toistaa.

Kuvassa ylimpänä olevaan 'Web Service / REST rajapinta' -kerrokseen toteutettiin itse REST-rajapinta. Tämän kerroksen tehtävä on vastaanottaa kutsut, kutsua sovelluslogiikkakerrosten metodeja ja välittää vastaukset kutsujalle.

Koska sovellukseen toteutettiin myös selainpohjainen käyttöliittymä REST-rajapinnan lisäksi, kerroksittaisesta rakenteesta oli välitöntä hyötyä. Selainpohjainen käyttöliittymä käytti samoja sovelluslogiikkakerroksen metodeja kuin REST-rajapinta. Selainpohjainen käyttöliittymä rajattiin tämän lopputyön ulkopuolelle.

Kerrosten ohjelmoinnissa käytettiin 'Inversion Of Control' periaatetta(IOC), jossa komponenttien välisiä keskinäisiä riippuvuuksia ei ohjelmoida itse koodiin, vaan se toteutetaan konfiguroinnilla ja sopivalla IOC-kehyksellä. Tämä vähentää koodin määrää sekä tekee sovelluksesta helpommin muunneltavan.

IOC-kehyksenä käytettiin Spring Framework:iä, joka on suuren suosion saavuttanut avoimen lähdekoodin sovelluskehys. Se tuo mukanaan myös mahdollisuuden käyttää 'Aspect Oriented Programming' (AOP) -tyyppisiä ratkaisuja, jossa sovelluksen sellaiset ominaisuudet, mitä tarvitaan useissa toiminnoissa eri puolella

sovellusta, voidaan eriyttää omiksi komponenteikseen ja ottaa ne käyttöön konfiguroinnin avulla. Mm. sovelluksen tietokantatransaktiot ja lokitus toteutettiin näitä ominaisuuksia hyväksi käyttäen.

4.3 Hakurajapinnan toteutus (yhden rajapinnan toteutus esimerkkinä)

Kutsujen ja vastausten toteutus aloitettiin määrittämällä vastaussanomien muoto xml skeema (xsd) -tiedostoilla.

Ohessa asiakirjan ja asiakirjaluettelon määritys skeema tiedostona:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <xs:element name="letters">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="letter"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="letter">
    <xs:complexType>
      <xs:sequence minOccurs="0">
        <xs:element ref="language"/>
        <xs:element ref="url"/>
        <xs:element ref="letter-id"/>
        <xs:element ref="letter-code"/>
        <xs:element ref="odcorresp"/>
        <xs:element ref="form-id"/>
        <xs:element ref="datafile-id"/>
        <xs:element ref="citation-id"/>
```

```
        <xs:element ref="letter-description"/>
        <xs:element ref="letter-content-b64"/>
        <xs:element ref="external-link"/>
        <xs:element ref="mailing-date"/>
        <xs:element ref="transfer-timestamp"/>
        <xs:element ref="due-date"/>
        <xs:element ref="ack-of-receipt-required"/>
        <xs:element ref="is-acknowledged"/>
        <xs:element ref="acknowledge-timestamp"/>
        <xs:element ref="acknowledge-user-id"/>
        <xs:element ref="acknowledge-certificate-id"/>
        <xs:element ref="first-time-read-by"/>
        <xs:element ref="first-time-read-timestamp"/>
        <xs:element ref="is-letter-deleted"/>
        <xs:element ref="delete-reason"/>
        <xs:element name="document-id" type="xs:string"/>
              <xs:element name="attachments">
              <xs:complexType>
                    <xs:sequence>
                          <xs:element maxOccurs="unbounded" ref="letter" />
                    </xs:sequence>
              </xs:complexType>
              </xs:element>
    </xs:sequence>
    <xs:attribute name="sequence" type="xs:NCName"/>
  </xs:complexType>
</xs:element>
<xs:element name="language" type="xs:NCName"/>
<xs:element name="url" type="xs:string"/>
<xs:element name="letter-id" type="xs:long"/>
<xs:element name="letter-code" type="xs:string"/>
<xs:element name="odcorresp" type="xs:int"/>
<xs:element name="form-id" type="xs:int"/>
<xs:element name="datafile-id" type="xs:int"/>
<xs:element name="citation-id" type="xs:int"/>
<xs:element name="letter-description" type="xs:string"/>
```

```
  <xs:element name="letter-content-b64" type="xs:string"/>
  <xs:element name="external-link" type="xs:string"/>
  <xs:element name="mailing-date" type="xs:date"/>
  <xs:element name="transfer-timestamp" type="xs:dateTime"/>
  <xs:element name="due-date" type="xs:date"/>
  <xs:element name="ack-of-receipt-required" type="xs:boolean"/>
  <xs:element name="is-acknowledged" type="xs:boolean"/>
  <xs:element name="acknowledge-timestamp" type="xs:dateTime"/>
  <xs:element name="acknowledge-user-id" type="xs:long"/>
  <xs:element name="acknowledge-certificate-id" type="xs:long"/>
  <xs:element name="first-time-read-by" type="xs:string"/>
  <xs:element name="first-time-read-timestamp" type="xs:dateTime"/>
  <xs:element name="is-letter-deleted" type="xs:boolean"/>
  <xs:element name="delete-reason" type="xs:string"/>
</xs:schema>
```

Maven Jaxb2 pluginia käyttämällä ko. tiedostoista tuotettiin java-luokat rest-kutsujen vastauksia varten. Syntyneet luokat sisälsivät kuhunkin olioon liittyvät tiedot ja niiden get- ja set-metodit, sekä Java-annotaatiolla kuvaus siitä miten kukin tieto kerrotaan xml vastauksessa. Annotaatio kuvasi xml-tagien nimet ja niiden keskinäiset riippuvuudet yms.

Ohessa on muodostunut Letters-luokka esimerkkinä:

```
package fi.prh.service.documentservice.jaxb;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
```

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "letter"
})
@XmlRootElement(name = "letters")
public class Letters {

    @XmlElement(required = true)
    protected List<Letter> letter;

    public List<Letter> getLetter() {
        if (letter == null) {
            letter = new ArrayList<Letter>();
        }
        return this.letter;
    }

}
```

Seuraavaksi luotiin varsinaisen REST-rajapinnan tarjoava java-rajapinta. Luokka merkittiin Rest-rajapinnaksi antamalla sille java-annotaatio Provider, sekä kertomalla Path-annotaatiolla rajapinnan yleinen osoite:

```
@Provider
@Path(IDocumentService.REST_SERVICE_URI)
public interface IDocumentService {

        public static final String REST_SERVICE_URI = "/rest";
        public static final String INBOX_URI = "/mail/inbox";

        /**
         * Get letters pointed to customer, ordered and contained in          *
         * dossiers.
         * @param customerId
         * @return
         * @throws Exception
```

```
 */
@GET
@Path("/{customer-id}" + INBOX_URI)
@Produces({ "application/xml", "application/json" })
public SearchResults getMailInbox(
        @PathParam("customer-id") String customerId,
        @QueryParam("searchText") String  searchText,
        @QueryParam("sendDateSearchBegin") String
                sendDateSearchBegin,
        @QueryParam("sendDateSearchEnd") String    sendDateSearchEnd,
        @QueryParam("transferDateSearchBegin") String
                transferDateSearchBegin,
        @QueryParam("transferDateSearchEnd") String
                transferDateSearchEnd,
        @QueryParam("dueDateSearchBegin") String  dueDateSearchBegin,
        @QueryParam("dueDateSearchEnd") String  dueDateSearchEnd)
throws Exception;
```

Rajapinnan osoitteeksi muodostuu https://docservice.prh.fi/rest/{customer-id}/mail/inbox, jossa customer-id on kullekin asiakastunnukselle yksilöllinen tunniste. Rajapinta hakee kutsusta customer-id tiedon. Lisäksi osoitteen perään voidaan antaa hakuparametreja rajaamaan tuloksia vapaatekstihaulla tai erilaisilla asiakirjaan kohdistuvilla päivämäärillä.

Skeema-tiedostot, niistä syntyneet luokat ja palvelun kuvaava rajapinta toteutettiin omaan erilliseen java-projektiin. Projektista oli tällöin mahdollista luoda erillinen Java jar-tiedosto, joka voitiin antaa asiakkaan käyttöön. Samaa projektia käytettiin sitten myös luotaessa varsinainen rajapinnat tarjoava sovellus.

Rajapinta toteutettiin käyttämällä annotaatioita metodikutsuissa. Ohessa esimerkki rajapinnan toteuttavan luokan metodista, joka palauttaa listauksen asiakkaalle kuuluvista asiakirjoista:

```
@Override
public SearchResults getMailInbox(@PathParam("customer-id") String     customerId,
        @QueryParam("searchText") String  searchText,
        @QueryParam("sendDateSearchBegin") String  sendDateSearchBegin,
        @QueryParam("sendDateSearchEnd") String  sendDateSearchEnd,
        @QueryParam("transferDateSearchBegin") String
        transferDateSearchBegin,
        @QueryParam("transferDateSearchEnd") String  transferDateSearchEnd,
        @QueryParam("dueDateSearchBegin") String  dueDateSearchBegin,
        @QueryParam("dueDateSearchEnd") String  dueDateSearchEnd) throws
        Exception {


        SearchTermsForInbox searchTermsForInbox = new
        SearchTermsForInbox();
        searchTermsForInbox.setSearchText(searchText);
        searchTermsForInbox.setSendDateSearchBegin(
        convertUtil.convertStringToDate( sendDateSearchBegin, null));

        searchTermsForInbox.setSendDateSearchEnd(
        convertUtil.convertStringToDate(sendDateSearchEnd, null));

        searchTermsForInbox.setTransferDateSearchBegin(
        convertUtil.convertStringToDate(transferDateSearchBegin,
        null));
                searchTermsForInbox.setTransferDateSearchEnd(
                convertUtil.convertStringToDate(transferDateSearchEnd,
                        null));
                searchTermsForInbox.setDueDateSearchBegin(
                convertUtil.convertStringToDate(dueDateSearchBegin, null));
                searchTermsForInbox.setDueDateSearchEnd(
                convertUtil.convertStringToDate(dueDateSearchEnd, null));

        SearchResults searchResults =
```

```
        searchManager.getSearchResultsForCustomerInChosenOrder(customerId,
                true, "readDate", searchTermsForInbox);

                return searchResults;
        }
```

Rajapinnan toteutus muuttaa annetut tekstipohjaiset päivämäärät Javan Date-olioiksi ja suorittaa kutsun sovelluslogiikkakerroksen metodille, joka palauttaa tulokset listauksena Java-objekteja. Nämä objektit ovat xml-skeematiedostoista luotuja luokkia. Sovelluskehys osaa itse muodostaa objektien pohjalta oikeanlaisen vastauksen xml-muodossa.

4.4 Tietovarasta- ja logiikkakerrosten toteutus

Tietovarastona sovelluksessa käytettiin MySQL-tietokantaa. Tietokantaluokat toteutettiin Java Persistence API -annotaatioita käyttäen, josta Hibernaten työkaluilla luotiin tietokannan rakenne.

Ohessa ote tietokantaluokasta:

```
package fi.prh.service.documentservice.domain;

import java.util.ArrayList;
…….
import org.hibernate.annotations.Cascade;

/**
 * Domain object to hold information of the account that uses
 * documentservice.
 */
@Entity
@Table(name="account" )
```

```
public class Account extends PersistableEntity {

	@Column(unique = true, length = 32, name="account_site", nullable =
	false)
	private String accountSite;
	@Column(unique=true, name="account_name", nullable = false)
	private String accountName;
	@Column(name="account_description", nullable = false)
	private String accountDescription;

	@Column(length = 32, name="email_notification_address", nullable =
	false)
	private String emailNotificationAddress;

	@OneToMany(targetEntity = Letter.class ,fetch = FetchType.LAZY,
	cascade = CascadeType.ALL)
	@Cascade({org.hibernate.annotations.CascadeType.ALL,
	org.hibernate.annotations.CascadeType.DELETE_ORPHAN})
	@JoinColumn(name = "account_id")
	@OrderBy("dossier")
	private List<Letter> letters = new ArrayList<Letter>();

	public List<Letter> getLetters() {
		return letters;
	}

	... ...

}
```

Tietokantaluokka esittää asiakastilin tallentamiseen tarkoitettua luokkaa. Asiakkaasta tallennetaan mm. tilin nimi, tilin kuvaus ja sähköpostiosoite. Lisäksi luokassa on viittaus kaikkiin asiakkaalle kuuluviin kirjeisiin. Hibernate-työkalu tulkitsee luokan ja muodostaa sen pohjalta tarvittavat sql-lauseet sopivan tietokantarakenteen luomiseksi.

Toiminnot tietojen luomiseen, päivittämiseen, poistamiseen ja hakemiseen toteutetaan omiin rajapinta- ja toteutusluokkiin.

Esimerkki tietokannan toimintorajapinnasta;

```
@Transactional
public interface IAccountDao extends IBaseDao<Account> {

	@Transactional(readOnly=true)
	public Account getAccountByAccountId(String accountId);
	@Transactional(readOnly=true)
	public Account getAccountByAccountName(String accountName);
	@Transactional(readOnly=true)
	public Account getAccountByAccountIdFetchLetters(String accountId);
	... ...
}
```

Rajapintaan on kuvattu halutut metodit tietojen käsittelyä varten. Spring-kehyksen Transactional-annotaatiolla voidaan kertoa kehykselle, että toiminto halutaan tapahtuvan tietokantatransaktion sisällä. Sovelluksessa on otettu käyttöön springin tapa transaktioiden käsittelyyn. Tällöin omassa sovelluskoodissa ei tarvitse erikseen aloittaa tai lopettaa transaktioita, vaan kaikki määritellään annotaatioilla.

Rajapinnan toteutus on omassa luokassaan:

```
public class AccountDaoHibernateImpl extends	BaseDaoHibernateImpl<Account>
implements	IAccountDao {

	@Override
	public Account getAccountByAccountId(String accountId) {
		Query query = sessionFactory.getCurrentSession()
	.createQuery("from Account a where a.accountSite = :accountId ");
		query.setString("accountId", accountId);
		List list  = query.list();
```

```
            if (list.size() > 0) {
                return (Account)list.get(0);
            } else {
                return null;
            }
        }

        @Override
        public Account getAccountByAccountIdFetchLetters(String
        accountSite) {
            Query query = sessionFactory.getCurrentSession() .createQuery(
            "from Account a LEFT JOIN FETCH a.letters where  " +
                    "a.accountSite = :accountId ");
            query.setString("accountId", accountSite);
            List list  = query.list();
            if (list.size() > 0) {
                return (Account)list.get(0);
            } else {
                return null;
            }
        }
        … …
}
```

Toteutuksessa on käytetty hibernaten omaa HQL-kieltä tietokantahaun tekemiseen. Hibernate muodostaa annetusta HQL-lauseesta oikeat SQL-lauseet, suorittaa haun kantaan, luo vastaavat Java-oliot ja asettaa olioihin kannasta saadut tiedot. Hibernaten käyttäminen yksinkertaistaa omaa ohjelmaa, erityisesti kun haetaan olioita, joilla on yhden suhde moneen riippuvuuksia. Hibernate pystyy hakemaan myös riippuvat oliot ja asettamaan ne olioon.

Sovelluslogiikkakerros toteutetaan myös rajapinta ja toteutusluokka-paria käyttämällä.

Esimerkki sovelluslogiikkakerroksen rajapinnasta:

```
@Transactional
public interface ISearchManager {


        /**
         * Get Letters for customer, orderField specifies field used for
         * sorting
         * @param accountSite
         * @param orderField , possible values: "description",
         *  "dossier.documentId.dossierNumber", "mailingDate", "dueDate"
         * @param ascending, true if ascending, false if descending order
         * @return
         */
        @Transactional(readOnly = true)
        public List<Letter> getLettersForCustomerInChosenOrder(String
                accountSite, boolean ascending, String orderField,
                SearchTermsForInbox searchTermsForInbox);

        ... ...


}
```

Myös sovelluslogiikkakerroksen metodeissa voidaan kertoa transaktio-annotaatioilla transaktiomäärityksiä. Tällöin transaktio alkaa jo sovelluslogiikkakerroksen metodin käynnistyessä ja päättyy metodin lopussa. Tällöin jos yhdestä metodista kutsutaan esimerkiksi useampaa eri tietokantakerroksen metodia, tapahtuvat molemmat kutsut yhden ja saman transaktion aikana.

Rajapinnan toteutus on omassa luokassaan:

```
public class SearchManagerImpl implements ISearchManager,
        InitializingBean {
```

```
        private ILetterDao letterDao;
        private IDeskClient deskClient;


        @Override
        public List<Letter> getAttachmentsForCustomerInChosenOrder(String
        accountSite, long letterId, boolean ascending, String
        orderField,   SearchTermsForInbox searchTermsForInbox, int
        firstResult, int maxResults) {

                List<Letter> letters =
                letterDao.getAttachmentsForCustomerAndLetterInChosenOrder(
                        accountSite, letterId, orderField, ascending,
                        searchTermsForInbox, firstResult, maxResults);
                deskClient.addReferenceInformation(letters);
                return letters;
        }
        ... ...
}
```

Oheisessa esimerkissä tietokantakerroksesta pyydetään tietoa, ja sen jälkeen tietoa rikastetaan tekemällä toinen kutsu, joka tässä tapauksessa hakee lisätietoa PRH:n sisäisestä järjestelmästä. Tässä esimerkissä käy myös hyvin ilmi toimintojen jako eri kerroksiin. Tietokantakerroksen tarvitsee vain hakea tietoa tietokannasta, sovelluslogiikkakerros kutsuu tietokantakerrosta ja rikastaa sitä tekemällä kutsun muualle. Sovelluslogiikkakerroksen yläpuolella olevan kerroksen (web, web services) tarvitsee vain puolestaan kutsua sovelluslogiikkakerroksen metodia.

5 YHTEENVETO

Työn tarkoituksena oli toteuttaa REST-arkkitehtuurityyliin pohjautuva sähköinen palvelu PRH:n patenttikirjeenvaihtoa varten. Työn yhteydessä tutustuin REST:in teoriaan, erityisesti Roy Fieldingin väitöstyöhön. Siinä kuvataan hyvin HTTP-protokollan tarjoamat hyödyt hajautetuille järjestelmille. REST-arkkitehtuurityyliä noudattamalla pystytään toteuttamaan rajapintoja, jotka ovat selkeitä, suorituskykyisiä ja yhtenäisiä muitten REST rajapintojen kanssa.

Varsinaisen projektin toteutus tapahtui ketterää scrum-menetelmää käyttämällä. Alkutilanteessa oli tavoite ja yleiset suuntaviivat, mutta toteutuksen edetessä määrityksiä tarkennettiin. Sovellus toteutettiin Java-ohjelmointikielellä käyttäen monia ohjelmistokehyksiä apuna. Sovelluksen sisäinen arkkitehtuuri noudatti tyypillistä kolmikerrosarkkitehtuuria, jossa kerrosten välinen kommunikointi tapahtui selkeitä rajapintoja kutsuen. Sovellus on otettu tuotantokäyttöön, ja siihen on oltu tyytyväisiä.

LÄHTEET

Allamaraju, S. 2010. RESTful Web Services Cookbook. United States of America: Yahoo.

Fielding, R. 2000. Architectural Styles and the Design of Network-based Software Architectures. Irvine: University of California.

Richardson, L. & Ruby, S. 2007 RESTful Web Services. United States of America: O'Reilly Media, Inc.

Schwaber, K. & Beedle, M. 2002 Agile Software Development with Scrum. United States of America: Prentice Hall, Inc.

Suomenkielinen scrum-sanasto. 2014. Pdf-dokumentti. Saatavissa: https://scrumwell.files.wordpress.com/2014/03/suomenkielinen-scrum-sanasto-2014-v2.pdf. Luettu 22.5.2015.

Patentti- ja Rekisterihallituksen vuosikertomus 2013. Pdf-dokumentti. Saatavissa: https://www.prh.fi/stc/attachments/tietoaprhsta/vuosikertomus/vuosikertomus_2013.indd.pdf. Luettu 22.5.2015.