Hakumashup asiakaspalvelutyön tueksi

Arkkitehtuurin suunnittelu ja toteutus

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittelyn ko.

Visamäki, kevät 2013

Ville Pynttäri

HAMK
HÄMEEN AMMATTIKORKEAKOULU

TIIVISTELMÄ

Hämeenlinna
Tietojenkäsittelyn ko.
Systeemityö

Tekijä Ville Pynttäri **Vuosi** 2013

Työn nimi Hakumashup asiakaspalvelutyön tueksi, arkkitehtuurin suunnittelu ja toteutus

TIIVISTELMÄ

Työn toimeksiantaja oli Ambientia Oy:n asiakaspalvelukeskus. Se halusi työskentelynsä tehostamiseksi työkalun, jolla voidaan tehdä yhdistetty haku yrityksen tarvitsemiin järjestelmiin. Yhdistetyllä haulla on tarkoitus saada puhelun aikana tarpeelliset tiedot asiakkaasta, jotta puhelua ei tarvitsisi katkaista tiedon hakemisen ajaksi.

Tämän opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa asiakaspalvelukeskukselle työkalu, joka toteuttaa haut järjestelmiin. Työkalu on Spring-sovelluskehyksen päälle tehty sovellus, jossa yhdistetään JIRAn, Confluencen ja PlanMillin haut kutsumalla niiden REST-rajapintoja. Hakutyökalu on tehty juuri asiakaspalvelukeskuksen tarpeisiin, ei täysin yleishyödylliseksi integraatioksi.

Opinnäytetyön tuloksena saatiin aikaan kehitysympäristössä täysin toimiva sovellus. Sovellus on myös asetettu tuotantoympäristöön, mutta käyttöönotto vaatii vielä pieniä toimenpiteitä. Sovelluksen käyttöliittymän lopullisen ilmeen teko ei kuulunut opinnäytetyön vaatimuksiin ja se on vielä tekemättä.

Avainsanat Spring Framework, REST, integraatio, Spring Security, JSON

Sivut 31 s. + liitteet 8 s.

Hämeenlinna
Degree programme in Business Information Technology
System work

Author Ville Pynttäri **Year** 2013

Subject of Bachelor's thesis Searchmashup for the support of the customer service, the design and implementation of the architecture

ABSTRACT

The client of this thesis was the customer service center of Ambientia Oy. This thesis is originated from the need of the client for a tool that integrates search functions of various systems they use. The purpose of the tool is to make the client's working process more efficient. With integrated search, the customer service center is supposed to get all information related to a customer during a phone call. Therefore, the call does not need to be interrupted for searching information.

The purpose of this thesis was to design and develop an application that performs searches for used systems. The application is built on Spring Framework and it integrates search features of JIRA, Confluence and PlanMill. The search Integration is implemented by calling the REST application programming interfaces. This implementation is customised for needs of the customer service center and benefits for other parties are limited.

The result of this thesis is an application which is fully operational in the development environment. The application has been set up on the production environment but there are still some issues to be resolved. The creation of the final user interface appearance was no requirement of the thesis and it has not been done yet.

Keywords Spring Framework, REST, Integration, Spring Security, JSON

Pages 31 p. + appendices 8 p.

SISÄLLYS

TERMIT JA LYHENTEET

Spring Framework	Spring Framework on Java-pohjaisten Web-sovellusten tekoon tarkoitettu sovelluskehikko, joka mahdollistaa olioiden käyttöönoton konfiguraatiotiedostoissa ja niiden joustavan käytön Web-ohjelmassa.
Atlassian JIRA	JIRA on joustavaksi suunniteltu tehtävänhallintajärjestelmä, jonka avulla voidaan seurata meneillään olevien projektien, ominaisuuksia, ongelmia ja tehtäviä. JIRAn vahvuus tehtävänhallinnassa korostuu käsiteltäessä lukuisia sisällöltään pieniä tehtäviä. Yksi JIRA:n eduista on sen hyvä laajennettavuus suuren liitännäisvalikoiman ansiosta.
Atlassian Confluence	Confluence on tiedon hallintaan, vaihtoon ja keräämiseen tarkoitettu Wiki-ohjelmisto, jossa on kiinnitetty huomiota erityisesti yritysmaailman tarpeisiin, kuten käyttöoikeuksien hallintaan ja integroitavuuteen. JIRA:n tapaan myös Confluence on helposti laajennettavissa liitännäisten avulla.
Atlassian Crowd	Web-sovelluksille tarkoitettu keskitetty käyttäjienhallintajärjestelmä.
PlanMill	PlanMill tarjoaa pilvipohjaisia asiakkuudenhallinta-, toiminnanohjaus- ja projektinhallintajärjestelmiä, joista on valittavissa omiin tarpeisiin sopiva paketti.
Jackson	JSON-prosessoija, jolla voidaan muuttaa JSON-muotoista tekstiä Java-olioiksi ja toisinpäin
Apache Maven	Hakemistossa olevan ohjelmointiprojektin hallintatyökalu, jolla voidaan hallita ohjelman kääntöä, raportteja ja dokumentointia.
J2EE	Kerrosarkkitehtuuria tukeva ohjelmistokehitysalusta Java-sovellusten kehitykseen ja suoritukseen.
AspectJ	Java-laajennus, joka tukee piirrelähtöistä ohjelmointia.
Businesslogiikka	Toimintoja tai algoritmeja, joilla hallitaan tiedon välitystä tietokannan ja käyttöliittymän välillä.
Kerrosarkkitehtuuri	Ohjelmistoarkkitehtuurityyli, jolla jaetaan vastuualueita sovelluksen eri toiminnoille.
Enterprise-sovellus	Yrityksissä tai organisaatioissa käytetty sovellus, joka on liian suuri ja monimutkainen yksityisten tai pienten yritysten käyttöön.
POJO	Tulee sanoista Plain Old Java Object ja tarkoittaa tavallista Java-oliota
Bean / papu	POJOsta tehty uudelleenkäytettävä komponentti ohjelmistokehityksessä. Pavut noudattavan tiettyä nimeämiskäytäntöä, jotta niitä voidaan käyttää sovelluksen eri osissa.
EJB	Java-papuhin pohjautuva arkkitehtuuri, joka mahdollistaa transaktionaalisen komponenttipohjaisen Enterprise-sovellusten toteutuksen.
Serialisointi	Tarkoittaa ohjelmarakenteiden tai olioiden muuntoa tallennettavaan muotoon. Deserialisointi on serialisoinnin päinvastainen toiminto.

1 JOHDANTO

Tämän opinnäytetyön aiheena on Ambientia Oy:lle toteutettava Intranet-hakutoiminto, joka yhdistää Ambientian sisäisten järjestelmien hakutoiminnot yhdeksi kokonaisuudeksi. Hakutoiminnon on tarkoitus helpottaa Ambientian asiakaspalvelukeskuksen toimintaa puheluiden aikana siten, että puhelua ei tarvitse keskeyttää tiedonhaun ajaksi, vaan tarvittavat tiedot asiakkaasta voidaan löytää yhdellä haulla.

Ambientia on vuodesta 1996 asti toiminut IT-alan yritys, joka on erikoistunut sähköiseen liiketoimintaan ja viestintään sekä yhteisöllisiin ratkaisuihin. Ambientia tekee konsultointia, suunnittelua ja toteutusta kansainvälisillä markkinoilla ja yhteistyötä yksityisen ja julkisen sektorin toimijoiden kanssa. Ambientia suunnittelee ja toteuttaa räätälöityjä verkkopalveluita ja sovelluksia. Sen erikoisosaamisalueita ovat verkkoliiketoiminnan asiantuntemus ja teknologinen osaaminen ja erilaisten konseptien suunnittelu. (Asiakkaan liiketoiminnan digitalisoiminen 2012.)

Opinnäytetyöaiheeni esiteltiin minulle ollessani työharjoittelussa Ambientialla ja se on saanut alkunsa Ambientian asiakaspalvelukeskuksen järjestelmien hakutoimintoja koskevien kehittämistarpeiden pohjalta. Tässä opinnäytetyössäni toteutetaan haut yhdistävä sovellus. Se ohjelmoidaan käyttäen Spring-sovelluskehystä alustana. Tämä opinnäytetyö on hyvin tekninen, mutta työssäni en käy läpi ohjelmoinnin perusasioita, vaan lukijalta edellytetään niiden tuntemusta. Opinnäytetyöni käytännön osuus käsittelee haut integroivan sovelluksen suunnittelua ja toteutusta. Teoriaosuus kertoo tärkeimmistä sovelluksen toteutukseen liittyvistä asioista.

2 WEB-PALVELUT

2.1 REST

REST on Roy Fieldinin väitöskirjassaan esittelemä arkkitehtuurityyli, joka tähtää yhteensopivuuden säilyttämiseen hajautetuissa hypermediajärjestelmissä, joissa osapuolet kehittyvät itsenäisesti. Lyhenne REST tulee sanoista *Representational State Transfer*. Ensimmäinen sanoista viittaa resurssin esitysmuotoon. Resurssilla tarkoitetaan verkossa olevaa asiaa, johon voidaan viitata URI-osoitteella. Lyhenteen toinen sana State tarkoittaa tilaa, jolla viitataan resurssin ja asiakkaan tiloihin. Viimeisellä sanalla tarkoitetaan tilojen siirtoa asiakkaan ja palvelimen välillä. Käytännössä tämä tarkoittaa, että asiakas pystyy päivittämään palvelimella olevan tiedon haluamakseen tai hankkia itselleen tiedon resurssin tilasta palvelimelta. (REST.)

REST ei itsessään ole arkkitehtuuri, mutta se on joukko rajoitteita, joita noudattamalla järjestelmän suunnittelussa ja toteutuksessa päästään REST-tyyliseen arkkitehtuuriin. REST-arkkitehtuurityylin omaavan järjestelmän täytyy noudattaa rajoitteita. REST-rajoitteisiin kuuluu, että sen on oltava asiakkaan ja palvelimen välinen tilaton järjestelmä. Järjestelmällä ei pitäisi olla tarvetta säilyttää käyttäjien istuntoja, vaan jokaisen pyynnön pitäisi olla toisista riippumaton. REST-järjestelmän täytyy tukea välimuistijärjestelmää. Verkon infrastruktuurin tulisi tukea välimuistia eri tasoilla. Sen käytettävyyden tulee olla yhdenmukaista. Jokaisella resurssilla tulee olla yksilöivä osoite ja voimassa oleva pääsy. Sen täytyy olla kerrostettu ja tukea skaalautuvuutta. Lisäksi REST-järjestelmän tulisi tarjota ohjelmistokoodia tarvittaessa, vaikka tämä on valinnainen rajoite. (REST.)

Nämä rajoitteet eivät pakota käyttämään tiettyä teknologiaa. Ne ainoastaan määrittelevät, kuinka dataa siirretään ja mitä etua ohjesäännöistä on. Vielä tärkeämpänä asiana on se, että uusia teknologioita tai verkkoprotokollia ei tarvitse keksiä, vaan olemassa olevia verkkoinfrastruktuureja voidaan käyttää REST-tyylisten arkkitehtuurien toteutukseen. (Sandoval 2009.)

2.2 HTTP

HTTP-lyhenne tulee sanoista *HyperText Transfer Protocol* ja se on sovellustason protokolla, jolla liikutellaan hypertekstiä Internetissä. Hypertekstillä tarkoitetaan tekstiä, jota ei ole rajoitettu sen lineaariseen lukemiseen, vaan se voi sisältää linkkejä, joilla voidaan liikkua muihin hypertekseihin. (What is Hypertext.)

Ensimmäinen HTTP-versio HTTP/0.9 oli yksinkertainen protokolla raakatiedon välittämiseen Internetissä. HTTP:n 1.0-versio toi protokollaan parannuksen, joka mahdollistaa datatyypin määrittelevien MIME-viestien

käytön. Nykyisen HTTP/1.1-version perustoiminnot ovat samat kuin vanhassa versiossa, mutta uudessa versiossa täytyy täsmentää isäntänimi pyyntöä varten. HTTP/1.1-versioon on myös lisätty aikaisemmasta puuttuvia ominaisuuksia, joista tärkeimpiä ovat sisältöneuvottelu, lohkottaiset siirrot, tavualueet ja tuki välimuistille sekä välityspalvelimille. (Fielding ym. 1999.)

HTTP:n tunnetuin käyttökohde on WWW tai tutummin Web on lyhennelmä sanoista *World Wide Web*. On tärkeää tietää, että Web ei ole synonyymi Internetille, vaan että se on Internetin osa. Web koostuu sivuista ja sivustoista, joita voidaan käyttää selaimella. Palvelut kuten tiedonsiirtoprotokolla FTP, Internet-pelit ja pikaviestintäpalvelu IRC ovat osa Internetiä, mutta ei Webiä. (WWW 2012.)

2.2.1 Request-response-paradigma

HTTP-protokolla perustuu asiakas-palvelinmalliin, jossa asiakas lähettää palvelimelle pyynnön, johon palvelin vastaa. HTTP:ssä jokaista asiakkaan lähettämää pyyntöä kohden tulee yksi vastaus palvelimelta. Käytännössä asiakas ja palvelin kommunikoivat keskenään ainoastaan käytettäessä omalla koneella olevaa palvelinta. Yleensä HTTP-pyynnöt kulkevat yhden tai useamman välityspalvelimen kautta. (Vihavainen & Luukkainen 2012.)

2.2.2 REST-arkkitehtuurityylissä käytetyt HTTP-metodit

RESTissä käytettävät Web-palvelut tarjoavat rajapinnat tiedon luomiseen, hakemiseen, päivitykseen ja poistoon tietokannasta. Näitä rajapintoja kutsutaan nimellä CRUD. RESTissä ainoat sallitut HTTP-metodit ovat GET, POST, PUT ja DELETE. GET-metodin käyttämisen pitäisi olla turvallista ja sillä ei pitäisi olla sivuvaikutuksia, eli sen ei pitäisi muuttaa resurssin tilaa palvelimella. POST-, PUT- ja DELETE metodit taas voivat muuttaa resurssin tilaa, ja näin aiheuttaa sivuvaikutuksia. (Wilde & Pautasso 2011, 117-122.)

GET on yksi HTTP:n yksinkertaisimmista metodeista. Sen päätehtävä on pyytää palvelimelta resursseja. Mikäli pyydetty resurssi on saatavilla, palvelin palauttaa sen käyttäjälle. Resurssi voi olla HTML-sivu, kuva, äänitiedosto jne. Vaikka GET-metodin tarkoituksena on hakea tietoa palvelimelta, se ei tarkoita, että sillä ei voisi välittää tietoa palvelimelle. Mahdollisten merkkien määrä, ja siten tiedon määrä GET-metodin avulla lähetettävissä pyynnöissä on kuitenkin todella rajallinen. Lisäksi kaikki tieto näkyy siinä URI-osoitteessa, eikä siten ole kaikissa tapauksissa turvallista. (GET and POST Method of HTTP 2008.)

PUT-metodi kirjoittaa dokumentteja palvelimelle päinvastaisella tavalla, kuin GET lukee niitä. Jotkut julkaisujärjestelmät antavat käyttäjän luoda

sivuja ja asettaa ne suoraan palvelimelle käyttäen PUT-metodia. Palvelimen toimintatapa, PUT-metodin käsittelyyn on ottaa sen sisältöosuus, ja luoda siitä pyyntö URL-osoitteen mukaan nimetystä uudesta dokumentista. Mikäli osoitteen mukainen dokumentti löytyy, se korvataan uudella sisällöllä. Monet palvelimet vaativat kirjautumisen PUT-metodin käyttöön, koska se voi tehdä muutoksia sisältöön. (Gourley & Totty 2002, 54-55.)

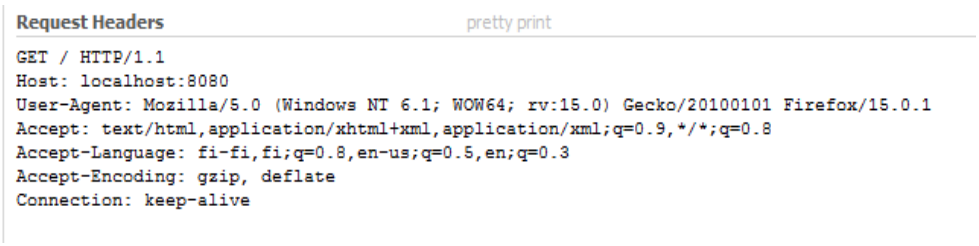
POST-metodi on suunniteltu syötetyn tiedon lähettämiseen palvelimelle, mutta sitä käyttämällä voidaan pyytää palvelimelta tietoa. POST-metodia käytetään usein HTML-lomakkeiden lähettämiseen. (GET and POST Method of HTTP.)

DELETE-metodi tekee sen, mitä sen odotetaankin tekevän. Se pyytää palvelinta poistamaan URL-osoitteen mukaan määritellyn resurssin. Ei kuitenkaan ole taattua, että asiakkaan pyyntö täyttyy. HTTP:n määritelmä mahdollistaa palvelimen yliajaa pyyntö ilman, että asiakkaalle tiedotetaan siitä. (Gourley & Totty 2002, 58.)

2.2.3 Otsakkeet

HTTP-otsakkeet ovat joukko sääntöjä, joilla alustetaan tietynlaista dataa sekä tietoa ja ohjeita. Otsakkeet kulkevat HTTP-pyyntöjen ja vastausten mukana molempiin suuntiin, asiakkaalta palvelimelle ja palvelimelta asiakkaalle. Otsakkeiden muodot vaihtelevat, mutta jokainen niistä sisältää avausrivin, otsakerivejä ja tilarivin. Jokainen otsake tulee olla omalla rivillään, ja jokaisen palvelimen lähettämän vastausotsakkeen ja sisältöosuuden välissä on tyhjä rivi. (Best 2008)

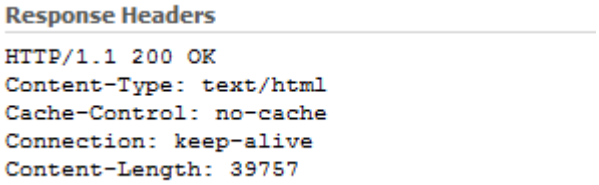
Otin yhteyttä paikallisella koneellani sijaitsevaan Web-palvelimeen ja tarkastelin otsakkeita. Kuvassa 1 pyynnön otsakkeiden ensimmäisellä rivillä näkyy käytettävä HTTP-metodi ja -versio, sekä palvelimelta haettava URI eli tässä tapauksessa palvelimen juuriosoite. Kolmannella rivillä kerrotaan tietoa asiakassovelluksesta eli selaimesta. Neljännellä rivillä kerrotaan millaista tietoa selain hyväksyy. Viides rivi kertoo mitä kieliä selain hyväksyy. Kuudes rivi kertoo hyväksyttävästä pakkauksesta ja viimeinen rivi minkätyyppistä yhteyttä selain suosii.



Kuva 1. Pyyntöotsakkeet

Palvelimen palauttamat otsakkeet näkyvät kuvassa 2. Ensimmäinen otsakerivi kertoo HTTP-version ja statuskoodi 200. Koodi tarkoittaa, että pal-

velin on saanut täytettyä asiakkaan pyynnön. Toisella rivillä kerrotaan millaista sisältöä palvelimen lähettämä vastaus sisältää. Otsakkeiden kolmas rivi kertoo välimuistikäytännöistä ja neljäs rivi yhteystyypistä. Viimeinen rivi kertoo selaimelle, kuinka pitkä vastauksessa tuleva sisältö on.



Kuva 2. Vastausotsakkeet

2.2.4 Evästeet

Evästeet on tämänhetkisistä tavoista paras tapa tunnustaa käyttäjiä ja toteuttaa pysyvät istunnot. Ne eivät kärsi monista ongelmista, jotka olivat ongelmallisia vanhemmissa tekniikoissa, mutta näitä tekniikoita usein käytetään evästeiden kanssa tuomaan lisäarvoa. Evästeet ovat alunperin Netscapen kehittämiä, mutta nykyään niitä tukevat kaikki tunnetut selaimet.

Evästeet voidaan pääosin luokitella kahteen eri ryhmään: istuntoevästeisiin ja pysyvyysevästeisiin. Istuntoeväste on väliaikainen eväste, joka pitää kirjaa asetuksista ja mieltymyksistä, kun käyttäjä selaa sivua. Istuntoeväste poistetaan, kun käyttäjä sulkee selaimen. Pysyvyysevästeet voivat olla voimassa istuntoevästeitä pidempään. Ne tallennetaan käyttäjän kovalevylle, ja pysyvät siellä, vaikka käyttäjä sulkee selaimen tai käynnistää tietokoneen uudelleen. Pysyvyysevästeitä käytetään usein säilyttämään asetusprofiili tai käyttäjätunnus sivustolle, jolla käyttäjä vierailee ajoittain. (Gourley & Totty 2002.)

Selaimilla voi olla varastossaan satoja tai tuhansia evästeitä. Selaimet eivät kuitenkaan lähetä kaikkia evästeitä jokaiselle sivulle, vaan tyypillisesti vain kaksi tai kolme evästettä sivua kohden. Kaikkien evästeiden sisältämän datan lähettäminen hidastaisi palvelua dramaattisesti. Selaimet siirtäisivät enemmän tietoa evästeinä kuin itse sisältöä. Suurin osa evästeistä olisi myös täysin olematonta tietoa suurimmalle osalle sivustoista, koska niillä on palvelinkohtaiset nimi-arvoparit. Lisäksi kaikkien evästeiden lähettäminen kaikille sivuille loisi potentiaalisen tietoturvariskin, koska tietoa luotetuilta sivustoilta voisi päätyä ei luotetuille sivustoille. Pääasiallisesti selain lähettää palvelimille ne evästeet, jotka on saatu kyseiseltä palvelimelta. (Gourley & Totty 2002.)

2.3 JSON

JSON on lyhenne sanoista *JavaScript Object Notation*. JSON on kevyt tiedonsiirtomuoto, joka on helppolukuinen ihmiselle, mutta lisäksi tietokoneiden on helppo luoda ja jäsentää sitä. JSON on täysin ohjelmointikielestä riippumaton tekstimuoto, mutta se tukee käytäntöjä, jotka ovat tuttuja C-perheen kieliä käyttäville ohjelmoijille. Edellä mainittujen ominaisuuksien vuoksi JSON on ihanteellinen kieli tiedon siirtoon. JSON koostuu kahdenlaisista rakenteista, jotka ovat kokoelma nimi-arvopareista ja järjestetty lista. Nimen ja arvon pari vastaa ohjelmoinnin käsitteitä, kuten *object, record, struct, dictionary, hash table, keyed list*, tai *associative array. Järjestetty lista taas vastaa käsitteitä, kuten array, vector, list*, tai *sequence*. JSON-kielen rakenteet ovat kuvattu alla olevassa taulukossa 1. (Introducing JSON.)

Taulukko 1. JSON-rakenteet

Nimi-arvoparit	Järjestetty lista
Olio on järjestelemätön sarja nimi-arvopareja. Olio alkaa vasemmalla aaltosulkeella ({) ja loppuu oikeaan aaltosulkeeseen (}). Jokaista nimeä seuraa kaksoispiste (:) ja nimi-arvoparit erotetaan pilkuilla (,).	**Taulukko** on järjestelty kokoelma arvoja. Taulukko alkaa vasemmalla hakasulkeella ([ ) ja loppuu oikeaan hakasulkeeseen ( ]). Taulukon arvot erotellaan pilkuilla (,). Taulukon arvoksi voidaan laittaa merkkijono, numero, olio, taulukko, totuusarvomuuttuja tai null-arvo.
Merkkijono on lainausmerkkien sisälle paketoitu sarja, joka voi sisältää nollan tai enemmän Unicode-merkkejä. JSON-merkkijonot ovat hyvin samanlaisia kuin Javan ja C:n merkkijonot.	
Numero pitkälti samanlainen kuin Javan tai C:n numero, mutta ei käytä oktaali- ja heksadesimaalimuotoja.	

Alla esimerkki yksinkertaisesta JSON-oliosta, joka sisältää kaksi merkkijonoa ja olion, joka taas sisältää taulukon olioita. Taulukon oliot sisältävät kaksi merkkijonoa.

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

3 YKSIKKÖTESTAUS

Yksikkötestit ovat ohjelmistokehittäjän kirjoittamia koodinpätkiä, jotka testaavat hyvin pienen ja tarkan osan ohjelmakoodia. Yleensä yksikkötestit testaavat, jonkin tietyn metodin toimintaa tietyssä yhteydessä. Voidaan esimerkiksi lisätä arvojen mukaan järjestettyyn listaan suuri arvo, ja tarkastaa yksikkötestillä, että arvo on listalla, sen lopussa. Yksikkötestejä tehdessä ei vielä tiedetä, onko testin tulos sitä, mitä asiakas tai loppukäyttäjä siltä toivoo. Ne ainoastaan tehdään todistamaan, että testattava koodinpätkä tekee tosiaan, mitä kehittäjä haluaakin sen tekevän. Yksikkötestaus parantaa koodin laatua ja vähentää virheiden etsimiseen käytettyä aikaa. (Hunt & Thomas 2007, 3-5.)

3.1 Testivetoinen kehitys

Testivetoisen kehityksen periaate on yksinkertainen. Vanhemmissa kehitystavoissa ohjelmointi aloitettiin tekemällä sovellusikkuna tai Web-sivu käyttöliittymäksi, jolla koodin toimintoja tarkasteltiin. Testivetoisessa kehityksessä ohjelmointi aloitetaan kirjoittamalla yksikkötestiä. Tämä tunnetaan testi ensin -menetelmänä, ja se voi alkuun tuntua hieman kiusalliselta. Kirjoittamalla testit ensin kehittäjä luo vaatimukset, joiden mukaan ohjelmakoodi suunnitellaan. Testivetoisen kehityksen etuja on, että se takaa laadukkaan koodin alusta alkaen, rohkaisemalla kehittäjiä kirjoittamaan ainoastaan vaatimuksen täyttävän määrän, eikä yhtään enempää. Lisäksi se takaa korkean vertailuasteen koodin ja vaatimusten välillä. Jos vaatimukset on kirjoitettu testeinä ja kaikki menevät läpi, kehittäjä voi korkealla itseluottamuksella sanoa, että koodi vastaa tarpeita. Testivetoinen kehitys myös helpottaa sovelluksen päivitettävyyttä. Koodin muuttuessa olemassa olevat testit estävät nykyisen toiminnallisuuden vahingoittumisen. (Bender & McWherter 2011, 8-10.)

Ensimmäinen huomio aloittaessa kehitystä testi ensin -periaatteella on, että testi epäonnistuu. Tämä tapahtuu, koska testi yrittää luoda ilmentymää luokasta, jota ei ole määritelty tai käyttää objektin metodia, jota ei ole vielä olemassa. Ensimmäinen vaihe on yksinkertaistettuna luoda testattava luokka ja sen metodit. Tässä vaiheessa testit edelleen epäonnistuvat, koska ne ovat olemassa, mutta eivät vielä tee mitään. Seuraava vaihe on kirjoittaa koodia täsmälleen niin paljon, että testi menee läpi. Sen pitäisi olla yksinkertaisin mahdollinen ratkaisu, jolla testi saadaan läpäistyä. Ajatus on olla kirjoittamatta koodia, jota saatetaan tarvita, mutta ei välttämättä tulla koskaan tarvitsemaan. Tämä estää turhan monimutkaisen koodin kirjoittamisen tilanteissa, joissa yksinkertaisempikin ratkaisu on kelvollinen. Yksi testivetoisen kehityksen tavoitteista onkin helposti ymmärrettävän ja ylläpidettävän koodin kirjoittaminen. (Bender & McWherter 2011, 8-10.)

Kun ensimmäisen testi on mennyt läpi, seuraava vaihe on kirjoittaa niitä lisää samalle toiminnolle. Testejä pitäisi olla niin kattavasti, että niillä pystytään tarkastamaan kaikki toiminnolle asetetut vaatimukset. Tämä pitää

sisällään metodien testaamisen monilla syöttöarvoilla. Myös sallittujen rajojen ulkopuolelle menevät arvot tulee testata. Esimerkiksi testattaessa metodia, joka ottaa vastaan merkkijonon, olisi hyvä testata, mitä tapahtuu, kun sille annetaan tyhjä arvo. Yleensä tällaisissa tilanteissa paluuarvona tulee jokin virheilmoitus. Kun kaikki järkevyyden rajoissa olevat mahdolliset tilanteet on testattu, on toiminto valmis. (Bender & McWherter 2011, 8-10.)

3.2 Yksikkötestauksen sovelluskehykset

Yksikkötestauksen sovelluskehykset ovat ohjelmointityökaluja, jotka sisältävät perustan yksikkötestien rakentamiseen, toiminnallisuuden testien ajamiseen ja työkalut testitulosten raportoimiseen. Ne eivät ole ainoastaan testaustyökaluja, vaan ne toimivat myös sovelluskehityksessä esikääntäjinä ja virheiden jäljittäjinä. Sovelluskehykset voivat olla osallisina lähes jokaisessa sovelluskehityksen vaiheessa, kuten arkkitehtuurin ja ulkoasun suunnittelussa, ohjelmistokoodin kirjoituksessa, virheiden korjauksessa, suorituskyvyn optimoinnissa ja laadun varmistuksessa. (Hamill 2005, 1.)

3.2.1 JUNIT

JUNIT on Java-ohjelmoinnissa käytetty yksikkötestauksen sovelluskehys. Se on epäilemättä laajimmin käytetty, laajennetuin ja puhutuin sovelluskehys tämän päivän yksikkötestauksessa. JUNIT on myös perustana monille erikoistuneimmille sovelluskehyksille. Sen käyttäjien suuri yhteisö tarkoittaa, että JUNIT on perustana monille uusille ideoille ja kehitysmahdollisuuksille yksikkötestausteknologiassa. Java on suunniteltu alusta asti todelliseksi olio-ohjelmointikieleksi, joka sisältää monia moderneja ominaisuuksia, kuten täysin abstraktit luokat, olioiden reflektointi ja sisäänrakennettu virheiden hallinta. JUNIT kykenee käyttämään täydellisesti näitä ominaisuuksia. (Hamill 2005.)

3.2.2 Mockito

Mockito on avoimen lähdekoodin testaussovelluskehys Java-ohjelmointikielelle. Mockito-sovelluskehys automatisoi yksikkötestejä testivetoisen ohjelmointitavan käytössä irrottamalla testikoodin järjestelmästä. Tämä tapahtuu poistamalla kaava, jossa ennen testien ajamista asetetaan olettamuksia. Mockito-sovelluskehys tarjoaa yksinkertaisemman ja intuitiivisemman tavan, jossa kysytään ajon aikana tapahtuneista asioista jälkeenpäin. Näin voidaan tarkastaa vain halutut tapahtumat. Mallissa, jossa asetetaan oletukset ennen ajoa, joudutaan usein tarkastelemaan epäolennaisia tapahtumia. (Crisper 2009, 12; Faber 2010.)

4 SPRING-SOVELLUSKEHYS

Spring-sovelluskehys on kevyt ratkaisu enterprise-sovellusten tekoon Java-ohjelmointikielellä. Spring tarjoaa kattavan infrastruktuurin, jotta kehittäjä voi keskittyä sovelluksen ohjelmointiin. Spring tähtää sovelluskehyksellään helpompaan J2EE-ohjelmointiin ja hyvien ohjelmointitapojen tukemiseen. Tämä saavutetaan mahdollistamalla Web-sovelluksen rakentaminen POJO-luokista, mikä onkin Spring-sovelluskehyksen pääasiallinen tarkoitus. Sovelluskehys piilottaa kehittäjältä paljon monimutkaisia toimintoja, ja auttaa näin valitsemaan yksinkertaisia toteutuksia, mikä on todella arvokasta. (Johnson ym. 2012, 1; Johnson 2005.)

Spring-sovelluskehys on modulaarinen, ja käyttäjä voi ottaa käyttöönsä vain ne osat, mitä hän tarvitsee. Spring ei keksi pyörää uudelleen, ja siksi se ei sisällä itsessään esimerkiksi logging- tai connection pool- järjestelmiä. Näistä järjestelmistä on jo olemassa avoimen lähdekoodin projekteja, joiden käytön helpottamiseen Spring tähtää. Spring ei myöskään halua suoraan kilpailla muiden avoimen lähdekoodin sovelluskehysten kanssa, mikäli se ei usko pystyvänsä tarjoamaan jotain uutta. (Johnson ym. 2012, Johnson 2005.)

Spring toimii monilla sovelluspalvelimilla, kuten esimerkiksi WebLogic, Tomcat, Resin, JBoss, Jetty, Geronimo ja WebSpehere. Spring pyrkii välttämään kaikkia standardoimattomia ja alustariippuvaisia ratkaisuja. Spring-soveluskehyksen POJO-lähestymistapa mahdollistaa ympäristökohtaisten toimintojen käytön ilman, että siirrettävyys kärsii. (Johnson 2005.)

4.1 IoC-säiliö

Termi IoC tulee sanoista *Inversion of Control*, ja se on ohjelmointitekniikka, jossa sovelluksen tapahtumaketju on käänteinen. Tavallisesti olion kutsuja päättää, miten oliota käytetään, mutta IoC-tekniikassa kutsuttava olio päättää, miten ja milloin kutsujalle vastataan. Tällä tapaa kutsuja ei ole vastuussa sovelluksen pääasiallisesta tapahtumaketjusta. (Goenka 2011.)

Spring-sovelluskehyksen tapauksessa IoC-termi yhdistetään sovelluskehyksen IoC-säiliöön, josta käytetään myös termiä riippuvuuksien injektointi. Se on prosessi, jossa oliot määrittelevät itselleen riippuvuussuhteet. Riippuvuussuhteilla tarkoitetaan muita olioita, joiden kanssa kyseinen olio työskentelee. Spring-sovelluskehyksen IoC injektoi olioille riippuvuudet samalla, kun se luo niistä pavut. (Johnson ym. 2012, 34.)

Spring-sovelluskehyksen ydin on org.springframework.beans-paketti, joka on tarkoitettu työskentelemään Java-papujen kanssa. Sovelluskehittäjä ei tavallisesti käytä tätä pakettia suoraan, mutta se pitää sisällään paljon sovelluskehyksen toimintoja. Seuraavaksi korkein abstrakti kerros on paputehdas eli bean factory. Se on geneerinen tehdas, joka mahdollistaa olioi-

den hakemisen niiden nimen perusteella. Lisäksi se hallitsee olioiden välisiä suhteita. Paputehdas tukee kahdenlaisia olioita, joita ovat singleton ja prototype. Singleton tässä tapauksessa tarkoittaa nimettyä oliota, josta on vain yksi jaettu instanssi. Se on yleisin oliomuoto ja myös oletusasetus oliolle. Singleton on omiaan esimerkiksi tilattomille palvelukerroksen olioille. Prototype on oliomuoto, josta luodaan oma itsenäinen olio hakijalle jokaisessa kerralla, kun sitä haetaan. (Johnson 2005.)

Paputehdas tarjoaa perustoiminnot ja asetusrajapinnan, mutta yksityiskohtaisempaa toiminnallisuutta saadaan ApplicationContext-rajapinnalla. Tämä rajapinta edustaa Spring IoC -säiliötä ja se on vastuussa papujen instansoinnista, määrittelyistä ja asemoinnista. ApplicationContext on paputehtaan alirajapinta ja se tarjoaa paputehdasta helpomman integraation Spring-sovelluskehyksen AOP-toimintojen kanssa, jotka ovat selitetty tarkemmin aliluvussa 4.2. (Johnson ym. 2012, 152.)

4.2 Spring AOP

AOP tarkoittaa piirrepohjaista ohjelmointia. Se on lyhenne sanoista *Aspect Oriented Programming*. Piirrepohjainen ohjelmointi täydentää olio-ohjelmointia siten, että se tuo siihen toisenlaisen tavan ajatella ohjelman rakennetta. Olio-ohjelmoinnissa modulaarisuuden toteuttamisessa avainyksikkö on luokka, kun taas piirrepohjaisessa ohjelmoinnissa se on aspekti. Aspektit mahdollistavat ominaisuuksien modularisoinnin asioissa, kuten esimerkiksi monentyyppisten olioiden kanssa toimiva transaktioiden hallinta. Toisin sanoen piirrepohjaisella ohjelmointitavalla sovelluksen läpileikkaavat ominaisuudet saadaan koottua samaan paikkaan. Läpileikkaavilla ominaisuuksilla tarkoitetaan toimintoja, joita käytetään sovelluksen eri luokissa ja sovelluskerroksissa. (Salo 2007; Johnson ym. 2012, 193.)

AOP on yksi Spring-sovelluskehyksen pääkomponentteja. Vaikka sovelluskehikon IoC-säiliö ei ole riippuvainen AOP-komponentista, se tukee IoC-komponenttia olemalla varsin toimiva väliohjelmisto. Spring AOP on tehty vastaamaan enterprise-sovellusten haasteisiin. (Johnson ym. 2012.)
Sen tarkoitus on tarjota helppoja ratkaisuja yleisimpien poikkileikkaavien ominaisuuksien modularoimiseen. Spring-sovelluskehyksen 2.0-version, mukana tuli kaksi uutta tapaa tehdä poikkileikkauksia. Näitä olivat skeemapohjainen poikkileikkaus ja AspectJ-annotaatiot. Spring-sovelluskehyksen AOP ei tavoittele täydellistä piirreohjelmointiympäristön asemaa, vaan se on enemmänkin sovelluskehyksen lisätyökalu. Sen tavoitteena on tarjota yksinkertainen ratkaisu aspektien käytön yleisimpiin tapauksiin. (Sorsa 2008.)

4.3 Spring MVC

Spring-sovelluskehyksen Web-rajapinta on nimeltään model-view-controller eli MVC. Se on rakennettu sovelluskehikon DispatcherServlet-luokan ympärille, joka on pyyntöjä käsittelijöille lähettävä luokka. Spring MVC auttaa rakentamaan löyhästi yhdistettyjä Web-sovelluksia. MVC-suunnittelumalli auttaa kehittäjää erottamaan toisistaan businesslogiikan, esityslogiikan ja navigaatiologiikan. Spring MVC:n Model on vastuussa tiedon kapseloinnista. Näkymä näyttää HTTP-vastauksen käyttäjälle mallin avustuksella. Kontrolleri vastaa pyynnön vastaanottamisesta käyttäjältä ja taustapalveluiden kutsumisesta. Taulukossa 2 on kuvattu tapahtumaketju, joka kertoo mitä tapahtuu, kun pyyntö lähetetään Spring MVC -sovelluskehykselle. (Muthuraman 2012.)

Taulukko 2. Spring MVC -tapahtumaketju

1.	Dispatcher Servlet vastaanottaa pyynnön
2.	Dispatcher Servlet konsultoi HandlerMapping-luokkaa ja herättää pyyntöön yhdistetyn kontrollerin.
3.	Kontrolleri prosessoi pyynnön kutsumalla asianmukaisia palvelukerroksen metodeja ja palauttamalla ModelAndView-olion DispatcherService-oliolle. ModelAndView olio sisältää mallin tiedot ja näkymän nimen.
4.	DispatcherServlet lähettää näkymän nimen ViewResolver-oliolle, joka etsii varsinaisen näkymän.
5.	DispatcherServlet lähettää malliolion näkymälle.
6.	Mallin tiedot sisältävä näkymä esittää tulokset käyttäjälle.

4.4 Spring Security Core

Spring Security tarjoaa kattavat tietoturvapalvelut J2EE-tekniikkaan perustuville enterprise-sovelluksille. Spring-sovelluskehys on johtava J2EE-ratkaisu, ja Spring Securityn kohdalla tälle sovelluskehykselle on annettu erityinen painoarvo. Spring Security -sovelluskehystä käytetään monista syistä, mutta monien kiinnostus projektia kohtaan herää niiden löydettyään enterprise-sovelluksessa tarvittavia ominaisuuksia, joita ei tarvittavalla tasolla löydy J2EE Servlet -spesifikaatiosta tai EJB -spesifikaatiosta. (Alex & Taylor 2013.)

Kaksi merkittävää asiaa sovelluksen tietoturvassa on käyttäjän tunnustautuminen ja oikeuksien hallinta. Näiden kahden asian hallintaan Spring Security tähtää. Tunnistautuminen on prosessi, jossa näytetään toteen asiakkaan olevan se, kuka se väittää olevansa. Oikeuksien hallinta viittaa pro-

sessiin, jossa päätetään, onko asiakkaalla oikeus tehdä yrittämiään toimenpiteitä sovelluksen sisällä. Oikeuksien tarkastamisvaiheeseen saapuminen edellyttää, että asiakas on jo tunnistautunut sovellukseen. (Alex & Taylor 2013.)

Tunnistautumisen osalta Spring Security tukee laajaa valikoimaa tunnistautumismalleista. Monet näistä ovat tulleet kolmansilta osapuolilta, mutta Spring Security tarjoaa kokoelman omia tunnistautumistoimintoja. Esimerkkejä Spring Security -sovelluskehyksen tukemista tunnistautumismahdollisuuksista on kuvattu taulukossa 3. (Alex & Taylor 2013.)

Taulukko 3. Spring Security -sovelluskehyksen tukemia tunnistautumisia

HTTP BASIC	Yksinkertainen standardi, jossa tunnus ja salasana asetetaan HTTP-pyynnön otsakkeisiin
HTTP DIGEST	Tukee hajautusalgoritmin käyttöä tunnistautumisotsakkeissa, joka tekee siitä turvallisemman tunnistautumistavan, kuin HTTP BASIC
HTTP X.509	Asiakassertifikaatin vaihto
LDAP	Erittäin yleinen lähestymistapa alustariippumattomien järjestelmien tunnistautumistarpeisiin
Lomakepohjainen tunnistautuminen	Yleinen yksinkertaisissa järjestelmissä
OpenID	Avoin standardi, joka mahdollistaa käyttäjän tunnistautumisen Internetissä ilman, että käyttäjän pitää luoda uutta tunnusta

5 HAKUMASHUP-SOVELLUKSEN ALKUVALMISTELUT

Tämän opinnäytetyön käytännön osuudeksi toteutettava Hakumashup on sovellus, joka on tarkoitettu helpottamaan APK:n eli Ambientian asiakaspalvelukeskuksen toimintaa erityisesti puhelun aikana. Hakumashupin avulla APK:n päivystäjän pitäisi pystyä puhelun aikana selvittämään asiakkaan kysymys niin, ettei asiaan tarvitsisi palata myöhemmin. Puhelun aikana sovellukseen syötetään hakusana, jonka perusteella se hakee tietoa Ambientian sisäisistä järjestelmistä niiden rajapintojen kautta. Yleisempiä käytettäviä hakusanoja ovat asiakkaan nimi, asiakasyrityksen nimi, verkkopalvelun URL-osoite tai projektinumero.

Hakumashupissa yhdistettävät järjestelmät ovat Ambientian JIRA, Sisäinen Confluence, Asiakkaiden Confluence ja Ambientian PlanMill. Hakumashup-sovelluksen tärkeimmät haut on listattu alla olevassa taulukossa 4.

Taulukko 4. Hakumashup-sovelluksen tärkeimmät haut

1.	Salasanojen ja tunnusten haku sisäisestä Confluencesta asiakkaan palvelun nimen tai WWW-osoitteen perusteella.
2.	Asiakkaan aikaisempien tehtävien haku JIRAsta asiakkaan sähköpostin perusteella.
3.	Asiakkaiden Confluencesta ryhmien haku, mihin kyseinen asiakas kuuluu.
4.	Asiakkaan avoimien projektien haku PlanMillista.

Hakumashupin käyttöliittymäksi halutaan yksi sivu, jossa tarkastellaan kaikkia hakutuloksia samanaikaisesti. Hakutulosten näkymien on tarkoitus olla mahdollisimman yhteneväisiä keskenään.

5.1 Sovelluksen arkkitehtuurin suunnittelu

Hakumashup-sovelluksen arkkitehtuuri koostuu kolmesta sovelluskerroksesta ja neljännestä kerroksesta eli tietolähteestä. Luokkarakenne suunniteltiin koostumaan yhdestä kontrollerista, kahdesta näkymästä, omista palvelukerroksen luokista haussa käytettäville järjestelmille ja hakutulosten tietoja säilyttävistä luokista.

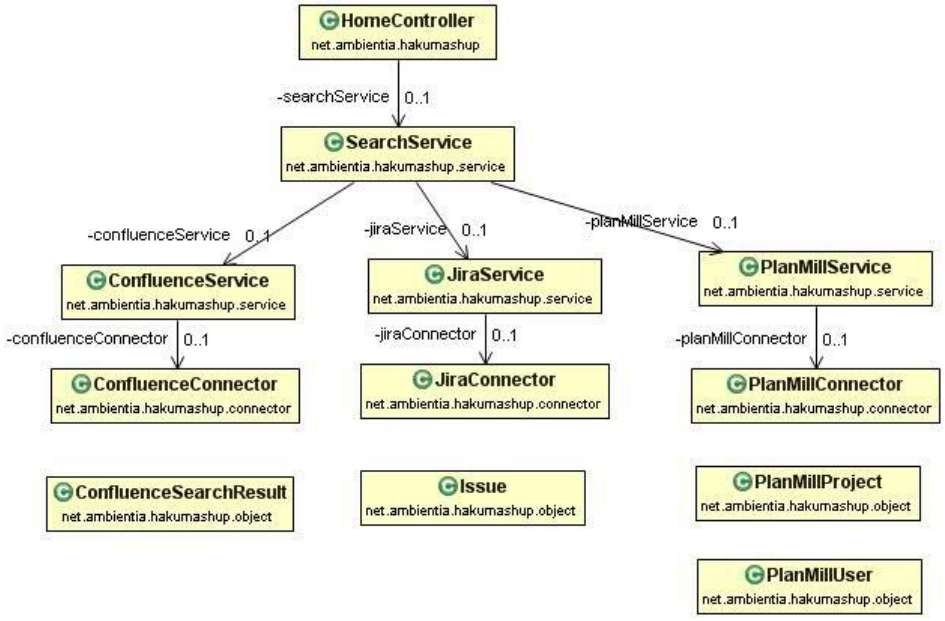
Ensimmäinen kerros on käyttäjää lähimpänä oleva näkymä eli HTML-sivupohja. Tässä kerroksessa on kaksi eri sivua, kirjautumissivu ja varsinainen hakusivu.

Toinen kerros koostuu yhdestä ainoasta kontrolleriluokasta. Sovelluksen on tarkoitus toimia yhdessä päänäkymässä, joten useammille kontrollereille ei ole tarvetta. Kontrolleri ohjaa käyttäjän hakusivulle, kun oikeat kirjautumistiedot vahvistetaan. Kontrolleri ottaa vastaan hakusanan parametrina, ja tekee pientä analysointia hakusanalle, jonka perusteella kutsutaan palvelukerroksen toimintoja. Palvelukerrokselta vastauksena saadut tiedot kontrolleri lähettää näkymälle.

Kolmannessa eli palvelukerroksessa on luokka jokaiselle Hakumashupissa käytettävälle järjestelmälle. Luokat sisältävät toimintalogiikan sille, miten hakusanasta muodostetaan HTTP-kysely järjestelmien REST-rajapinnoille ja, miten saadut vastaukset käsitellään.

Lisäksi sovelluksessa on POJO-luokkia tiedon käsittelyn avuksi. Nämä luokat toimivat väliaikaisina tallennusmuotoina hakusanalle ja hakutuloksille. Tarpeen tullen voidaan myös eristää monissa luokissa käytettäviä apumetodeita apuluokkiin.

Kuvassa 3 esitetty valmiin sovelluksen arkkitehtuuri vastasi hyvin pitkälti suunnitellun mukainen. Ainoa merkittävä ero suunniteltuun arkkitehtuuriin on palvelukerroksen luokkien suurempi lukumäärä. Yksikkötestausta varten oli tietolähteeseen yhteydessä oleva osa eristettävä omaan luokkaansa. Jokaista integroitavaa järjestelmää kohden tuli yksi lisäluokka. Integroitaville järjestelmille viestejä lähettävät metodit oli eristettävä omiin luokkiinsa, jotta niistä saatiin tehtyä Mock-oliota. Jatkokehityksen kannalta on tärkeää, että sovelluksessa on kunnollinen testipino, joka on riippumaton työn kehityksessä käytetystä ympäristöstä.



Kuva 3. Hakumashup sovelluksen luokkarakenne

5.2 Kehitysympäristön asennus ja konfigurointi

Sovelluksen kehitysvaiheessa ei tietoturvasyistä voitu testata hakutoimintoja tuotantoympäristössä. Virheellinen sovellus olisi voinut haitata haussa käytettäviä järjestelmiä tai pahimmassa tapauksessa voinut kaataa niitä. Lisäksi jokainen järjestelmä vaatii käyttöoikeuden, joita ei ollut saatavilla kehitysvaiheessa.

PlanMill-palvelulla oli olemassa valmis testiympäristö, joka sisältää valmista testidataa. Atlassianin palveluiden testausta varten helpoin tapa oli pystyttää testiympäristö paikalliselle tietokoneelle ja luoda testidata sinne. Kaikki JIRA-, Confluence- ja Crowd-versiot ovat saatavilla myatlassian.com-sivustolla sovelluskehittäjän lisenssin omaavalle henkilölle. Selvitin käytössä olevat versiot palveluista ja asensin ne, jotta välttyisin käyttöönottovaiheessa mahdollisista versioiden eroavaisuuksista johtuvilta ongelmilta.

Atlassianin järjestelmien asennus paikalliselle koneelle on yksinkertaista, ja jokaisen järjestelmän asennus seuraa samaa kaavaa. Puretaan standalone-paketti kovalevylle ja ajetaan skripti bat-tiedostosta, jolloin skripti käynnistää tomcat-palvelimen ja kyseisen järjestelmän. Ensimmäisellä käynnistyskerralla järjestelmä kysyy lisenssiä ja tietokanta-asetuksia. Tässä tapauksessa oletusasetukset olivat riittävät.

5.3 Crowdin käyttäjätilit JIRAssa ja Confluencessa

Ensimmäinen vaihe Crowd-käyttäjätilien käyttöönotossa JIRAssa ja Confluencessa on lisätä ne Crowdin sovelluslistaan sen ylläpitovalikossa. Lisäksi sovelluksille täytyy määritellä käyttäjäkansiot ja käyttäjäryhmät.

JIRA ja Confluence on suunniteltu toimimaan helposti yhdessä Crowdin kanssa. Käytössä olevassa JIRA-versiossa Crowdin käyttöönotto tapahtuu lisäämällä Crowdin käyttäjäkansio suoraan JIRAn ylläpitovalikossa, minkä jälkeen voidaan tuoda käyttäjiä ja käyttäjäryhmiä Crowdista.

Uudemmissa Confluence-versioissa on myös mahdollista tehdä Crowd-kytkös ylläpitovalikossa, mutta käytössä olevassa versiossa Crowd-kytkös on tehtävä lisäämällä ja muuttamalla tiedostoja Crowdin hakemistoissa.
Toimiakseen Confluence tarvitsee Crowdin hakemistosta omaansa kopioitavat välimuistiasetustiedoston ja asiakasohjelmakirjaston. Kopioitavien tiedostojen sijainti- ja kohdehakemistot näkyvät alla olevassa taulukossa 5.

Mistä kopioidaan	Minne kopioidaan
CROWD/client/crowd-integration-client-X.X.X.jar	CONFLUENCE/confluence/WEB-INF/lib
CROWD/client/conf/crowd.properties	CONFLUENCE/confluence/WEB-INF/classes
CROWD/client/conf/crowd-ehcache.xml	CONFLUENCE/confluence/WEB-INF/classes/crowd-ehcache.xml

Taulukko 1. Confluencen Crowd-kytköksen vaatimat tiedostot

Lisäksi Confluence pitää määrittää käyttämään asiakasohjelmakirjastoa. Tämä tapahtuu lisäämällä tarvittava rivi tiedostoon, jonka polku on: CONFLUENCE/confluence/WEB-INF/classes/atlassian-user.xml. Alla oleva rivi ottaa kirjaston käyttöön.

```
<crowd key="crowd" name="Crowd Repository"/>
```

Asetustiedostossa määritellään Crowdin tunnus, salasana, osoite ja muut asetukset. Alla olevassa esimerkissä näkyy asetukset, jossa Crowd on sa-

malla koneella Confluencen kanssa. Crowdiin on lisätty Confluencelle samanniminen ohjelma, jonka salasana on admin.

```
session.lastvalidation=session.lastvalidation
session.isauthenticated=session.isauthenticated
application.password=admin
application.name=confluence
session.validationinterval=0
crowd.server.url=http://localhost:8095/crowd/services/
session.tokenkey=session.tokenkey
application.login.url=http://localhost:8080/
```

Confluencen seraph-config.xml-tiedostosta kommentoidaan Confluencen oma todentaja pois käytöstä. Tiedostoon lisätään rivi, joka käskee Confluencen käyttämään Crowd-todentajaa.

```
<!-- <authenticator class=
"com.atlassian.confluence.user.ConfluenceAuthenticator" />
-->

<authenticator class=
"com.atlassian.crowd.integration.seraph.ConfluenceAuthentica
tor"/>
```

5.4 Hakumashup-sovelluksen riippuvuudet

Hakumashup-sovellus on riippuvainen useista lisäosista. Kaikki sovelluksen käyttämät lisäosat haetaan Apache Maven -työkalun avulla. Käytännössä lisäosien lataaminen tapahtuu määrittelemällä ne riippuvuuksiksi pom.xml-tiedostossa, jolloin Maven osaa ladata ne tiedostolähteistä. Sovelluksen käytössä on Atlassianin, Spring Sourcen, JBossin ja Apache Mavenin tiedostolähteet. Riippuvuuksien joukossa on myös IDE:n automaattisesti lisäämiä riippuvuuksia, joita Spring-sovellus tarvitsee.

5.5 Spring security coren käyttöönotto

Spring Security Core -lisäosan liittäminen projektiin tapahtuu lisäämällä rivit pom.xml-tiedostoon, jotka kertovat projektin riippuvan lisäosasta. Tämän perusteella Maven osaa ladata lisäosan tiedostolähteistä projektihakemistoon. Spring Security Coren toiminnallisuus määritellään sille tehtävissä xml-asetustiedostoissa. Hakumashup-sovellukseen ei tulevaisuudessa tehdä omaa käyttäjienhallintaa, vaan se hoidetaan Crowd-integraation avulla. Integraatio vaatii omanlaisensa asetukset, joten Spring Security Coren käyttöönotto aloitetaan suoraan integraation vaatimilla asetuksilla.

5.6 Crowd - Spring Security Core -integraatio

Integraation ensimmäinen vaihe on tehdä Crowd tietoiseksi sovelluksesta, joka yrittää yhteyttä Crowdiin, eli tässä tapauksessa Hakumashupista. Tämä tapahtuu lisäämällä Hakumashupin Crowdin ohjelmalistaan samalla tavalla, kuten sovellukset alaluvussa 5.3.1. Samaan tapaan sovellukselle tulee määrittää sovellukselle näkyvät kansiot ja käyttäjäryhmät, jotka saavat kirjautua sovellukseen.

Integraation avuksi ohjelmakoodipuolelle on saatavilla crowd-integration-client -lisäosa. Lisäosan voi asentaa manuaalisesti siirtämällä jar-tiedoston projektikansioon, mutta tässä tapauksessa lisäosa haetaan Atlassianin Maven ohjelmistolähteistä lisäämällä pom.xml-tiedostoon rivit:

```
<dependency>
   <groupId>com.atlassian.crowd</groupId>

    <artifactId>
      crowd-integration-springsecurity
    </artifactId>

   <version>${crowd.version}</version>

   <scope>runtime</scope>

</dependency>
```

JIRAn ja Confluencen tapaan integroitava sovellus tarvitsee välimuistitiedoston, joka kopioidaan Crowdin kansiosta projektikansioon. Lisäksi tarvitaan crowd.properties-asetustiedosto, joka löytyy oletusasetuksilla myös Crowdin kansiosta. Asetustiedostossa on samat asiat, kuin Confluencen vastaavassa. Myöhemmässä kehitysvaiheessa nimesin asetustiedoston uudelleen app.properties-nimiseksi, koska sovellukseen tarvittiin muitakin pysyviä asetuksia. Asetustiedosto on nähtävissä kokonaisuudessaan opinnäytetyön liitteessä 5. Yhden asetustiedoston hallitseminen ja ylläpitäminen on tässä tapauksessa helpompaa, koska tiedoston sisältö pysyy pienenä. Tiedostojen lähde- ja kohdehakemistot ovat merkitty alla olevaan taulukkoon 6.

Taulukko 2. Hakumashup-Crowd integraation vaatimat tiedostot

Mistä kopioidaan	Minne kopioidaan
CROWD/client/conf/crowd-ehcache.xml	Hakumashup/WEB-INF/classes/crowd-ehcache.xml
CROWD/client/conf/crowd.properties	Hakumashup /src/main/java/app.properties

On olemassa kaksi tapaa integroida sovellus Crowdiin. Centralised user management eli keskitetty käyttäjien hallinta on tapa, jossa käyttäjävarasto Crowdista kohdennetaan sovellukselle. Tällä tapaa sovellus voi hakea tietoja käyttäjistä sekä suorittaa todennuksia. Single sign-on tuo sovellukselle

keskitetyn käyttäjienhallinnan lisäksi SSO-käyttäytymistavan. Käytännössä tämä tarkoittaa tapaa, jossa käyttäjän kirjautuessa yhteen SSO-sovellukseen on käyttäjä kirjautunut kaikkiin Crowdiin integroituihin SSO-sovelluksiin. Hakumashup-sovelluksessa käytetään SSO:ta, koska se helpottaa kirjautumista sekä hakutuloksista siirtymistä hakutulokseen.

Keskitetyn käyttäjien hallinnan käyttöönotossa piti sisällyttää Crowdin pavut käynnistävä asetustiedostot sovelluksen kokoonpanoon. Tämä tapahtui kertomalla web.xml-tiedostossa contextConfigLocation-parametrille arvon, joka kertoo tiedoston sijainnin. Tässä sovelluksessa käytetään kahta eri tiedostoa käyttäjien hallinnan asetuksille. Alla olevassa esimerkissä näkyy projektin sisäiset polut tiedostoihin. Tiedosto web.xml on nähtävissä kokonaisuudessaan opinnäytetyön liitteessä 1.

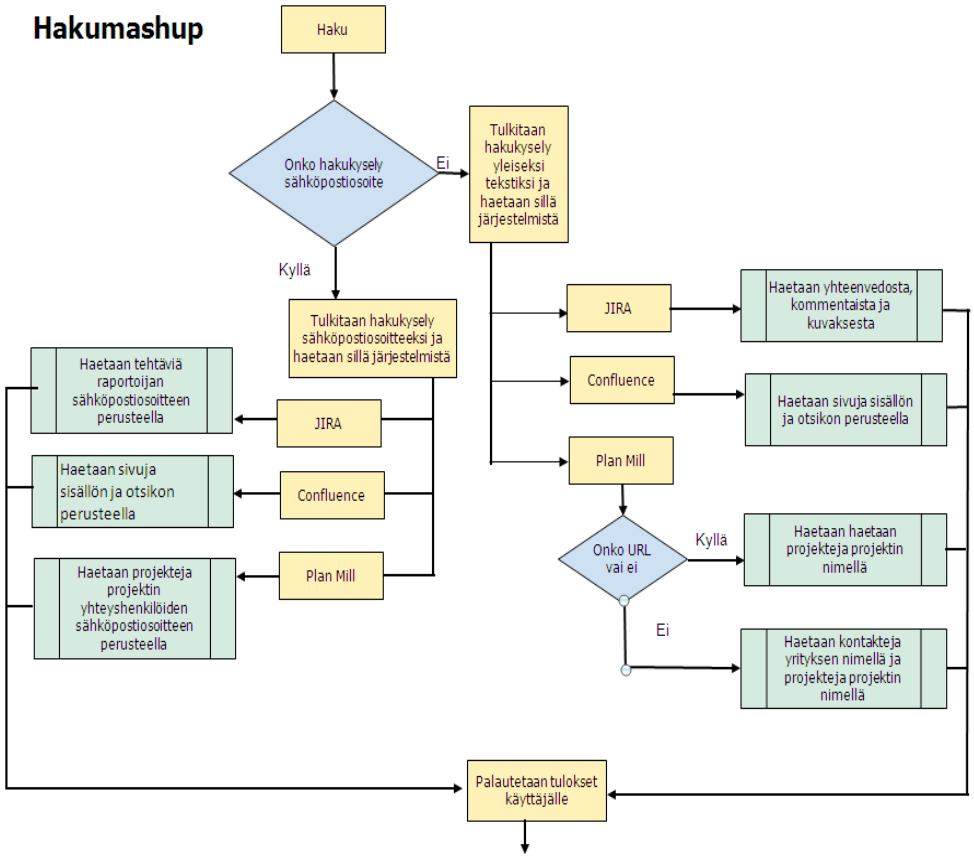
```
<context-param>
            <param-name>contextConfigLocation</param-name>

            <param-value>
            .
            .
            /WEB-INF/spring/crowdClient.xml
            /WEB-INF/spring/security.xml
            .
            .
            </param-value>

</context-param>
```

Tiedostot ovat todella suuri kokoonpano papumäärityksiä, joiden tarkkojen toimintojen selittäminen on opinnäytetyön kannalta tarpeetonta ja veisi liian suuren osan tästä opinnäytetyöstä. Kokoonpano on kasattu yhdistelemällä valmiiden kokoonpanojen esimerkeistä poimittuja määrityksiä. Yläpuolella olevassa esimerkissä määritetty crowdClient.xml-tiedosto on kokonaisuudessaan liitteessä 2 ja security.xml-tiedosto liitteessä 4. Ne vastaavat tyypillisen Spring-sovelluksen ApplicationContext-tiedostoa.

6 HAKUMASHUP-SOVELLUKSEN TOTEUTUS

Hakuprosessi alkaa käyttäjästä katsottuna sivupohjan tasolta. Hakukenttään syötetty hakusana lähtee kontrollerille napin painalluksesta. Kontrolleritasolla analysoidaan, onko hakusana sähköpostiosoite vai yleinen merkkijono, jonka mukaan suoritetaan seuraavat toiminnot. Seuraavaksi hakusana ohjataan palvelukerrokseen SearchService-luokalle. Luokan tarkoitus on kasata JiraService-, ConfluenceService- ja PlanMillService-luokkien toiminnot selkeämmäksi kokonaisuudeksi. SerchService-luokan metodit suorittavat muiden palveluluokkien metodit, joissa luodaan integroitavien järjestelmien ymmärtävät kyselyt ja palauttavat vastauksena saadut hakutulokset SearchService-luokan kautta takaisin kontrollerille. Kontrollerissa vastauksista tehdään sivupohjan ymmärtävät listat, jotka viimein käyttäjälle näytetään Web-sivulla. Tarkemmat kuvaukset eri oh-

jelmakerrosten toiminnoista kuvataan tulevissa luvuissa. Alla olevassa kuvan 4 vuokaaviossa on esitetty haun eteneminen hakusanan mukaan vaihtuvissa tilanteissa.



Kuva 4. Vuokaavio haun edistymisestä

6.1 JIRA ja Confluence REST-haut

Hakumashup sovelluksessa haut JIRAan ja Confluenceen tehdään järjestelmien REST-rajapinnan kautta. Kyseisten järjestelmien hakutoiminnot ovat samankaltaisia. JIRAn rajapinta sekä varsinainen hakutoiminto ovat paljon Confluencen vastaavia kehittyneempiä. JIRAn REST-hauissa käytetään JQL-kyselyitä, joita voidaan lähettää GET- ja POST-metodeilla. POST-metodia käytettäessä voidaan kysely lähettää myös JSON-muodossa.

Hakumashupissa REST-haku lähetetään käyttäen Spring Rest Template -työkalua. Kyselyt luodaan palvelukerroksen luokissa kokoamalla HTTP-pyyntö otsakkeista ja sisällöstä. Rest Template lähettää kyselyn järjestelmälle ja antaa vastauksen paluuarvona. Pyynnön lähettäminen Rest Template -työkalun avulla on selitetty tarkemmin alaluvussa 6.2.3. Takaisin tuleva viesti onnistuessaan on molemmissa tapauksissa JSON-muotoinen kuvaus hakutuloksista. Hakutulos deserialisoidaan POJO-olioksi, josta tarkemmin alaluvussa 6.4.

6.1.1 Otsaketietojen luominen

Hakumashup sovelluksessa järjestelmään kirjautuminen ei yksin riitä, vaan jokaisen JIRAan ja Confluenceen lähetettävän HTTP-pyynnön on sisällettävä tunnustautumistiedot. Tunnustautumiseen käytetään crowd.token_key -evästettä, koska tekstimuotoisten tunnusten ja salasanojen kuljettaminen olisi tietoturvariski. Eväste asetetaan HTTP-pyynnön otsakkeisiin. Evästeen perusteella pyynnön vastaanottava järjestelmä vahvistaa Crowdilta käyttäjän pääsyn järjestelmään.

Tarvittava eväste saadaan sovellukselle Spring Security Coren SecurityContext-rajapinnasta, joka pitää sisällään tietoa kirjautuneesta käyttäjästä. Hakumashup sovelluksessa kirjautumistietojen hakemiseen liittyvät metodit ovat eristetty Hakumashup-sovelluksen AuthenticationUtils-luokkaan. Tämän luokan seuraava metodi hakee nykyisen käyttäjän evästeen merkkijonoon:

```
public String getCurrentCredintials() {

 try  {
        return SecurityContextHolder.getContext().
        getAuthentication().getCredentials().toString(); }

 catch (Exception e) {

        return "";
 }
}
```

Evästetiedot haetaan hakuprosessin SearchService-vaiheessa ja asetetaan JIRAn ja Confluencen Connector-luokkien instansseihin muuttujiin. Kun hakupyyntö viimein saapuu Connectorille, tehdään Spring sovelluskehykseen kuuluva HttpHeaders-objekti, jolle eväste asetetaan. Alla esimerkki otsakkeet luovasta metodista:

```
public HttpHeaders createJSONHeaders() {

 return new HttpHeaders() {
  {
    set("Cookie", "crowd.token_key=" + securityToken);
    set("Content-Type", "application/json; charset=utf-8");
  }
 };
}
```

6.1.2 JSON-pyynnön rakentaminen JIRA-tehtävien hakemiseksi

JIRAan lähetettävät pyynnöt ovat JSON-tyyppisiä POST-pyyntöjä. JiraService-luokassa on muutama metodi, jotka tekevät hieman erilaiset ky-

selyt hakusanasta riippuen. Hakusanan ollessa sähköpostiosoite haetaan JIRA-tehtäviä raportoijan perusteella. Tämän hakukyselyn tekevä metodi on yksinkertaisin esimerkki JIRA-kyselyitä tekevistä metodeista:

```
public List<Issue> searchByReporter(String reporter) throws
JSONException, JsonParseException, JsonMappingException,
IOException,RestClientException, URISyntaxException {

      String restCall =
      "{\"jql\":\"reporter='"+reporter+"'\"}";

      String resultBody = jiraConnector.
      sendRequestToJira(restCall).getBody();

      JsonNode issues = getIssueNodesFromJson(resultBody);
      List<Issue> jiraIssues =
      convertIssuesNodeListToIssueList (issues);

      return jiraIssues;
}
```

Lyhyesti sanottuna metodi muodostaa JQL-kyselystä JSON-olion ja lähettää sen JiraConnector-luokan kautta JIRAan. Paluuviestille tehtävät toimenpiteet käsitellään alaluvussa 6.4.

Pelkkä JQL-kysely ei yksin riitä, vaan pyyntöä lähetettäessä on luonnollisesti tiedettävä JIRAn URL-osoite. Osoite haetaan app.properties-tiedostosta, jossa on kaikkien integroitavien järjestelmien osoitteet ja muita tarvittavia tietoja. Tälle tiedostolle on tehty papumääritykset root-context.xml-tiedostossa, joka on kokonaisuudessaan liitteessä 3. Olemassa olevan pavun avulla voidaan asetustiedosto ottaa käyttöön Connector-luokassa ja sieltä voidaan poimia haluttu osoite.

6.1.3 POST-pyynnön lähettäminen JIRAan

REST-palveluita käytetään Java-sovelluksissa yleensä jonkin apuluokan kautta. Spring Rest Template on apuluokka, joka tarjoaa korkean tason metodit HTTP:n perusmetodien käyttöön. Korkealla tasolla tarkoitetaan sitä, että REST-kutsu voidaan toteuttaa yhdellä koodirivillä. Spring Rest Template sisältää toiminnot HTTP:n perusmetodien käyttöön. (Johnson ym. 2012, 603)

POST-pyynnön lähettämiseen JIRAan käytetään Spring Rest Template -luokan exchange-metodia. Sille annetaan osoitteen lisäksi käytettävä HTTP-metodi ja halutun vastauksen muoto, joka on tässä tapauksessa merkkijono. JQL-kysely asetetaan HTTP-pyynnön runkoon HttpEntity-olioon. Sille asetetaan myös otsakemääritykset sisältävä HttpHeaders-olio, johon tunnustautumistiedot asetettiin alaluvussa 6.2.1. Alla esimerkki kyselyn JIRAan lähettävästä metodista:

```
public ResponseEntity<String> sendRequestToJira(String
restJQL) throws JsonParseException, JsonMappingException,
IOException {

   RestTemplate rest = new RestTemplate();
   HttpEntity<String> httpEntity = new HttpEntity<String>
   (restJQL, createJSONHeaders());

   ResponseEntity<String> result = rest
   .exchange(restApiAddress + "search", HttpMethod.POST,
   httpEntity,String.class);

    return result;
}
```

6.1.4 GET-pyynnön lähettäminen Confluenceen

Confluencen REST-rajapinta on vielä kehitysvaiheessa ja se ottaa vastaan ainoastaan GET-pyyntöjä. Pyynnön lähettämisen periaate on kuitenkin sama, kuin alaluvussa 6.2.3. Haun parametrit tulevat URL-osoitteeseen. Varsinainen osoite tulee JIRAn tapaan app.properties-tiedostosta Connector-luokassa. Osoitteen parametrit muodostetaan ConfluenceService-luokan metodeissa.

Hakumashup sovelluksen käyttäjä hakee monissa tapauksissa URL- ja www-osoitteita Confluencesta. Hakutoimintoa testatessa näissä tapauksissa se suoriutui hauista kehnosti. Hakusanan oli oltava lähes täsmälleen palvelussa olevaa osoitetta vastaava. Hakutoiminto ei esimerkiksi löytänyt osoitetta ilman www-etuliitettä, kun se annettiin hakusanaan. Työn tilaajan toiveesta Confluence hakua tehostettiin tarkastamalla hakusanaa. Mikäli hakusana sisältää etuliitteen, käytetään hakutoiminnon OR-määrettä hakemaan ilman etuliitettä ja sen kanssa. Haku etsii Confluence sivuja, joiden otsikko tai sisältöosuus sisältää hakusanassa esiintyvän osoitteen, ei itse Confluence-sivun osoitetta.

6.2 PlanMillin projektien ja kontaktien REST-haut

PlanMillin REST-rajapinnasta haku eroaa muista järjestelmistä siten, että sieltä tulee ulos eri tietorakenteen omaavia tuloksia. Hakumashup sovelluksella halutaan hakea järjestelmästä kontakteja ja projekteja. Projekteja haetaan niiden nimellä sekä projektiin liittyvän käyttäjän sähköpostin perusteella. Kontakteja haetaan tilin nimellä, joka yleensä on jonkin yrityksen nimi. Tarkemmat tapauskohtaiset tiedot on esitelty aikaisemmin kuvassa 4.

Haku PlanMillista oli ongelmallisin integroitavista järjestelmistä. Sen REST-rajapinnasta puuttui yleinen hakutoiminto, joka hakisi käyttäjiä, kontakteja ja projekteja merkkijonon perusteella. Otin yhteyttä PlanMillin henkilökuntaan ja tiedustelin, puuttuuko hakutoiminto tosiaan rajapinnas-

ta. PlanMillilta vastattiin, että kaikki rajapinnan GET-metodit ovat hakuja. Rajapinnan GET-metodeilla ei tuloksia pystynyt suodattamaan, kuin id-numeron perusteella.

Opinnäytetyöni vaatimuksiin ei kuulunut omien hakualgoritmien kirjoittaminen. PlanMillin jättäminen integraation ulkopuolelle tuntui liian suurelta menetykseltä ja olin edennyt luokkien kirjoittamisessa niin pitkälle, että otin vapauden tehdä muutamia hakuja kyseiseen järjestelmään.

6.2.1 Tunnustautuminen PlanMill-rajapintaan

PlanMillin tunnustautumiskäytäntö eroaa täysin JIRAn ja Confluencen vastaavista. PlanMillin REST-rajapintaan ei tunnustauduta jokaisen Hakumashup-sovellukseen kirjautuvan henkilön omilla tiedoilla, vaan käytössä on yksi tunnustieto yritystä kohtaan. Ennen haun tekemistä kuitenkin varmistetaan SearchService-luokassa, että käyttäjä kuuluu ryhmään, jolla on oikeudet hakea PlanMillista.

6.2.2 PlanMillin hakutapaukset ja algoritmit

Kaikissa toteutetuissa hakutapauksissa haetaan PlanMillilta kaikki kontaktit tai projektit listaan. Listat käydään läpi ja niistä poimitaan hakusanaa vastaavat tulokset.

Projektien hakeminen projektin nimellä on yksinkertainen. Siinä verrataan hakusanaa kenttään, joka sisältää projektin nimen. Samoin haettaessa kontakteja tilin nimen perusteella, poimitaan listalta hakusanan mukaisen tilin nimen omaavat kontaktit. Nämä kaksi hakua suoritetaan samassa haussa.

Projektien hakeminen käyttäjän sähköpostiosoitteen perusteella vaatii kaksi vaihetta. Ensin haetaan käyttäjälista ja poimitaan sieltä sähköpostiosoitetta vastaava käyttäjä. Jokaiselle projektille on merkitty kolme yhteyshenkilöä. Haun toinen vaihe on hakea lista projekteista ja verrata löytyneen yhteyshenkilön id-numeroa listalla olevien projektien kontaktihenkilöihin.

Hakusanan ja haettavan tietokentän täsmällisyysvaatimusta pehmentämään on käytetty Levenšhteinin etäisyysalgoritmia käyttävää metodia, jolla lasketaan merkkijonojen eroavaisuusprosenttia. Algoritmi ei ole kovin kehittynyt tähän käyttötarkoitukseen, mutta se helpottaa varsinkin pitkien merkkijonojen vertailuissa esiintyvien, pienien virheiden huomiotta jättämistä. Hauissa vertailtavat merkkijonot lisäksi muutetaan pieniksi kirjaimiksi.

6.3 Hakutulosten deserialisointi

Kaikki Hakumashup-sovelluksen hakutulokset tulevat JSON-muodossa. Tulokset ovat taulukkoja, jotka sisältävät olioita. JIRAn kohdalla oliot ovat tehtäviä ja Confluencen tulokset ovat sivuja. PlanMillista haetaan kahden tyyppisiä tuloksia, kontakteja ja projekteja.

Jotta tuloksia voidaan käsitellä järkevällä tavalla, ne täytyy deserialisoida eli muuttaa Java-olioiksi. Muuntoon käytetään Jackson nimistä JSON-prosessoijaa. Ensimmäinen vaihe muunnossa on lukea Jacksonin avulla hakutulosmerkkijono JsonNode-olioksi. Tämä olio voi sisältää muutakin tietoa, kuten tässä tapauksessa se sisältää tietoa saatujen tulosten määrästä. Tämän vuoksi sen sisältä pitää hakea varsinainen lista löydetyistä tehtävistä toiseen JsonNode-olioon. Ohjelmakoodissa tämä tapahtuu seuraavasti:

```
public JsonNode getIssueNodesFromJson(String resultBody)
throws IOException, JsonParseException,
JsonMappingException {

    ObjectMapper objectMapper = new ObjectMapper();
    JsonNode nodes = objectMapper.readValue(resultBody,
    JsonNode.class);

    JsonNode issues = nodes.get("issues");

    return issues;

}
```

Paluuarvona saatava JsonNode-olio on lista tehtäviä, mutta sen tietorakenne ei ole riittävän selkeä sen jatkokäsittelyä varten. Sen vuoksi seuraava vaihe on muuttaa se Java-listaksi itse kirjoitetun Issue-luokan olioita. Jackson osaa muuttaa JsonNode-listan kentän Issue-oliksi, mikäli siinä on kaikki tarvittavat muuttujat. Muuttuja täytyy lisäksi merkitä JSON-ominaisuudeksi annotaatiolla:

```
@JsonProperty("Id")
private String id;
```

PlanMillin ja Confluencen kohdalla noudatetaan täsmälleen samaa kaavaa, kuin JIRAn.

6.4 Hakutulosten käsittely Controller-tasolla

Kontrollerin tehtävä hakusanan palvelukerrokselle välittämisen lisäksi on hallita listojen sisällön näyttämistä sivunumeron mukaan. Tuotantokäytössä voidaan odottaa suuriakin määriä hakutuloksia, joten käyttäjän helpottamiseksi täytyy hakutulosten määrää rajoittaa yhdellä sivulla. Jokaisen järjestelmän hakutulosten selaamiseen tarvitaan oma määrältään rajoitettu lista.

Sivutus on toteutettu käyttämällä Spring-sovelluskehyksen PagedListHolder-luokkaa. Se mahdollistaa listojen tarkastelun sivupohjassa siten, että listasta näytetään vain osa. Kontrolleri saa SearchService-luokalta HashMap-olion, josta voi poimia hakutuloslistat niiden nimien perusteella. Kontrollerissa otetaan HashMapista olemassa olevat listat hakutuloksista ja asetetaan ne omiin PageListHolder-säiliöihin. Säiliöt vaativat parametrikseen myös sivunumeron. Jos käyttöliittymästä ei saada sivunumeroa säiliölle, on se oletuksena 0, joka tarkoittaa ensimmäistä sivua. Säiliöt luova metodi on eristetty SearchService-luokkaan kontrollerin metodimäärän vähentämiseksi. Metodin sisällä oleva muuttuja kertoo, montako hakutulosta yhdellä sivulla näytetään.

Sivutuksen hoitamisessa kontrollerin kautta huonona puolena on se, että haut joudutaan suorittamaan aina uudelleen seuraavalle sivulle siirtyessä. Sivutuksen hoitamiseen sivupohjassa olisi tarvittu todella paljon JavaScript-koodia, joka olisi tehnyt sivupohjasta sekavan ja vaikeasti päivitettävän. Lisäksi halutut hakutulokset oletettavasti löytyvät useimmissa tapauksissa ensimmäiseltä sivulta. Sisäverkossa toimivalla Hakumashupilla ei myöskään tule olemaan kovin montaa samanaikaista käyttäjää, joten sivutuksesta aiheutuva rasite ei tule olemaan ongelma.

6.5 Hakumashup sovelluksen testaus

Hakumashup-sovelluksen yhtenä vaatimuksena oli sovelluksen yksikkötestaus. Testikattavuudesta ei ollut tarkempaa puhetta, mutta itse olen pitänyt 80 %:n testikattavuutta koko projektin osalta hyvänä tavoitteena. Tärkeintä on testata palvelukerros, koska se sisältää kaikki hakutuloksiin vaikuttavat toiminnot. Sovellus on kehitetty lähes kokonaan testauslähtöisellä menetelmällä. Ainoastaan kontrollerin toimintoja tehtäessä näin ei toimittu kaikissa tilanteissa, koska mahdollisia menetelmiä sivutuksen toteutukseen oli useita. Sivutustoiminnot toimivat kontrollerin ja sivupohjan yhteispelillä ja niitä oli testattava kokeilemalla niiden toimivuutta käytännössä.

Kuvasta 5 nähdään, että testikattavuus jäi hieman alle tavoitellun tason sovellusta kehitettäessä, mutta se olisi kuitenkin nostettavissa kirjoittamalla lisää testejä jälkikäteen. Testikattavuutta pienensivät luokat, jotka käytettiin datan deserialisointiin. Ne sisältävät paljon muuttujia ja näin myös koodirivejä, mutta varsinaista toiminnallisuutta niissä ei ole. Siksi niitä olisi turhaa testata.

Hakumashup (1) (17.4.2013 13:43:18)

Element	Coverage	Covered Instructions	Missed Instru...	Total Instructions
Hakumashup	74,3 %	3 184	1 099	4 283
src/main/java	60,3 %	1 527	1 007	2 534
net.ambientia.hakumashup.object	30,3 %	185	426	611
net.ambientia.hakumashup.connector	0,0 %	0	319	319
net.ambientia.hakumashup.service	87,1 %	1 104	163	1 267
JiraService.java	78,8 %	205	55	260
AuthenticationUtils.java	0,0 %	0	47	47
SearchService.java	83,9 %	245	47	292
PlanMillService.java	97,3 %	505	14	519
ConfluenceService.java	100,0 %	149	0	149
net.ambientia.hakumashup	70,6 %	238	99	337
HomeController.java	70,6 %	238	99	337
src/test/java	94,7 %	1 657	92	1 749
net.ambientia.hakumashup.controller	67,8 %	118	56	174
HomeControllerTests.java	67,8 %	118	56	174
net.ambientia.hakumashup.service	97,6 %	1 485	36	1 521
PlanMillServiceTests.java	91,7 %	253	23	276
JiraServiceTests.java	97,7 %	292	7	299
ConfluenceServiceTests.java	98,2 %	328	6	334
SearchServiceTests.java	100,0 %	612	0	612
net.ambientia.hakumashup.object	100,0 %	54	0	54

Kuva 5. Hakumashup-sovelluksen testikattavuus

Aikaisimmissa kehitysvaiheissa yksikkötestit hakivat testidatan suoraan kehitysympäristöstä. Testausta varten oli aina kirjauduttava johonkin integroitavista järjestelmistä selaimella, ja poimittava kirjautumiseen käytettävä eväste selaimen kehittäjätyökaluilla. Paikallisella tietokoneella oleva kehitysympäristö ei kuitenkaan tule olemaan ikuisesti olemassa, joten jatkokehityksen kannalta testien ajaminen kehitysympäristöä vasten ei voinut jäädä pysyväksi ratkaisuksi.

Pysyvämpi ratkaisu yksikkötestaukseen tehtiin tallentamalla JSON-muotoiset hakutulokset tiedostoihin. Tiedostoon tallennettiin testimateriaalia onnistuneiden hakutulosten lisäksi tyhjistä hakutuloksista, jotta testitapauksia saataisiin mahdollisimman laajasti. Testimateriaali on projektihakemistossa versionhallinnassa, joten testit toimivat millä tahansa koneella.

6.6 Mock-olioiden käyttö testauksessa

Tiedostossa olevan testimateriaalin käyttämistä varten oli tehtävä uusia luokkia. Uudet luokat nimettiin Connectoreiksi, ja niihin eristettiin palvelukerroksen luokista ne metodit, jotka ovat yhteydessä integroitaviin järjestelmiin. Uudet luokat oli tehtävä, koska niistä tarvittiin mock-olioita, joita jatkossa kutsutaan mokeiksi. Tässä tapauksessa Connector-luokkien mokit palauttavat tiedostoihin tallennettuja JSON-muotoisia hakutuloksia. Tällä tapaa mokit huijaavat Service-luokkia luulemaan, että ne saavat hakutuloksia tietolähteestä.

Tässä työssä mokkeja luodaan testaukseen tarkoitetulla Mockito-sovelluskehikolla. Mokin luominen tapahtuu yksinkertaisesti sovelluskehikon mock-metodilla. Kappaleen alapuolella on esimerkki, jossa luodaan mokki ConfluenceConnect-luokasta ja asetetaan se ConfluenceService-oliolle ennen jokaista yksittäistä testiä ajettavassa before-metodissa.

Mock-metodi on tuotu testiluokan käyttöön käyttämällä Javan import static -toimintoa

```
private ConfluenceService confluenceService;
private ConfluenceConnector mockedConnector;

@Before
public void before() throws JsonParseException,
JsonMappingException, IOException {

    confluenceService = new ConfluenceService();
    mockedConnector = mock(ConfluenceConnector.class);
    confluenceService.
    setConfluenceConnector(mockedConnector);

}
```

Mokkien metodien palautusarvojen asettamiseen käytetään Mockito-sovelluskehyksen when-metodia. Alla olevassa esimerkissä tehdään mokki SpringRestTemplate-luokan käyttämästä ResponseEntity-luokasta. Mokki asetetaan palauttamaan tiedostosta merkkijonoksi luettu JSON-vastaus. Lisäksi luodaan mokki ConfluenceConnector-luokasta, ja asetetaan pyynnön lähettävä metodi palauttamaan ResponseEntity-mokki. Tällä tapaa saadaan palautettua onnistuneen haun tuloksia jäljittelevä vastaus ConfluneService-luokalle.

```
public void addWhenMethodsForSuccesfullSearch() throws
JsonParseException,JsonMappingException, IOException {

   String jsonFilePath = this.getClass().getClassLoader()
   .getResource("testConfluenceSearch.json").toString()
   .replace("%20", " ").substring(5);

   confluenceJsonTestFile = new File(jsonFilePath);
   confluenceTestJson = readFile(confluenceJsonTestFile);

   ResponseEntity<String> mockedResponse =
   mock(ResponseEntity.class);

   when(mockedResponse.getBody())
   .thenReturn(confluenceTestJson);

   when(mockedConnector.sendRequestToConfluence
   (Mockito.anyString()))
   .thenReturn((ResponseEntity<String>) mockedResponse);

}
```

6.7 JSP-sivu käyttöliittymänä

Opinnäytetyön aikana toteutettava käyttöliittymä on koristeeton Web-sivu, kuten kuvasta 6 näkyy. Sivun elementit on järjestelty selkeisiin osioihin, jotta jatkokehityksessä graafisen ulkoasun tekeminen olisi helppoa. Hakusana lähettämiseen sivulla on yksi tekstikenttä ja painike. Sivulla on viisi palstaa hakutuloksille: JIRAn tulokset, Sisäisen Confluencen tulokset, Ul-

koisen Confluencen hakutulokset sekä PlanMillin projektien- ja kontaktien tulokset. Hakutulokset asetetaan sivulla HTML-listoihin. Lista näytetään vain, jos kyseisestä palvelusta saadaan hakutuloksia käytetyllä hakusanalla.

Kuva 6. Hakumashup-sovelluksen käyttöliittymä

7 YHTEENVETO

Opinnäytetyön tavoitteena ollut sovellus saatiin toteutettua kaikilla halutuilla ominaisuuksilla. Kehitysympäristössä sovellus toimii moitteetta, mutta tuotantoympäristössä sitä ei ole vielä voitu kunnolla testata. Tuotantokäyttöön siirtyessä sovelluksessa ilmeni ongelmia PlanMillin osalta, koska järjestelmän palauttamat vastaukset olivat muuttuneet hieman. JIRAn ja Confluencen osalta Hakumashup toimii moitteetta. Tulen korjaamaan PlanMillin hakutoiminnon piakkoin, kun saan tarvittavat oikeudet palvelimelle.

Suurin vastaan tullut ongelma opinnäytetyössäni oli se, että PlanMill-järjestelmän ohjelmointirajapinnassa ei ollut hakutoimintoa. Tiedot saatiin kyllä haettua järjestelmästä, mutta niitä ei voinut suodattaa muuten kuin id-numerolla. PlanMill-järjestelmän pois jättäminen olisi ollut iso menetys Hakumashup-sovelluksen kannalta, joten kirjoitin sille muutamia hakualgoritmeja, vaikka se ei kuulunut opinnäytetyöhöni. Muita suuria ongelmia ei opinnäytetyön aikana tullut vastaan. Muut vaikeudet olivat Spring Security -sovelluskehyksen ja Crowdin integraatioon liittyviä konfiguraatioongelmia. Tämä oli kuitenkin iso osa koko toteutuksesta eikä sen odotettu olevan ihan yksinkertaista.

Teoriaosuuden kirjoittamiseen käytin järjestelmien dokumentaation lisäksi paljon Internet-lähteitä niiden lyhyen ja yksinkertaisen olemuksen vuoksi. Kirjallisuutta käytin myös jonkin verran. Hyödyntämäni teokset ovat käytännönläheisiä ja lähinnä edistyneille ohjelmoijille suunnattuja.

Minulta kului opinnäytetyöhön paljon enemmän aikaa kuin alun perin suunnittelin. Käytännön osuus vaikutti suunnitelman pohjalta yksinkertaiselta, mutta se osoittautui työläämmäksi kuin odotin. Toteutettuun sovellukseen kertyi yli 2 300 riviä ohjelmakoodia. Pelkästään koodaamiseen kului yhtä kauan, kuin olin suunnitellut käyttäväni koko opinnäytetyön valmiiksi saattamiseen.

Pidän opinnäytetyötäni onnistuneena projektina. Toteutettava sovellus valmistui ennakkoon oletetun tason mukaisesti. Täysin valmis sovellus ei vielä ole. Siitä puuttuu vielä ulkoasun koristelut ja pieniä hiomisia.

LÄHTEET

Alex B. & Taylor L. 2013. Spring Security. Reference Documentation. Spring Source. Viitattu 5.4.2013. http://static.springsource.org/spring-security/site/docs/3.2.x/reference/introduction.html

Asiakkaan liiketoiminnan digitalisoiminen. 2012. Yritys. Ambientia Oy. Viitattu 6.9.2012. http://www.ambientia.net/portal/fi/yritys/

Bender J. & McWherter J. 2011. Professional Test Driven Development with C# Developing Real World Applications with TDD. Wiley Publicing Inc. 20.3.2013. http://books.google.fi/books?id=pZ49u0fSuGsC&printsec=frontcover&hl=fi#v=onepage&q&f=false

Best J. 2008. HTTP Headers in Web Development. HTTP Headers in Web Development. DevShed. Viitattu 27.9.2012. http://www.devshed.com/c/a/PHP/HTTP-Headers-in-Web-Development/1/

Crisper L. 2009. Methods and Tools: Practical knowledge for the software developer, tester and project manager. Viitattu 2.4.2013. http://www.methodsandtools.com/PDF/mt200902.pdf

Faber S. 2010. Features and Motivations. Mockito. Viitattu 2.4.2013. https://code.google.com/p/mockito/wiki/FeaturesAndMotivations

Fielding R., Irvine UC., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T. 1999. Hypertext Transfer Protocol -- HTTP/1.1. Protocols. W3C. Viitattu 25.9.2012. http://www.w3.org/Protocols/rfc2616/rfc2616.txt

GET and POST Method of HTTP. 2008. Servlets. Rose India Technologies PVT. LTD. Viitattu 26.9.2012. http://www.roseindia.net/servlets/GetAndPostMethodOfHttp.shtml

Gourley D. & Totty B. 2002. HTTP: the Definitive Guide. O'Reilly Media, Inc. Viitattu 16.3.2013. http://books.google.fi/books?id=qEoOl9bcV_cC&printsec=frontcover&hl=fi#v=onepage&q&f=false

Hamill P. 2005. Unit Test Frameworks: Tools for High-Quality Software Development. O'Reilly Media Inc. Viitattu 1.4.2013. http://books.google.fi/books?id=2ksvdhhnWQsC&printsec=frontcover&hl=fi#v=onepage&q&f=false

Hunt A. & Thomas D. 2007 Pragmatic Unit Testin in C# with NUnit, Second Edition. The Pragmatic Programmers LLC.

Introducing JSON. JSON. Viitattu 5.3.2013. http://www.json.org/

Johnson R. 2012. Introduction to the Spring Framework. Viitattu 3.4.2013. http://www.theserverside.com/news/1364527/Introduction-to-the-Spring-Framework

Johnson R., Hoeller J., Donald K., Sampaleanu C.,Harrop R., Risberg T., Arendsen A., Davison D., Kopylenko D., Pollack M., Templier T., Vervaet E, Tung P., Hale B., Colyer A., Lewis J., Leau C., Fisher M., Brannen S., Laddad R., Poutsma A., Beams C., Abedrabbo T., Clement A., Syer D., Gierke O., Stoyanchev R., Webb P. 2012. Spring Framework Reference Documentation. Spring Source. Viitattu 3.4.2013. http://static.springsource.org/autorepo/docs/spring-framework/3.2.1.SPR-10138-httpservletbean-environment-SNAPSHOT/spring-framework-reference/pdf/spring-framework-reference.pdf

Muthuraman M. 2012. Spring MVC Framework Tutorial. DZone. Viitattu 6.4.2013. http://www.dzone.com/tutorials/java/spring/spring-mvc-tutorial-1.html

REST. Tampereen teknillinen yliopisto. Viitattu 27.3.2013. http://www.cs.tut.fi/kurssit/OHJ-5201/materiaali/9.pdf

Sandoval J. 2009. Restful Java Web Services. Packt Publishing Ltd. Viitattu 13.3.2013. http://www.google.fi/books?id=NS6FeLs6hwMC&printsec=frontcover&hl=fi#v=onepage&q&f=false

Salo P. 2007. Mitä on Aspect Oriented Software Programming. Aspekt Oriented Programming. Viitattu 5.4.2013. http://perttiaspect.blogspot.fi/

Sorsa M. 2008. ASPEKTIYMPÄRISTÖT ASPECTJ JA SPRING AOP. Joensuun Yliopisto. Tietojenkäsittelytiede. Pro gradu -tutkielma.

Vihavainen A. & Luukkainen M. 2012. Web-sovellusohjelmointi. Helsingin yliopisto. Viitattu 25.9.2012. http://www.cs.helsinki.fi/group/java/k12-wad/materiaali.html#9

What is HyperText.2012. W3C. Viitattu 25.9.2012. http://www.w3.org/WhatIs.html

Wilde E. & Pautasso C. 2011. REST: From Research to Practic. Springer Science Business Media, LLC.

WWW. 2012. Tech Terms. Viitattu 25.9.2012. http://www.techterms.com/definition/www

Liite 1

Asetustiedosto: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
           http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

<!-- The definition of the Root Spring Container shared by all
Servlets and Filters -->

<context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>
           /WEB-INF/spring/root-context.xml
           /WEB-INF/spring/crowdClient.xml
           /WEB-INF/spring/security.xml
       </param-value>
</context-param>


<!-- Creates the Spring Container shared by all Servlets and Filters -
->
<listener>
      <listener-class>
           org.springframework.web.context.ContextLoaderListener
      </listener-class>
</listener>

<!-- Processes application requests -->
<servlet>
     <servlet-name>appServlet</servlet-name>
     <servlet-class>
           org.springframework.web.servlet.DispatcherServlet
     </servlet-class>

           <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>
                      /WEB-INF/spring/appServlet/servlet-context.xml
                </param-value>
           </init-param>
           <load-on-startup>1</load-on-startup>
           </servlet>

           <servlet-mapping>
                <servlet-name>appServlet</servlet-name>
                <url-pattern>/</url-pattern>
           </servlet-mapping>

                     <!-- Spring Security -->
           <filter>
             <filter-name>
                  springSecurityFilterChain
             </filter-name>
             <filter-class>
                  org.springframework.web.filter.DelegatingFilterProxy
               </filter-class>
           </filter>
```

```
            <filter-mapping>
                 <filter-name>
                       springSecurityFilterChain
                  </filter-name>
                  <url-pattern>/*</url-pattern>
            </filter-mapping>

</web-app>
```

Asetustiedosto: crowdClient.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-
3.1.xsd">

<beans:bean id="resourceLocator"
class="com.atlassian.crowd.service.client.ClientResourceLocator">
        <beans:constructor-arg value="app.properties"/>
        <beans:constructor-arg value="${app_root}/"/>
</beans:bean>

<beans:bean id="clientProperties"
class="com.atlassian.crowd.service.soap.client.SoapClientPropertiesImp
l" factory-method="newInstanceFromResourceLocator">
<beans:constructor-arg type=
"com.atlassian.crowd.service.client.ResourceLocator"
ref="resourceLocator"/>
</beans:bean>

<beans:bean id="securityServerClient"
class="com.atlassian.crowd.service.soap.client.SecurityServerClientImp
l">
<beans:constructor-arg ref="clientProperties"/>
</beans:bean>

<beans:bean id="crowdAuthenticationManager"
class="com.atlassian.crowd.service.cache.CacheAwareAuthenticationManag
er">
        <beans:constructor-arg index="0" ref="securityServerClient"/>
        <beans:constructor-arg index="1" ref="crowdUserManager"/>
</beans:bean>

<beans:bean id="httpAuthenticator"
class="com.atlassian.crowd.integration.http.HttpAuthenticatorImpl">
        <beans:constructor-arg ref="crowdAuthenticationManager"/>
</beans:bean>

<beans:bean id="verifyTokenFilter"
class="com.atlassian.crowd.integration.http.VerifyTokenFilter" lazy-
init="true">
        <beans:constructor-arg ref="httpAuthenticator"/>
</beans:bean>

<beans:bean id="ehcacheManager"
class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
        <beans:property name="configLocation" value="classpath:/crowd-
           ehcache.xml"/>
</beans:bean>

<beans:bean id="clientCache"
class="com.atlassian.crowd.service.cache.CacheImpl">
        <beans:constructor-arg type="net.sf.ehcache.CacheManager"
           ref="ehcacheManager"/>
```

```
    </beans:bean>

    <beans:bean id="crowdUserManager"
class="com.atlassian.crowd.service.cache.CachingUserManager">
        <beans:constructor-arg index="0" ref="securityServerClient"/>
        <beans:constructor-arg index="1" ref="clientCache"/>
</beans:bean>

<beans:bean id="crowdGroupMembershipManager"
class="com.atlassian.crowd.service.cache.CachingGroupMembershipManager
">
        <beans:constructor-arg index="0" ref="securityServerClient"/>
        <beans:constructor-arg index="1" ref="crowdUserManager"/>
        <beans:constructor-arg index="2" ref="crowdGroupManager"/>
        <beans:constructor-arg index="3" ref="clientCache"/>
</beans:bean>

<beans:bean id="crowdGroupManager"
class="com.atlassian.crowd.service.cache.CachingGroupManager">
        <beans:constructor-arg index="0" ref="securityServerClient"/>
        <beans:constructor-arg index="1" ref="clientCache"/>
</beans:bean>



<beans:bean id="crowdLogoutHandler"
class="com.atlassian.crowd.integration.springsecurity.CrowdLogoutHandl
er">
    <beans:property name="httpAuthenticator" ref="httpAuthenticator"/>
</beans:bean>

<beans:bean id="logoutFilter"
class="org.springframework.security.web.authentication.logout.LogoutFi
lter">
    <beans:constructor-arg value="/index.jsp"/>
    <beans:constructor-arg>
        <beans:list>
            <beans:ref bean="crowdLogoutHandler"/>
            <beans:bean
class="org.springframework.security.web.authentication.logout.Security
ContextLogoutHandler"/>
        </beans:list>
    </beans:constructor-arg>
    <beans:property name="filterProcessesUrl" value="/logout.jsp"/>
</beans:bean>

</beans:beans>
```

Liite 3

Asetustiedosto: root-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:util="http://www.springframework.org/schema/util"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.0.xsd">

<util:properties id="appProps" location="classpath:app.properties" />

<beans:bean id="confluenceService"
class="net.ambientia.hakumashup.service.ConfluenceService" />

<beans:bean id="authenticationUtils"
class="net.ambientia.hakumashup.service.AuthenticationUtils" />

<beans:bean id="planMillService"
class="net.ambientia.hakumashup.service.PlanMillService" />

<beans:bean id="searchService"
class="net.ambientia.hakumashup.service.SearchService" />

<beans:bean id="jiraService"
class="net.ambientia.hakumashup.service.JiraService" />

<beans:bean id="jiraConnector"
class="net.ambientia.hakumashup.connector.JiraConnector" />

<beans:bean id="confluenceConnector"
class="net.ambientia.hakumashup.connector.ConfluenceConnector" />

<beans:bean id="planMillConnector"
class="net.ambientia.hakumashup.connector.PlanMillConnector" />

<beans:bean
class="org.springframework.web.servlet.mvc.annotation.AnnotationMethod
HandlerAdapter">
    <beans:property name="messageConverters">
        <beans:list>
            <beans:bean
class="org.springframework.http.converter.json.MappingJacksonHttpMessa
geConverter" />
        </beans:list>
    </beans:property>
</beans:bean>


</beans:beans>
```

Liite 4

Asetustiedosto: security.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans
xmlns:security="http://www.springframework.org/schema/security"
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-
3.1.xsd">



<security:http auto-config="false" create-session="stateless"
entry-point-ref= "crowdAuthenticationProcessingFilterEntryPoint">

<security:custom-filter position="FORM_LOGIN_FILTER"
ref='authenticationProcessingFilter' />

<security:custom-filter position="LOGOUT_FILTER" ref='logoutFilter' />


<security:intercept-url pattern="/login" access="ROLE_ANONYMOUS"/>

<security:intercept-url pattern="/**" access="ROLE_crowd-
administrators,ROLE_users,ROLE_jira-administrators"/>

<security:intercept-url pattern="/console/secure/**" ac-
cess="ROLE_crowd-administrators"/>
    <security:intercept-url pattern="/console/info/**" ac-
cess="ROLE_crowd-users,ROLE_users"/>
    <security:intercept-url pattern="/console/user/**" ac-
cess="IS_AUTHENTICATED_FULLY"/>
</security:http>

<security:authentication-manager alias="authenticationManager">
          <security:authentication-provider
                    ref='crowdAuthenticationProvider' />
          </security:authentication-manager>

<beans:bean id="crowdAuthenticationProcessingFilterEntryPoint"
class="org.springframework.security.web.authentication.LoginUrlAuthent
icationEntryPoint">

<beans:property name="loginFormUrl" value="/login" />
          </beans:bean>

<beans:bean id="authenticationProcessingFilter"
          class="com.atlassian.crowd.integration.springsecurity.Crowd
          SSOAuthenticationProcessingFilter">

<beans:property name="httpAuthenticator" ref="httpAuthenticator" />

<beans:property name="authenticationManager"
ref="authenticationManager" />

<beans:property name="filterProcessesUrl" value="/j_security_check" />
```

```
<beans:property name="authenticationSuccessHandler">
            <beans:bean
class="org.springframework.security.web.authentication.SavedRequestAwa
reAuthenticationSuccessHandler">

            <beans:property name="defaultTargetUrl" value="/" />
            </beans:bean>
            </beans:property>
</beans:bean>

<beans:bean
class="com.atlassian.crowd.integration.springsecurity.UsernameStoringA
uthenticationFailureHandler">

<beans:property name="defaultFailureUrl" value="/login.jsp?error=true"
/>
</beans:bean>

<beans:bean id="crowdUserDetailsService"
class="com.atlassian.crowd.integration.springsecurity.user.CrowdUserDe
tailsServiceImpl">

<beans:property name="authenticationManager"
ref="crowdAuthenticationManager" />

<beans:property name="groupMembershipManager"
ref="crowdGroupMembershipManager" />

<beans:property name="userManager" ref="crowdUserManager" />

<beans:property name="authorityPrefix" value="ROLE_" />

</beans:bean>

<beans:bean id="crowdAuthenticationProvider"
class="com.atlassian.crowd.integration.springsecurity.RemoteCrowdAuthe
nticationProvider">

<beans:constructor-arg ref="crowdAuthenticationManager" />
                      <beans:constructor-arg ref="httpAuthenticator" />
                      <beans:constructor-arg
                      ref="crowdUserDetailsService" />
</beans:bean>


</beans:beans>
```

Liite 5

Asetustiedosto: app.properties

```
application.name=     hakumashup
application.password= admin
application.login.url=           http://localhost:8095/crowd/console/
crowd.server.url=     http://localhost:8095/crowd/services/
crowd.base.url=       http://localhost:8095/crowd/
session.isauthenticated=         session.isauthenticated
session.tokenkey=     session.tokenkey
session.validationinterval =  2
session.lastvalidation =      session.lastvalidation


planmill.rest.api=https://online.planmill.com/openapi/services/rest
planmill.user.id=****
planmill.auth.key=****
planmill.address=https://online.planmill.com/openapi/
planmill.crowd.group=ROLE_users


jira.rest.api=http://localhost:8080/rest/api/2/
jira.address=http://localhost:8080/

****.confluence.rest.api=http://localhost:8081/confluence/rest/prototy
pe/latest/
super.confluence.rest.api=
http://localhost:8081/confluence/rest/prototype/latest/
```