Opinnäytetyö (AMK)

Tietotekniikan koulutusohjelma

Ohjelmistotekniikka

2010

Aleksi Lahtinen

WWW-SOVELLUKSEN TOTEUTTAMINEN GOOGLE APP ENGINELLE

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma | Ohjelmistotekniikka

Joulukuu 2010 | 35

Ohjaaja: TkL Jari-Pekka Paalassalo

Aleksi Lahtinen

WWW-SOVELLUKSEN TOTEUTTAMINEN GOOGLE APP ENGINELLE

Tämän opinnäytetyön tarkoituksena oli toteuttaa esimerkkitoteutus siitä, kuinka Googlen App Enginelle voidaan luoda Spring Frameworkin avulla WWW-sovelluksia. Google App Engine on uusi pilvipalveluympäristö, joka tarjoaa automaattisen skaalautuvuuden Googlen pilvipalvelussa, jota taas ei perinteisissä WWW-sovelluksissa voida toteuttaa yhtä helposti. Opinnäytetyö tehtiin yhteistyössä Sofokus Oy:n kanssa.

Toteutettava WWW-sovellus oli yksinkertainen tehtävälista, jolla voi seurata ohjelmistoprojektin edistymistä. Tehtävälistan ohjelmakoodi julkistettiin avoimen lähdekoodin GPLv3-lisenssillä, ja se on ladattavista Internetistä vapaasti.

Opinnäytetyön arkkitehtuuriosiossa keskityttiin käytettyihin suunnittelumalleihin ja pilvipalveluiden arkkitehtuuriin. Tekniikkaosiossa tarkasteltiin käytettyjä teknologioita ja Google App Enginen tekniikkaa. Toteutusosiossa käytiin läpi sitä kuinka arkkitehtuuri ja teknologiaa sovellettiin käytännössä toteutettuun WWW-sovellukseen.

Tämän opinnäytetyön tuloksena syntyi tehtävälista WWW-sovellus, jolla voi seurata ohjelmistoprojektien edistymistä.

ASIASANAT:

Spring Framework, Google App Engine, J2EE

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Information Technology | Software Engineering

December 2010 | 35

Instructor: Jari-Pekka Paalassalo, Lic.Tech., Principal Lecturer

Aleksi Lahtinen

IMPLEMENTING A WEB APPLICATION FOR GOOGLE APP ENGINE

The purpose of this thesis was to create an example application on how to create a web application for Google App Engine using Spring Framework. Google App Engine was a good choice for a subject of the thesis because it is a new cloud service offering automatic scalability in Google's cloud service in a much easier way than implementing it would be in a normal development. The thesis was made in collaboration with Sofokus Oy.

The implemented application was a simple to-do list application, which can be used to follow the progress of projects. The to-do list application was published with Open Source GPLv3 license and it is available for free download on the Internet.

The purpose of the application was to be an example on how to make web applications for Google App Engine. The implementation was meant to be as simple as possible and practical to be taken as an example.

As a result of this development a simple web application was created, which can be used by people already familiar with Java web application development, to help to get started with Google App Engine.

KEYWORDS:

Spring Framework, Google App Engine, J2EE

SISÄLTÖ

KUVAT

SYMBOLI- JA LYHENNELUETTELO

DAO	Data Access Object, tietorajapinta
GAE	Google App Engine, Googlen pilvipalveli
GPLv3	GNU General Public License, GNU-hankkeen yleinen lisenssi
GQL	Google Query Language, Googlen kyselykieli
J2EE	Java Platform, Enterprise Edition tai Java EE (aiemmin käytetty nimitystä Java 2 Platform, Enterprise Edition ja J2EE, versioon 1.4 saakka)
JAR	Java ARchive, Java paketti
JSP	Java Server Pages, teknologia HTML- ja XML-muotoisten sivujen luomista varten
MVC	Model-View-Controller, malli–näkymä–ohjain
PaaS	Platfrom as service, sovellusalusta palveluna
SSL	Secure Sockets Layer, salausprotokolla http-yhteyksien suojaamista varten
STS	SpringSource Tool Suite™, Eclipsen opitmoitu versio Spring Frameworkia varten

1 Johdanto

Työn tarkoituksena oli toteuttaa esimerkkisovellus Googlen App Enginelle [1] Spring Frameworkin [2] avulla. Google App Engine on ensimmäinen pilvipalvelu, joka tarjoaa pilvipalveluita ilmaiseksi pientä käyttöä varten, joka mahdollistaa esimerkiksi starttiyrittäjille edullisen aloituksen.

Googlen App Engine on Googlen tarjoama pilvipalvelu, jonne tehdyt ohjelmistot skaalautuvat käyttötarpeen mukaan asetetun kuukausibudjetin rajoissa. Jos taas ohjelmisto toteutetaan tavallisena palvelin-ohjelmistona, tarkoittaisi ohjelmiston skaalaaminen lisätyötä, kun taas pilvipalveluissa ohjelmistot skaalatutuvat automaattisesti pilven käytössä olevien resurssien rajoissa.

2 Arkkitehtuuri

Sovelluksen arkkitehtuuri on malli järjestelmän sisäisestä liiketoimintalogiikasta korkealla abstraktilla tasolla. Sovelluksen arkkitehtuurinen malli sisältää järjestelmän rakenneosat, ulkoiset ominaisuudet ja näiden väliset riippuvuudet.

2.1 Olio-ohjelmointi

Olio-ohjelmointi on ohjelmointitapa, jossa ohjelman perusyksikkönä toimii luokka (Kuva 1). [3] Luokassa voi olla attribuutteja ja metodeja jne. Oliot ovat luokkien ilmentymiä, mikä tarkoittaa käytännössä sitä, että kaikki oliot kirjoitetaan koodissa ensiksi luokkina, joista voidaan sitten luoda ilmentymiä toisissa luokissa. [4] Luokalla voi olla rakentaja ja tuhoaja. Rakentajan koodi ajatetaan silloin, kun luokasta luodaan olioinstanssi, ja tuhoajan koodi ajetaan taas silloin, kun olioinstanssi poistetaan muistista. Java-ohjelmointikielessä ei kuitenkaan ole käytössä tuhoajia, sillä Java-virtuaalikone huolehtii automaattisesti käyttämättömien olioiden poistamisesta muistista.

```java
package org.web.todo.core.entity;

import javax.persistence.Entity;

@Entity
public class TaskEntity {
    private Long id;
    private String title;
    private String type;
    private String description;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }

}
```

Kuva 1 Esimerkki Java-luokasta.

2.2 Palveluorientoitunut kolmikerrosarkkitehtuuri

Kolmikerrosarkkitehtuuri perustuu ohjelmiston jakamiseen kolmeen eri kerrokseen, näyttökerrokseen, loogiseen kerrokseen ja tietokerrokseen. [5] Kolmikerrosarkkitehtuurissa näyttökerros kommunikoi vain ja ainoastaan loogisen kerroksen kanssa, ja looginen kerros kommunikoi näyttökerroksen ja tietokerroksen kanssa. Jakamalla ohjelmisto kerroksiin, voidaan ohjelmistossa olevia sisäisiä riippuvuuksia vähentää, siten että kapseloidaan toiminnallisuus itsenäisiksi kokonaisuuksiksi. Kapseloinnin avulla kerrokset kommunikoivat toistensa kanssa vain

rajapintojen kautta, jolloin eri kerroksia voidaan kehittää toisistaan riippumattomasti ja ne voidaan myös tarvittaessa sijoittaa fyysisesti eri palvelimille. Kerroksien kehittäminen voi olla eri tiimien tai jopa eri yrityksien vastuulla, sillä ne ovat riippuvaisia vain toistensa rajapinta määrittelyistä.

Palveluorientoituneessa arkkitehtuurissa tietojärjestelmän liiketoimintalogiikka eli toiminnot ja prosessit pyritään suunnitelmaan itsenäisinä palveluina, jolloin sovelluksen sisäisiä riippuvuuksia saadaan vähennettyä. Itsenäiset palvelut ilmenevät sitä käyttäville tahoille rajapintana, jolloin itse palveluun tehtävät muutokset eivät vaikuta sitä käyttävien ohjelmien toimintaan millään tavalla niin kauan kuin itse rajapintaan ei tehdä muutoksia.

Esimerkki palveluorientoituneesta palvelusta voisi olla WWW-sovelluksen käyttäjien hallinta. Käyttäjien hallinnassa tarvittavia toimintoja ovat uuden käyttäjän lisääminen palveluun, käyttäjän tietojen muuttaminen ja käyttäjän poistaminen. Käyttäjien hallintapalvelua varten luodaan rajapinta (Kuva 2), johon lisätään palvelussa tarvittavat metodit.

```
1  package org.web.todo.core.service;
2
3  import javax.servlet.http.HttpServletRequest;
4
5  import org.web.todo.core.entity.User;
6
7  public interface UserService {
8
9      public boolean createNewUser(User user, String role);
10
11     public boolean deleteUser(Long id);
12
13     public boolean updateUser(HttpServletRequest request, String username) throws Exception;
14
15 }
```

Kuva 2 Käyttäjäpalvelun rajapinta.

Rajapinnan toteuttavassa luokassa (Kuva 3) rajapinnassa määritellyille metodeille toteutetaan toiminnallisuus. Käyttäjän poistamismetodiin lisätään @Secured-annotaatiota, jolla varmistetaan, että metodia voi ajaa vain pääkäyttäjän oikeudet omaava käyttäjä.

```
1  package org.web.todo.core.service;
2
3  import java.lang.reflect.InvocationTargetException;
23
24 @Service(value="userService")
25 public class UserServiceImpl implements UserService {
26
27     @Autowired private UserDao userDao;
28     @Autowired private PasswordEncoder passwordEncoder;
29     @Autowired private MessageSource messageSource;
30     @Autowired private SecurityService securityService;
31     @Autowired private FormService formService;
32     @Autowired private ProjectDao projectDao;
33     @Autowired private ProjectService projectService;
34
35     /**
36      * Create a new user
37      * @param username
38      * @param password
39      * @param role
40      */
41     public boolean createNewUser(User user, String role)
42     {
43         user.setRole(role);
44         user.setAccountNonExpired(true);
45         user.setAccountNonLocked(true);
46         user.setCredentialsNonExpired(true);
47         user.setEmailVerified(true);
48         user.setEnabled(true);
49         encodePassword(user);
50         userDao.save(user);
51         return true;
52     }
53
54     /**
55      * Delete user
56      * @param id
57      */
58     @Secured(value=Role.ROLE_ADMIN)
59     public boolean deleteUser(Long id)
60     {
61         User user = (User) userDao.getByKey(User.class, CommonUtils.generateKey(User.class, id));
62         // Admin users cannot be deleted.
63         if(user.getRole().equals(Role.ROLE_ADMIN))
64         {
65             return false;
66         }
67         Key userKey = user.getKey();
68         List<Project> projects = projectDao.getProjectsByUser(userKey);
69         for(Project project : projects)
```

Kuva 3 Käyttäjäpalvelun toteuttava luokka.

Palvelun käyttäminen ohjain -luokassa tapahtuu luomalla rajapinta-attribuutti (Kuva 4), jolle annetaan @Autowired Spring Frameworkin annotaatio, jonka avulla Spring Framework pystyy injektoimaan rajapinnalle käyttäjien hallintarajapinnan toteuttavan luokan.

```
1  package org.web.todo.core.controller;
2
3  import java.lang.reflect.InvocationTargetException;
21
22 @Controller
23 @RequestMapping(value="/" + Path.REGISTRATION + "/")
24 public class RegistrationController {
25
26     @Autowired private UserServiceImpl userService;
```

Kuva 4 Käyttäjien hallinta palvelun luominen.

Rajapinnan käyttäminen tapahtuu kutsumalla rajapinta-attribuuttia (Kuva 5) ja sen tarjoamia palveluita.

```
/**
 * Save registration
 * @param request
 * @return
 * @throws InstantiationException
 * @throws ClassNotFoundException
 * @throws NoSuchMethodException
 * @throws InvocationTargetException
 * @throws IllegalAccessException
 * @throws NoSuchFieldException
 * @throws IllegalArgumentException
 * @throws SecurityException
 */
@RequestMapping(method=RequestMethod.POST)
public RedirectView saveRegistration(HttpServletRequest request) throws SecurityException, IllegalArgumentException, NoSuch
{
    if(CommonUtils.isOperationCancelled(request))
    {
        return new RedirectView("/" + Path.FRONTPAGE + "/", false);
    }

    User user = new User();
    if(user.validate(user, messageSource, request))
    {
        userService.createNewUser(user, Role.ROLE_USER);
        MessageUtils.setSessionMessage(request, "registration.completed", Message.INFO);
        return new RedirectView("/" + Path.FRONTPAGE + "/", false);
    }
    else
    {
        user.setErrorForm(request);
    }

    return new RedirectView("/" + Path.REGISTRATION + "/", false);
}
```

Kuva 5 Käyttäjien hallinta palvelun käyttäminen ohjaimessa.

2.3 Malli–näkymä–ohjain arkkitehtuuri

Malli–näkymä–ohjain-arkkitehtuuri (Model-View-Controller) on arkkitehtuuri, jonka avulla voidaan erottaa käyttöliittymä sovelluksen liiketoimintalogiikasta. [6] Arkkitehtuuri mahdollistaa käyttöliittymän itsenäisen testaamisen ja toteuttamisen, sillä se erottaa nämä osa-alueet toisistaan rajapintojen avulla.

Malli–näkymä–ohjain-arkkitehtuuri eroaa kolmikerrosarkkitehtuurista kommunikoinnin osalta. Malli–näkymä–ohjain-arkkitehtuurissa näkymä lähettää syötteen ohjaimelle, ohjain päivittää mallin ja näkymä päivittyy suoraan mallista. Kun taas

kolmikerrosarkkitehtuurissa kaikki kommunikaatio tapahtuu loogisen kerroksen välityksellä.

Malli huolehtii sovelluksen tietojen kuvaamisesta ohjelmointikielen tasolla, mikä esimerkiksi Javassa tarkoittaisi tietokantataulua vastaavaa tietoentiteettiluokkaa. Arkkitehtuurin mukaan tiedonsaantikerros on sisällytetty malliin, mikä tarkoittaa esimerkiksi WWW-sovelluksissa sitä, että tietokantayhteyksistä vastaa malli.

Näkymä huolehtii tiedon esittämisestä, joka tarkoittaa, että tarvittaessa ohjelman tiedot voidaan helposti toteuttaa jollakin toisella esitystavalla, ilman että itse tietomallia tai ohjainkoodia täytyisi muuttaa. Käytännössä tämä tarkoittaisi sitä, että samasta ohjelmasta voitaisiin toteuttaa työpöytä- ja WWW-sovellus, muuttamatta sovelluksen malleja. Palveluorientoituneen arkkitehtuurin avulla toteutettuna itse ohjaimiin ei tarvittaisi suuria muutoksia, sillä ohjelmiston liiketoimintalogiikka on toteutettu ohjaimista riippumattomina palveluina.

Ohjain vastaanottaa syötteitä ja muokkaa syötteen mukaisesti mallia ja näkymää. Käytännössä yleensä ohjain hakee tiedon tietovarastosta, täyttää mallin tiedolla ja näkymä esittää mallin mukana tulevan tiedon. Ohjain voi kuitenkin muokata suoraan näkymää, mistä seuraa se, että saman ohjelman toteuttaminen työpöytä- ja WWW-sovelluksena vaatisi muutoksia myös ohjaimeen. Tämän takia ohjaimen toteutuksessa kannattaa tehdä mahdollisimman vähän muutoksia itse näkymään, jotta ohjaimien ja näkymien välille ei syntyisi liikaa riippuvaisuuksia.

2.4 Pilvipalvelut

Pilvipalveluiden ideana on tarjota sovellusalusta palveluna [7], jolloin asiakkaiden ei tarvitse ajaa WWW-sovellusta omilla tai vuokratuilla palvelimilla, vaan asiakas maksaa ainoastaan käytetyistä palveluista käytön mukaan. Tällöin asiakas voi keskittyä omaan ydinosaamiseensa ja antaa ulkoisten yrityksien vastata täysin sovellusalustan toiminnasta. Pilvipalveluiden tarkoituksena on luoda yksi suuri palvelinfarmi, jonne voidaan asentaa sovelluksia, jotka skaalautuvat automaattisesti käyttötarpeen mukaisesti.

Pilvipalvelut on tarkoitettu pääasiallisesti WWW-sovellusten toteuttamista varten, jolloin palveluun voi kirjautua sisään mistä päin vain maailmaa Internet-selaimen avulla. Joitakin pilvipalveluita voi käyttää myös rinnakkaislaskentaa varten, mutta pilvipalvelut eivät ole paras vaihtoehto rinnakkaislaskentaa varten.

Pilvipalveluiden tärkein hyöty liiketoiminnan kannalta on se, että maksaa palvelusta vain sen verran, minkä verran käyttää palvelua. Tällöin yritys pääsee eroon tavallisten ali- tai ylikapasiteettiongelmista, jotka aiheuttavat ylimääräisiä kuluja.

2.4.1 Pilvipalveluiden teknologia

Pilvipalvelut jakautuvat kahteen kategoriaan: [8] palveluihin, joissa asiakas ostaa skaalautuvan palvelunympäristön, ja palveluihin, joissa asiakas ostaa sovelluskehyksen, jonka päälle voidaan kehittää ohjelmia, jotka skaalautuvat toimittajan toteuttamassa pilvessä automaattisesti käyttötarpeen mukaan. Pilvipalveluiden taustalla on aina skaalautuva palvelinympäristö, joka osaa automaattisesti jakaa käytössä olevia resursseja asiakkaan tarpeiden mukaan.

2.4.2 Pilvipalveluiden skaalautuvuus

Skaalautuvuudellla tarkoitetaan palvelun käytössä olevien palvelinresurssien kykyä laajentua lisääntyvän tarpeen mukaan. Pilvipalvelut skaalautuvat prosessoritehojen, levytilan, kaistanleveyden ja tietovaraston osalta [9].

Prosessoriteho ja levytilan skaalautuminen tapahtuu jakamalla ohjelmiston suoritus useiden eri palvelinyksikköjen hoidettavaksi käyttötarpeen mukaan. Kaistanleveys taas on täysin riippuvainen serverifarmille rakennetuista tietoliikenneyhteyksistä ja niiden jakamisesta eri asiakkaiden kesken.

Tietovaraston skaalautuvuus on kaikista haastavin asia pilvipalveluissa, sillä kun tietovarasto on jaettu useille eri palvelimille, syntyy uusia haasteita tietovaraston yhtenäisyyden ja konsistenssisuden osalta.

2.4.3 Pilvipalveluiden tietovarastot

Pilvipalvelun tietovarasto on varasto, jossa tieto on tallennettuna useille eri palvelininstansseille. Varastoa voi käyttää palveluntarjoajan rajapinnan avulla. Tietovarastolla voidaan tarkoittaa joko skaalautuvaa tietokantaa tai tavallisten tiedostojen tallennusta varten tarkoitettua tietovarastoa.

Pilvipalveluiden tietovarastot takaavat tiedon automaattisen tiedon varmennuksen ja paremman turvallisuuden [10], sillä asiakkaan ei itse tarvitse huolehtia tietovaraston tietoturvasta ja tiedonvarmennuksesta. Myös luotettava pilvipalvelun toimittajan

ydinosaamiseen kuuluu pilvipalvelun tietoturvasta ja tiedonvarmennuksesta vastaaminen, jolloin näiden palveluiden taso on luultavammin korkeammalla, kuin jos yritys itse hoitaisi näiden palveluiden toimittamisen muun toimintansa ohessa.

Pilvipalveluiden tietovarastot tarjoavat hyötyjä verrattuna itse toteutettuun varastoon, sillä asiakas maksaa vain käytetystä tilasta, kun taas itse toteutetussa varastossa maksetaan myös käyttämättömästä tilasta.

Tietovarastoissa on useiden eri asiakkaiden tietoa, tällöin erittäin arkaluontoisen tiedon tallentaminen pilvipalvelun tietovarastoon voi aiheuttaa huolenaiheita mahdollisille asiakkaille. Mitä jos eri asiakkaiden tiedot menisivätkin jonkin tietoteknisen virheen takia sekaisin, ja arkaluontoista tietoa päätyisi jonkin toisen asiakkaan näkyville?

2.4.4 Pilvipalveluiden vertailu

Markkinoilla on useita eri pilvipalveluiden tarjoajia, jotka kaikki tarjoavat ainutlaatuisen pilvipalvelun, sillä mitään yleistä pilvipalveluiden standardia ei ole olemassa. Pilvipalvelut voidaan jakaa kahteen eri kategoriaan, palvelut, jotka tarjoavat automaattisesti skaalautuvia palvelininstansseja ja palveluihin, jotka tarjoavat sovellusalustaa palveluna (Platform as a service), jonka rajapinnalle voidaan rakentaa annettujen kehitystyökalujen avulla ohjelmistoja.

Microsoft Azure Services Platform pilvipalvelua [11] tarjotaan sovellusalusta palveluna. Azure-palvelussa on viisi pääpalvelua, joiden avulla Microsoftin pilveen voidaan toteuttaa sovelluksia. Näitä tärkeimmät palvelut ovat AppFabric ja SQL Azure. AppFabric on palvelu, jolla voidaan toteuttaa sovelluksia pilvipalveluun. SQL Azure:n avulla voidaan luoda skaalautuvia SQL-tietovarastoja.

Amazon EC2 tarjoaa palvelininstansseja [12], joita voidaan minuuteissa lisätä asiakkaan pilveen tukemaan asiakkaan tarvitsemaa palvelinkapasiteettia. Amazonin palvelu tarjoaa palvelininstansseja eikä niinkään sovellusalustaa, vaan asiakkaan pitää itse vastata ostettujen instanssien hallinnasta. Instanssien hallinnointia varten Amazon tarjoaa asiakkaillensa työkaluja, jotka helpottavat pilvipalvelun hallintaa.

Tietovarastoksi Amazon tarjoaa Amazon Elastic Block Store -palvelua [13], joka tarjoaa hyvin skaalautuvan tietovaraston.

Google App Engine [14] pilvipalvelua tarjotaan sovellusalusta palveluna. Google App Enginelle tarjotaan Python ja Java SDK:t. Google tarjoaa tietovarastoksi Googlen BigTable tietovaraston, joka on käytössä useissa Googlen palveluissa. Google App Engine on vielä beta-vaiheessa, joten se on vielä kehitysvaiheessa ja palvelussa on omat rajoitteensa.

3 Tekniikka

Opinnäytetyössä käytetyt tekniikat pohjautuvat täysin Java-teknologioiden pohjalle ja Googlen App Enginen SDK:hen.

3.1 Java

Java on Sun Microsystemsin (nykyisin Oracle Corporation) vuonna 1991 luoma ohjelmistokieli, joka kehitettiin "kirjoita kerran, aja kaikkialla"-periaatteen pohjalle [15]. Tarkoituksena oli luoda kieli, jota voisi ajaa missä tahansa käyttöjärjestelmässä. Tämän takia Java tarvitsee virtuaalikoneen, jotta sen koodia voitaisiin ajaa kohdekäyttöjärjestelmässä. Virtuaalikoneet kääntävät Java tavukoodin ajonaikaisesti binäärikoodiksi, joka mahdollistaa koodin optimoinnin ajonaikaisesti, mikä taas ei ole mahdollista binäärikoodiin perustuvissa ohjelmissa.

Java on TIOBE Programming Community indeksin mukaan toiseksi suosituin ohjelmointikieli maailmassa. [16] Javaa käytetään usein WWW-sovelluksien palvelinpuolen toteutuksessa, ohjelmistoissa joissa vaaditaan erittäin suurta luotettavuutta.

Java-ohjelmointikieli on vahvasti tyypitetty oliopohjainen ohjelmointikieli, joka on julkaistu vapaan lähdekoodin GPL-lisenssillä [17]. Javaa voi siis aja missä tahansa käyttöjärjestelmässä, jolle on tehty toteutus virtuaalikoneesta, mutta jos sellaista ei ole, niin virtuaalikoneen voi toteuttaa vapaasti Oraclen määrittelyiden mukaisesti.

Annotaatiot ovat keino lisätä metatietoa Javan metodeihin, luokkiin, rajapintoihin jne., tätä tietoa voidaan lukea ja käsitellä Java Reflection API:n [18] avulla ajonaikaisesti. Metatiedon avulla ohjelmoija voi ajanaikaisesti esimerkiksi etsiä tiettyä metatietoa sisältäviä metodeita ja metodeita voidaan sitten käsitellä jollain erityisellä tavalla. Metatieto tarvitsevat esimerkiksi ohjelmistokehykset, metatiedon avulla Spring Frameworkia käytettäessä ei tarvitse määritellä ohjaimia erikseen XML-

asetustiedostoissa, vaan ohjain luokkaan voidaan lisätä ohjain määrittely annotaatioiden avulla. Ja kun WWW-sovellus käynnistetään tutkia Spring Framework ennalta annetuista paketeista luokkia joille on asetettu ohjain annotaatio (Kuva 6).

```
@Controller
@RequestMapping(value="/" + Path.REGISTRATION + "/")
public class RegistrationController {
```

Kuva 6 Ohjaimen määrittely annotaatioiden avulla Spring Frameworkissa.

3.1.2 JavaBean

JavaBean on tavallinen Java-luokka [19], joka on toteutettu JavaBean -nimeämiskäytäntöä käyttäen. JavaBeanin määrittelyn mukaan JavaBean luokalla pitää olla rakentaja, jolle ei ole annettu mitään attribuutteja. Luokan jäsenmuuttujilla pitää olla get- ja set-metodit, jotka nimetään jäsenmuuttujien mukaan. Esimerkiksi jos muuttujan nimi on nimi, niin silloin sillä pitää olla getNimi() ja setNimi(String nimi) metodit. Luokan pitää olla serialisoitavissa.

3.2 Sovelluskirjastot

Sovelluskirjastot ovat kirjastoja, jotka tarjoavat työkaluja tai sovelluskehyksiä sovelluksen toteutusta varten. Toteutuksessa käytettiin sovelluskirjastoina ainoastaan avoimen lähdekoodin kirjastoja.

3.2.1 Spring Framework 3-sovelluskehys

Spring Framework on suosittu Java-ohjelmistokehys, jonka avulla voidaan toteuttaa MVC- ja kolmikerrosarkkitehtuuriin mukaisia sovelluksia vaivattomasti. Spring Frameworkilla voidaan toteuttaa aivan minkälaisia sovelluksia vain, mutta se on erityisen suosittu J2EE-sovellusten totutuksessa [20].

Spring Frameworkin [21] keskeisin ominaisuus on riippuvuuksien injektointi, jolla tarkoitetaan riippuvuuksien vähentämistä siirtämällä luokan instanssien luominen Spring Framewokin hoidettavaksi. Tällöin luokan tarjoamia palveluita käyttävän asiakasluokan ei tarvitse tuntea riippuvuuksia, jotka tarvitaan luokan luomiseksi, vaan nämä riippuvuudet hoidetaan vain yhdessä keskitetyssä paikassa. Tämä vähentää ohjelmisto monimutkaisuutta ja riippuvuuksia, kun luokkien muodostaminen on

keskistetty yhdelle palveluita tarjoavalle taholle. Riippuvuuksien injektointi mahdollistaa joustavasti sovelluksen yksikkötestaamisen.

3.2.3 Apache Tiles 2-sovelluskirjasto

Apache Tiles 2 [22] on avoimen lähdekoodin kirjasto, jonka avulla voidaan luoda HTML-mallipohjia WWW-sovelluksille. Tilesin avulla voidaan luoda uudelleen käytettäviä HTML-palasia, joita voidaan helposti yhdistellä Tilesin merkintäkirjaston avulla, jolloin samaa HTML-koodia ei tarvitse kirjoittaa enää uudelleen, jolloin ohjelmiston tarvitsema koodimäärä pienenee ja ohjelmiston ylläpito helpottuu.

3.2.4 Tuckey Url Rewrite Filter-sovelluskirjasto

Tuckey Url Rewrite Filter [23] on suodatin, jonka avulla WWW-sovelluksen polkuja voidaan siistiä. Ajatellaan, että jos ohjelman polku on normaalisti ohjelma ja ohjelman verkkotunnuksen nimi olisi www.ohjelma.fi, silloin hakupolku ohjelmaan olisi www.ohjelma.fi/ohjelma/ , joka ei näytä siistiltä eikä ole hakukoneystävällinen. Filtterin avulla tämä polku voidaan muuttaa siten, että ohjelmaan pääsee suoraan verkkotunnuksen kautta.

Suodatinta tarvitaan esimerkiksi Spring Framworkia käytettäessä, jotta sovelluksessa oleviin staattisiin tiedostoihin pääsisi käsiksi ilman Spring Frameworkin ohjaimien apua.

3.2.5 Spring Security 3-sovelluskirjasto

Spring Security 3 [24] on ohjelmistokehys, jonka avulla voidaan toteuttaa turvallinen todennus - ja auktorisointi WWW-sovellukselle sekä muita turvallisuusominaisuuksia. Yksinkertaisimmassa toteutuksessa tarvitaan vain oikeanlaisten asetuksien asettaminen XML-asetustiedostoon ja Spring Securityn JAR-kirjastojen lisääminen projektin luokkapolkuun. Spring Security mahdollistaa omien toteutuksien toteuttamisen kaikille sovelluksen merkittäville osille rajapintojen avulla.

3.3 Google App Engine

Google App Engine on Googlen pilvipalvelu, jonne voidaan toteuttaa WWW-sovelluksia Java- ja Python-ohjelmointikielillä. Palvelua voi käyttää ilmaiseksi tietyin resurssirajoituksin, jos rajoitukset ylittyvät palvelu ei ole enää käytettävissä kunnes taas

tietyn ajan kuluttua resurssit ovat käytössä rajoitetusti. Resursseja voi ostaa lisää asettamalla Google App Engine-tilillensä kuukausittaiset laskutuksen maksimirajat, jolloin ilmaisten resurssien ylimenevältä osuudelta voidaan veloittaa kuukausittain käytön mukaan aina maksimirajaan asti [25]. Google App Engine tukee tällä hetkellä vain Java- ja Python-ohjelmointikieliä, mutta Google on luvannut tulevaisuudessa tuoda tukea myös muillekin ohjelmointikielille.

3.3.1 Google App Enginen Java-tuki

Google tukee vain sen valkoisella listalla olevia Javan peruskirjastoja [26], koska pilvipalvelun turvallisuuden ja vakauden takia kaikkia toimintoja ei voida tukea. Esimerkiksi tiedostojärjestelmään ei voi kirjoittaa lainkaan ja ohjelmat eivät saa luoda uusia prosessisäikeitä, mutta ajastetut tehtävät ovat sallittuja rajoitetusti Googlen toteuttaman ajastinpalvelun avulla. Vaikka kaikki Javan peruskirjastot eivät olekaan tuettuja, tarjoaa Google palveluina suurimman osan ominaisuuksista, jotka on jätetty tukematta Javan peruskirjastoista.

3.3.2 Google App Enginen tietovarasto

Google App Engine käyttää tietovarastona Googlen kehittämää tietovarastoa, jonne voi tehdä kyselyitä GQL:n [27] avulla, joka on hyvin samankaltainen SQL-kielen kanssa.

Google App Enginen tietovarastossa ei ole varsinaista skeemaa [28], vaan skeemat luodaan samalla kun tietovarastoon tallennetaan tietoa. Tietovarasto perustuu Googlen BigTable tietovarastoon [29], joka on käytännössä suuri hajautustaulu [30], johon voi tallentaa avaimen avulla tietoa. BigTable on suunniteltu erityisesti tietovaraston skaalatutumista varten, jonka takia BigTable tietovarasto ei sisällä aivan kaikkia samoja ominaisuuksia mitä löytyy tyypillisistä relaatiotietokannoista. Esimerkiksi relaatiotietokannoista tuttuja Join-lauseita BigTable ei tue lainkaan. Mutta Join-lauseita vastaavia tietokantakyselyitä on kuitenkin mahdollista toteuttaa, mutta ei SQL:stä tutulla tavalla.

3.3.3 JDO ja JPA teknologiat

Google App Engine tukee JDO (Java Data Objects) [31] ja JPA (Java Persistence API) [32] standardeja. JDO on riippumaton tietovaraston toteutuksesta, kun taas JPA on toteutettu relaatiotietokantoja varten [33]. JDO:n avulla voi toteuttaa

tiedonsaantikerroksen siten, että tarvittaessa koko sovelluksen voi siirtää toiseen palvelinympäristöön ilman suurempia muutoksia.

3.3.4 Googlen App Enginen puutteet ja heikkoudet

Google App Engine on vielä kehitysvaiheessa, joten siinä on olemassa tiedossa olevia ikäviä ongelmia [34], jotka saattavat olla esteenä monelle yritykselle Goole App Engineen siirtymisessä.

Palvelussa esiintyy ajoittaista hitautta silloin, kun Java-virtuaalikoneen instanssi on sammutettu palvelun käyttämättömyyden takia, joka on yleensä 1-2 minuuttia, ja instanssi pitää käynnistää uudestaan. Kun instanssi käynnistetään uudestaan täytyy koko WWW-sovellus käynnistää kokonaan uudestaan, josta aiheutuu sovelluksen käynnistämiseen kuluvan ajan mittainen viive, ennen kuin sovellus pystyy vastamaan kyselyyn. Ongelma ilmenee kuitenkin vain sellaisissa palveluissa, joilla ei ole paljon käyttäjiä, jos taas palvelua käyttävät miljoonat eri käyttäjät, silloin instanssit eivät juurikaan joudu sammutetuksi käyttämättömyyden takia.

Full text search -toiminnallisuus puuttuu palvelusta ja vain erittäin yksinkertaiset kyselyt ovat mahdollisia tietovarastoon. Tästä syystä täysimittaista hakukonetta tarvitsevaa ohjelmistoa ei voi toteuttaa ainakaan tällä hetkellä Googlen App Enginessä.

SSL on rajoitettu vain *.appspot.com verkkotunnuksen alle, mikä tarkoittaa, että jos omistat verkkotunnuksen nimeltä www.mydomain.net, niin et voi salata liikennettä SSL:n avulla kyseiseen verkkotunnuksen alle. Tämän takia turvallisten sovellusten toteuttaminen voi olla vaikeaa, jos haluaa käyttää omaa verkkotunnustansa.

Google on koko ajan kehittämässä palvelua, ja edellä mainitut ongelmat ovat Googlen korjattavien listalla [35], joten on erittäin todennäköistä, että Google korjaa puutteet ja palvelusta tulee erittäin otollinen alusta yrityksien pilvipalvelualustaksi.

4 Toteutus

Opinnäytetyö toteutettiin avoimen lähdekoodin SpringSource Tool Suite™-ohjelmistolla [36], joka on SpringSource kehittämä ohjelmistokehitysympäristö, joka pohjautuu Eclipseen. Toteutus tallennettiin Google Code Project Hosting –palveluun, josta sen lähdekoodin voi ladata.

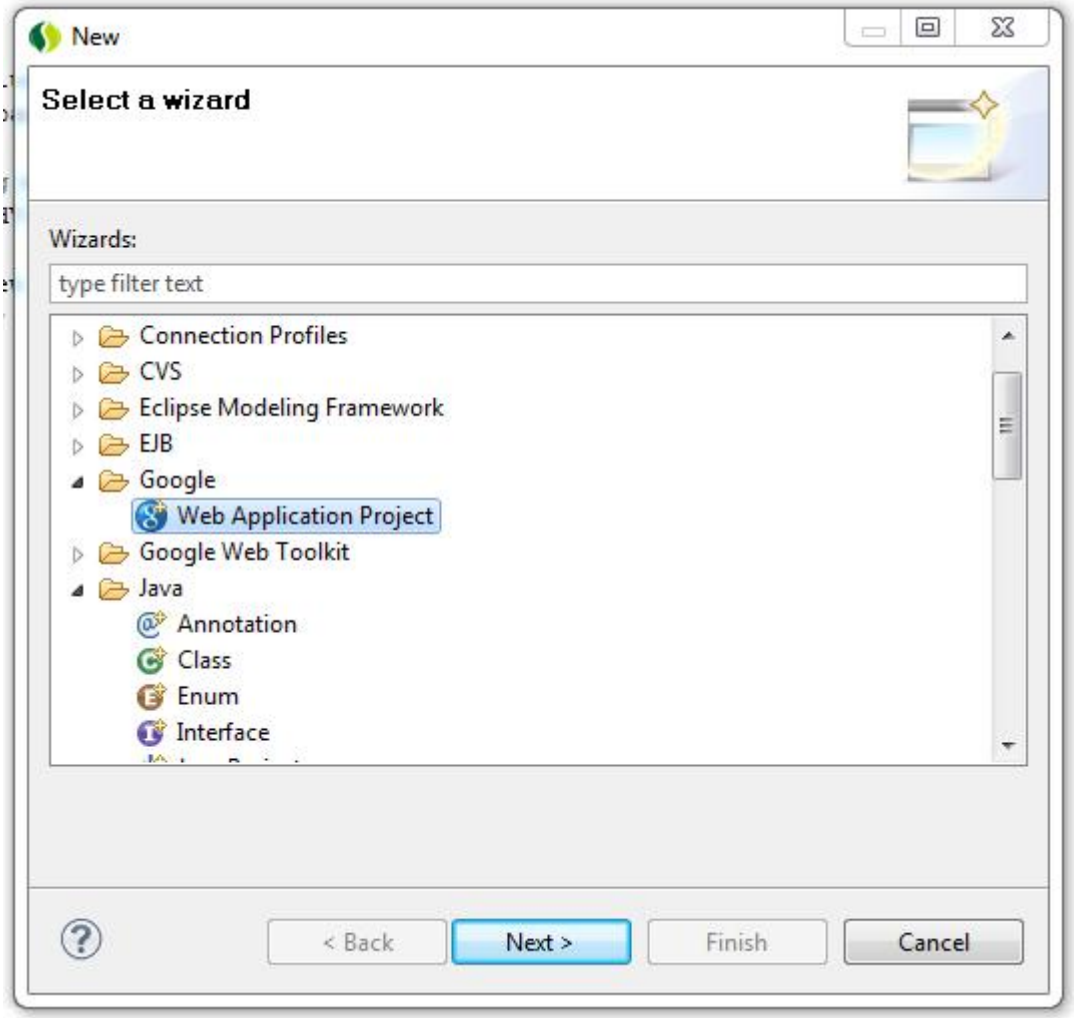
4.1 WebTodo -tehtävälista WWW-sovellus

Tavoitteena oli toteuttaa tehtävälista ohjelmisto, jonka avulla voidaan luoda projekteja ja projekteille voidaan antaa tehtäviä. Tehtävät voidaan merkität valmiiksi ja tehtäviin voidaan lisätä kommentteja. Ohjelmiston tarkoituksena on auttaa projektille asetettujen tehtävien etenemistä ja antaa mahdollisuus kommentoida yksittäisiä tehtäviä. Käyttäjäksi voi rekisteröityä sovelluksen etusivulta ja sen jälkeen käyttäjä voi kirjautua sisään sovellukseen.

Ohjelmiston täydellinen lähdekoodi on ladattavissa osoitteesta http://code.google.com/p/web-todo/. Ohjelmisto on julkistettu GLPv3 lisenssillä, joka tarkoittaa että se on avoimen lähdekoodin ohjelmisto.

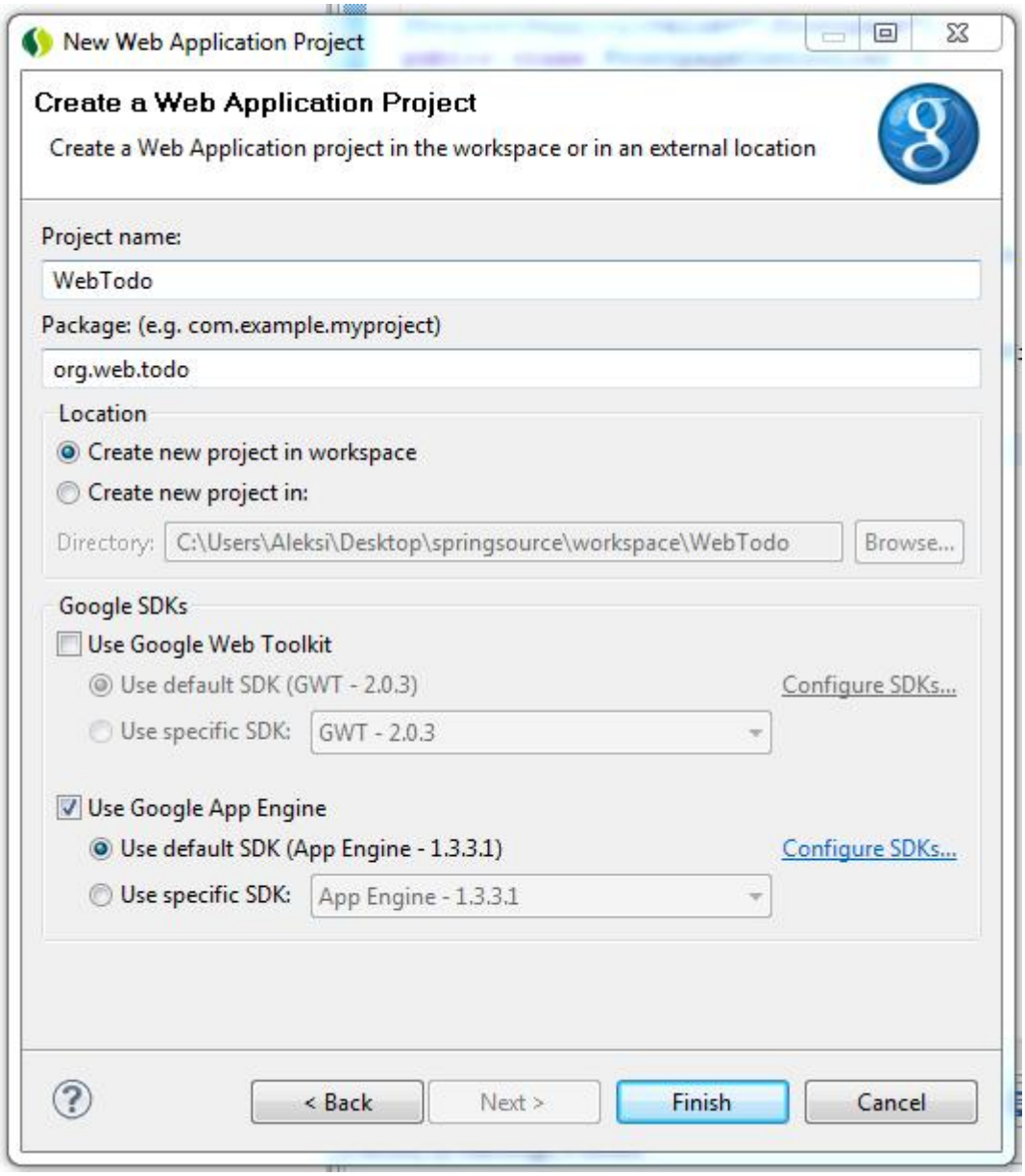
4.2 Projektin luominen STS:n avulla

STS:n avulla voi luoda valmiiksi projektipohjan Google App Engineä varten (Kuva 7). STS:ään pitää asentaa Google App Engine Eclipse liitännäinnen [37], jotta STS:llä voi luoda Google App Enginelle valmiita projektipohjia, ajaa paikallisella koneella kehitysversioita ja asentaa ohjelman Google App Engineen.



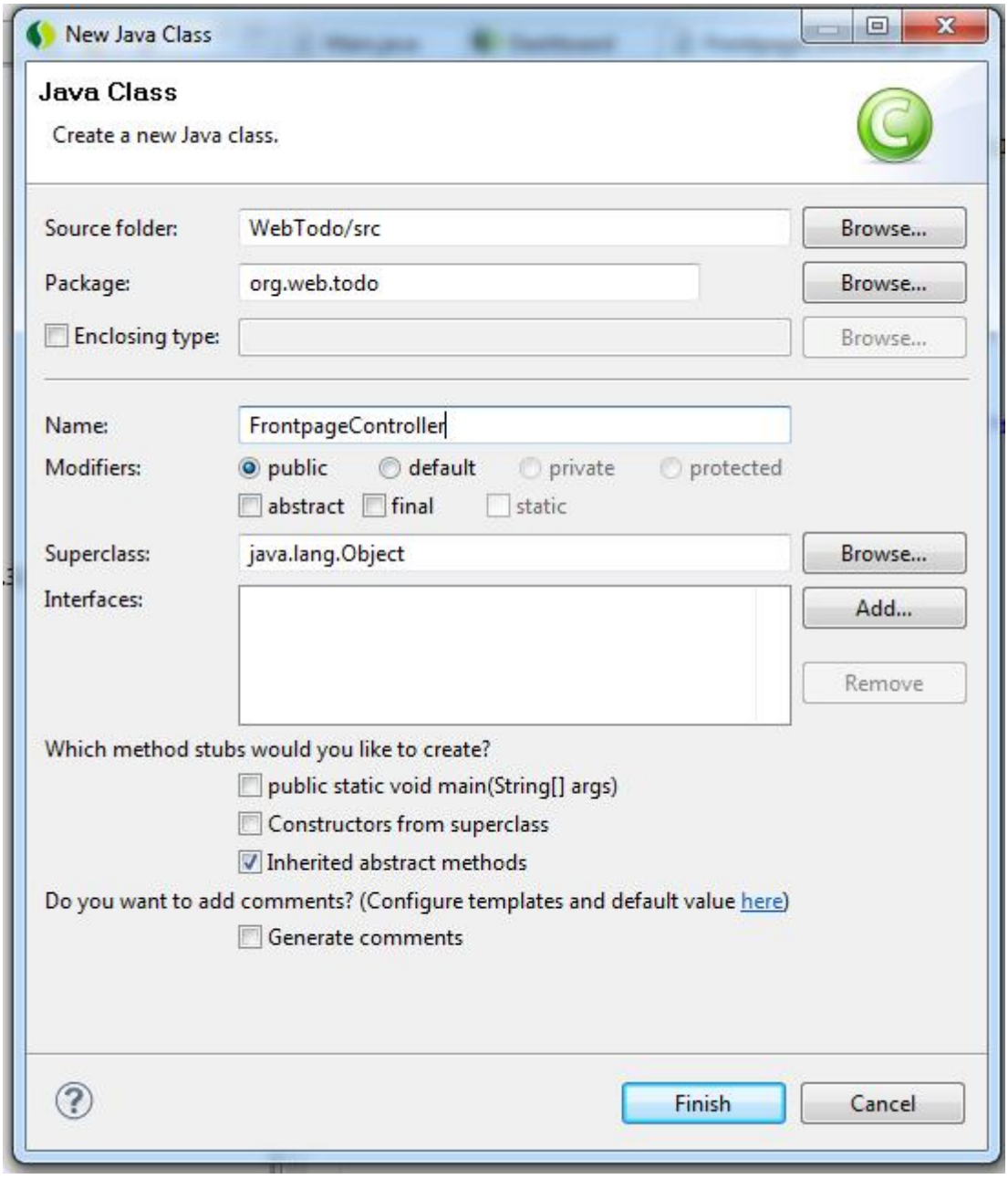
Kuva 7 Uuden Google App Enginen WWW-sovellusprojektin luominen STS:ssä.

Uuden projektin asetuksista pitää ottaa pois käytöstä ruksi Google Web Toolkit kohdalta, koska sitä ei käytetä tässä toteutuksessa lainkaan (Kuva 8).



Kuva 8 WWW-sovellusprojektin asetusten määrittäminen.

Luokkien luominen STS:n avulla onnistuu valitsemalla valikosta File -> New -> Class, jolloin esille tulee velho, jossa voidaan määritellä luokan nimi ja muut asetukset (Kuva 9).



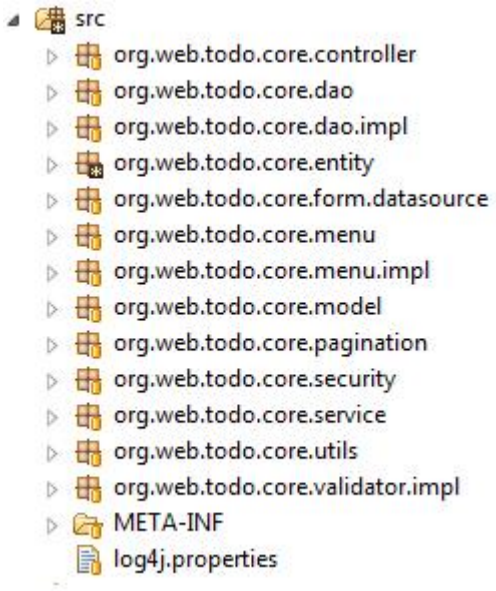
Kuva 9 Luokkien luominen STS:ssä.

4.3 Ohjelman asetukset

WWW-sovelluksen asetukset määritetään web.xml tiedostoon. Java-lähdekoodit sijoittuvat src kansioon ja pakettien nimeämisessä käytetään yleisiä käytäntöjä. Lokalistointi toteutettiin Spring Frameworkin tarjoamien kirjastojen avulla.

4.3.1 Java pakettien nimeäminen

Java-paketit nimetään hierarkkisesti siten, että kaikkien pakettien alussa on org.web.todo (Kuva 10). Kaikkien pakettien nimet tulee luoda sillä tavalla, että pakettiin sijoittuvien luokkien yleisen tarkoituksen voi ymmärtää pelkän paketin nimen lukemisella.



Kuva 10 Kuva ohjelmiston Java pakettien rakenteesta.

4.3.2 Web.xml-tiedosto

Web.xml tiedosto kertoo palvelimelle, kuinka ohjelma tulisi ottaa käyttöön. Web-xml-tiedostoon pitää määritellä Spring Frameworkin asennusluokka ja asetustiedostojen sijainti (Kuva 11). Sinne täytyy myös määritellä Spring Security ja Tuckey Url Rewrite filtterin asetukset.

```
<!-- Spring Framework dispatcher -->
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
    <!--
        Note about load-on-startup: the load actually occurs during the first
        request handled by the web server instance, not prior to it. (from
        http://code.google.com/appengine/docs/java/config/webxml.html#web_xml_Features_Not_Supported)
    -->
</servlet>
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/app/*</url-pattern>
</servlet-mapping>
```

Kuva 11 Esimerkki Spring Frameworkin asetuksista web.xml-tiedostossa.

4.3.3 SSL-suojauksen käyttöönotto

Google App Engine tukee SSL-suojausta silloin, kun sivustolle mennään *.appspot.com-osoitteen kautta. SSL-suojauksen asettaminen tapahtuu lisäämällä web.xml-tiedostoon kuvan 12 mukainen määrittely jolloin kaikkien sivujen pitäisi olla SSL-suojattuja. Jos käyttäjä yrittää mennä sivulle http-osoitteen kautta, ohjaa Google App Engine käyttäjän vastaavaan https-osoitteeseen. Tällä tavoin käyttäjien tiedot eivät ole esimerkiksi avoimissa WLAN.yhteyksissä kaikkien verkossa olevien käyttäjien luettavissa. SSL-salaus voidaan ottaa käyttöön vaikkapa vain kirjautumissivulla. Tietoturvan kannalta on järkevä tapa on suojata kaikki WWW-sovelluksen sivut, joilla käydessä lähetetään istunnon eväste palvelimelle, koska evästeen varastaminen on useimmissa tapauksissa aivan sama kuin hyökkääjä saisi selville käyttäjän käyttäjätunnuksen ja salasanan.

```
<!-- Enable SSL for all pages -->
<security-constraint>
    <web-resource-collection>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Kuva 12 SSL-suojauksen käyttöönotto.

4.3.4 Lokalisointi

Ohjelman lokalisointi toteutettiin Spring Frameworkin org.springframework.context.support.ReloadableResourceBundleMessageSource luokan avulla. Lokalisointi pitää määrittää Spring Frameworkin asetustiedostossa dispatcher-servlet.xml:ssä, jonne lisätään tarvittavat määrittelyt (Kuva 13).

```
<!-- Locale change interceptor -->
<bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"/>

<!-- Locale resolver -->
<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver">
    <property name="defaultLocale" value="en" />
</bean>

<!   Message Source   >
<bean id="messageSource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
    <property name="basenames">
        <value>/WEB-INF/messages/messages</value>
    </property>
</bean>
```

Kuva 13 Lokalisoinnin asetukset dispatcher-servlet.xml:ssä.

Lokalisointi on toteutettu siten, että WWW-sovelluksen war-hakemistoon lisätään esimerkiksi messages_fi.properties tiedosto, jossa tiedostonimessä messages_ on kielitiedostojen alkuosa ja fi taas kertoo, minkä kielen käännökset löytyvät kyseisestä tiedostosta. Viestitiedostot noudattavat i18n formaattia, jossa rivin alussa on viestin avainsana ja =-merkin toisella puolella on taas lokalisoitu viesti (Kuva 14). Avainsanojen nimeämisessä kannattaa käyttää selkeää logiikkaa, jotta avainsanan lukemalla saa selville, missä yhteydessä viestiä tarvitaan. Toteutuksessa valittiin avainsanojen kieleksi englanti aivan kuten itse ohjelmakoodissakin.

```
# Frontpage messages
frontpage.title = Login

# Admin
admin.title = Admin
admin.users.empty = Application has no users.

# Projects
projects.title = Projects
projects.completed = Completed projects
projects.projects.empty = You don't have any projects, please click "New project" link to add a new projec

# Project
project.title = Project
project.new.project = New project
project.new.task = New task

# Tasks
tasks.title = Tasks

# Login
login.username = Username
login.password = Password
login.login = Login
login.register = Don't have an account?
authentication-failed = Authentication failed
login.remember.me = Remember me
```

Kuva 14 Lokalisoidut tekstit messages.properties tiedostossa.

Jotta itse ohjelman kaikki viestit tulisivat lokalisoiduiksi, täytyy kaikissa näkymissä ja liiketoimintalogiikassa käyttää viestien avaimia ja lokalisointiluokkia viestien lokalisoimiseksi. Jsp-sivuilla käytetään Jsp-teknologian fmt merkintäkirjastoa, jonka avulla viestit saadaan helposti lokalisoitua (Kuva 15).

```
<div class="footer">
    <p><fmt:message key="common.footer.copyright"/></p>
</div>
```

Kuva 15 Lokalisointi Jsp-tiedostoissa.

Spring Security-viestien lokalisointi onnistuisi samalla tavalla lisäämällä Spring Securityn käyttämät kielitiedostot Spring Frameworkin asetuksiin. Mutta koska Spring Securityssä on oletuskielenä englanti ja toteutetun ohjelman oletuskielenä oli myös englanti, mitään käännöstyötä ja Spring Frameworkin asetuksia ei tarvinnut tehdä.

4.4 Mallien toteuttaminen

Pysyvän tiedon tallentamista varten malli-näkymä-kontrolleri skeemassa tietomalleja varten täytyy luoda oma rajapinta, joka sitten voidaan toteuttaa monella eri tavalla, jolloin tietokannan tai tietovaraston vaihtaminen tarkoittaa sitä että kun ohjelma halutaan liittää johonkin toiseen tietovarastoon, täytyy luoda vain uusi toteutus tietomallien rajapinnasta.

4.4.1 Tietoentiteetit

Tietoentiteetit ovat Java-luokkia (Kuva 16), jotka sisältävät samat tiedot kuin tietokannan tietomalli. Tietoentiteettien pitää noudattaa JavaBeans-nimeämiskäytäntöjä. Tietoentiteettien luomiseen käytetään Java Persistence API-annotaatioita, joka on virallinen Java teknologia. Google App Enginessä kaikkien tietoentiteettien pitää olla serialisoituja.

```java
package org.web.todo.core.entity;

import java.io.Serializable;

/**
 * User model
 *
 * @author Aleksi Lahtinen
 *
 */
@PersistenceCapable(detachable="true")
public class User implements Serializable {
    private static final long serialVersionUID = 1L;

    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key key;

    @Persistent
    @Unique
    private String username;

    @Persistent
    private String password;

    @Persistent
    private String passwordSalt;

    @Persistent
    private String firstname;

    @Persistent
    private String lastname;

    @Persistent
    private String email;

    @Persistent
    private Boolean emailVerified = false;

    @Persistent
    private String role;

    @Persistent
    private Boolean accountNonExpired;

    @Persistent
    private Boolean accountNonLocked;
```

Kuva 16 Esimerkki tietoentiteetistä.

4.4.2 Data Access Object

Jokaiselle tietoentiteetille luodaan oma DAO-rajapintamalli (Kuva 17) [38], johon määritellään, minkälaisia kyselyitä tietokantaan voidaan tehdä. Tämä rajapinta voidaan sitten toteuttaa vaikka monelle eri tietovarastolle, jolloin ohjelmisto ei ole millään tavalla riippuvainen käytetystä tietovarastosta, vaan ainoastaan itse rajapinta pitää toteuttaa kun halutaan ottaa käyttöön jokin toinen tietovarasto.

ProjectDao.java

```
1  package org.web.todo.core.dao;
2
3  import java.util.List;
6
7  public abstract class ProjectDao extends BaseDao {
8      public abstract List<Project> getProjects(int min, int max);
9      public abstract Integer getProjectsCount();
10 }
11
```

Kuva 17 Esimerkki DAO rajapinnasta.

DAO-rajapinnan toteutus (Kuva 18) voidaan Spring Frameworkin avulla injektoida käytetyissä luokissa asettamalla toteutus luokkaan annotaatio @Service. Google App Enginessä rajapinnan voisi toteuttaa JDO- tai JPA-teknologian avulla. Toteutetussa ohjelmistossa rajapinta toteutettiin JDO-teknologian avulla.

```
1  package org.web.todo.core.dao.impl;
2
3  import java.util.List;
14
15 @Service(value="projectDao")
16 public class ProjectDaoImpl extends ProjectDao {
17
18     @SuppressWarnings("unchecked")
19     @Override
20     @Transactional
21     public List<Project> getProjects(int min, int max, Key userKey) {
22         PersistenceManager pm = getPersistenceManager();
23         List<Project> projects, detached = null;
24         try
25         {
26             Query query = pm.newQuery(Project.class, "user == userParam");
27             query.declareParameters("com.google.appengine.api.datastore.Key userParam");
28             query.setRange(min, max);
29             projects = (List<Project>) query.execute(userKey);
30             detached = (List<Project>) pm.detachCopyAll(projects);
31         }
32         finally
33         {
34             pm.close();
35         }
36         return detached;
37     }
38
39     @Override
40     public Integer getProjectsCount() {
41         PersistenceManager pm = getPersistenceManager();
42         int count = 0;
43         try
44         {
45             Query query = pm.newQuery("select from " + Project.class.getName());
46             query.setResult("count(key)");
47             count = (Integer) query.execute();
48         }
49         finally
50         {
51             pm.close();
52         }
53         return count;
54     }
55
```

Kuva 18 Esimerkki DAO toteutuksesta.

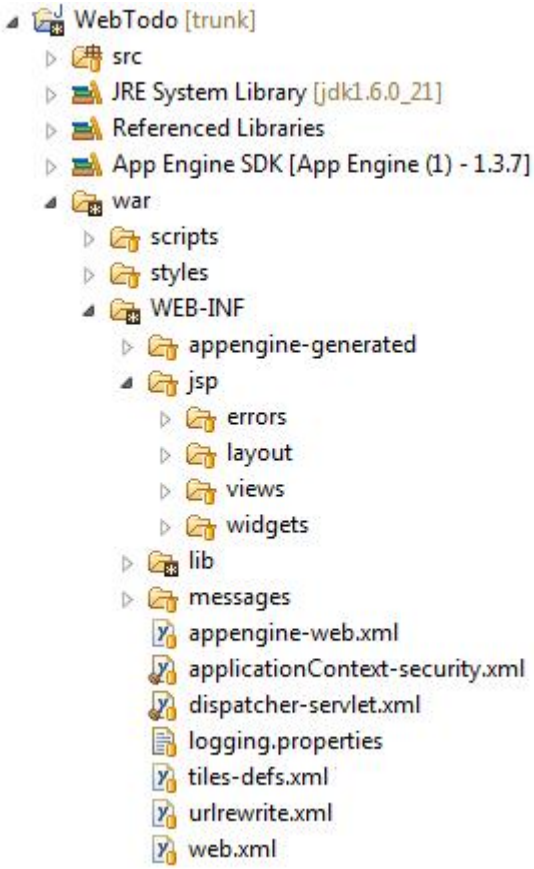
4.4.3 Http-istunnot Google App Enginessä

Google App Enginen istuntoihin tallennetut tiedot tallennetaan Googlen tietovarastoon samalla tavalla kuin tietoentiteetit [39], mikä tarkoittaa sitä, että myös istuntoihin tallennettujen objektien pitää olla serialisoitavissa ja ne tukevat vain Google App Enginen tietovaraston tukemia tietotyyppejä. Tämä tarkoittaa sitä, että istuntoon ei voi tallentaa tällä hetkellä esimerkiksi HashMap-luokkia, vaikka JDO teknologia tukeekin tietoentiteeteissä HashMap luokkaa, Google ei tällä hetkellä tue kyseistä luokkaa.

4.5 Näkymien toteuttaminen

Spring Frameworkissa näkymät voidaan totetuttaa millä tahansa näkymä-teknologialla. Toteutus tehtiin Jsp (Java Server Pages) [40] -teknologoiden ja Tiles 2 -frameworkin

avulla. Spring Frameworkissa kaikki näkymät lisätään JavaBeaniksi Spring Frameworkin kontekstiin ja niille annetaan nimet, joidenka avulla kyseiset JavaBeanin voidaan injektoida kontekstiin, joissa niitä käytetään. Tiles2 -frameworkissa määritellään tiles-defs.xml XML-tiedosto, johon määritellään sivujen mallit. Jsp-sivut tallennettiin war kansion WEB-INF kansioon (Kuva 19), jotta niihin pääsee käsiksi vain WWW-sovelluksen kontrollerit.



Kuva 19 WWW-sovelluksen kansiorakenne.

4.6 Kontrollerien toteuttaminen

WWW-ohjelmiston toteuttamisessa Javalla käytetään Servlet-teknologiaa ohjaimien toteuttamiseen. Spring Framworkissa lähettäjä Servlet vastaa kaikkiin sovelluksen pyyntöihin ja ohjaa ne Spring Frameworkin ohjaimelle, joilla on käytössä kaikki servleteilläkin käytössä olevat toiminnot. Ohjaimet luodaan käyttämällä Springin @Controller -annonnaatioita (Kuva 20), joiden avulla Spring pystyy löytämään luokat, jotka halutaan asettaa ohjaimiksi. Spring Frameworkin MVC-malli WWW-sovelluksille ei ole aivan puhtaasti MVC-mallin mukainen, sillä ohjaimissa on käytössä asetuksia,

jotka ovat tyypillisiä vain WWW-sovelluksille, esimerkiksi Servlet API:n rajapinnat ovat vapaasti käytettävissä.

```
1  package org.web.todo.core.controller;
2
3  import org.springframework.stereotype.Controller;
8
9  @Controller
10 @RequestMapping(value="/" + Path.FRONTPAGE)
11 public class FrontpageController {
12
13     /**
14      * Frontpage view
15      * @return
16      */
17     @RequestMapping(method=RequestMethod.GET)
18     public ModelAndView frontpage()
19     {
20         ModelAndView mav = new ModelAndView("frontpage");
21         return mav;
22     }
23 }
24
```

Kuva 20 Esimerkki Spring Frameworkin ohjainluokasta.

4.7 Palveluiden toteuttaminen

Palvelun toteuttaminen tapahtuu määrittelemällä @Service annontaation avulla luokkapalveluksi (Kuva 21). Kun palvelu halutaan ottaa käyttöön Springin luomassa luokassa, tapahtuu se käyttämällä @Autowired annonaatioita.

```
@Service(value="projectDao")
public class ProjectDaoImpl extends ProjectDao {
```

Kuva 21 Luokan määrittely palveluksi.

@Autowired annonaatiot toimivat vain luokissa (Kuva 22), jotka Spring Framework on luonut. Jos Spring Framework ei ole luonut luokkaa, silloin Spring-pavun voi ladata toteutamalla org.springframework.context.ApplicationContextAware luokan.

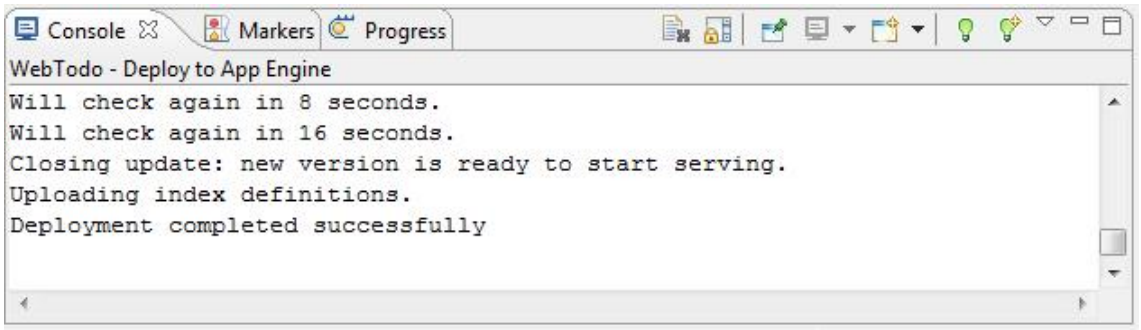
```
@Controller
@RequestMapping(value="/" + Path.PROJECTS + "/")
public class ProjectController {

    @Autowired private ProjectDao projectDao;
    @Autowired private  MessageSource messageSource;
```

Kuva 22 Palvelun käyttöönotto @Autowired-annontaation avulla.

Palveluiden tarkoitus on eristää liiketoimintalogiikka palveluiden taakse, jotka tarjoavat selkeän rajapinnan tietylle liiketoimintalogiikan osalle. Tällöin liiketoimintalogiikan toteutus ei sekoitu kontrollerien ohjelmistokoodiin ja kontrollerit keskittyvät vain toimimaan palveluiden välittäjinä Http-kyselyille. Palveluiden käytöstä on hyötyä silloin jos ohjelmiston näkymät pitääkin toteuttaa johonkin toiseen käyttöliittymään, jolloin itse ohjelman logiikka on sisällytetty palveluiden taakse.

4.8 Ohjelman asentaminen Google App Engineen

Ohjelman asennus Googlen App Engineen tapahtuu Googlen App Engine Eclipse-liitännäisellä, jonka voi lisätä osoitteesta http://dl.google.com/eclipse/plugin/3.5 Eclipsen päivitystoiminnon avulla. Kun liitännäinen on asennettu, pitää ohjelman Google App Engine tunniste lisätä projektiin Google App Engine liitännäisen asetuksista. Ohjelman voi lähettää Google App Engineen valitsemalla Googlen liitännäisen valikosta Deploy to App Engine linkin, joka taas kysyy Google tilin tunnuksia. Tämän jälkeen STS:n konsolissa pitäisi näkyä kuinka ohjelmiston lähetys onnistuu ja kun asennus on tapahtunut onnistuneesti ilmoitetaan siitä konsolissa, jos taas asennus on epäonnistunut ilmoitetaan konsolissa syy jonka takia asennus epäonnistui (Kuva 23).



Kuva 23 Onnistunut asennus Googlen App Engineen.

4.9 Valmiin ohjelman ajaminen ja testaus

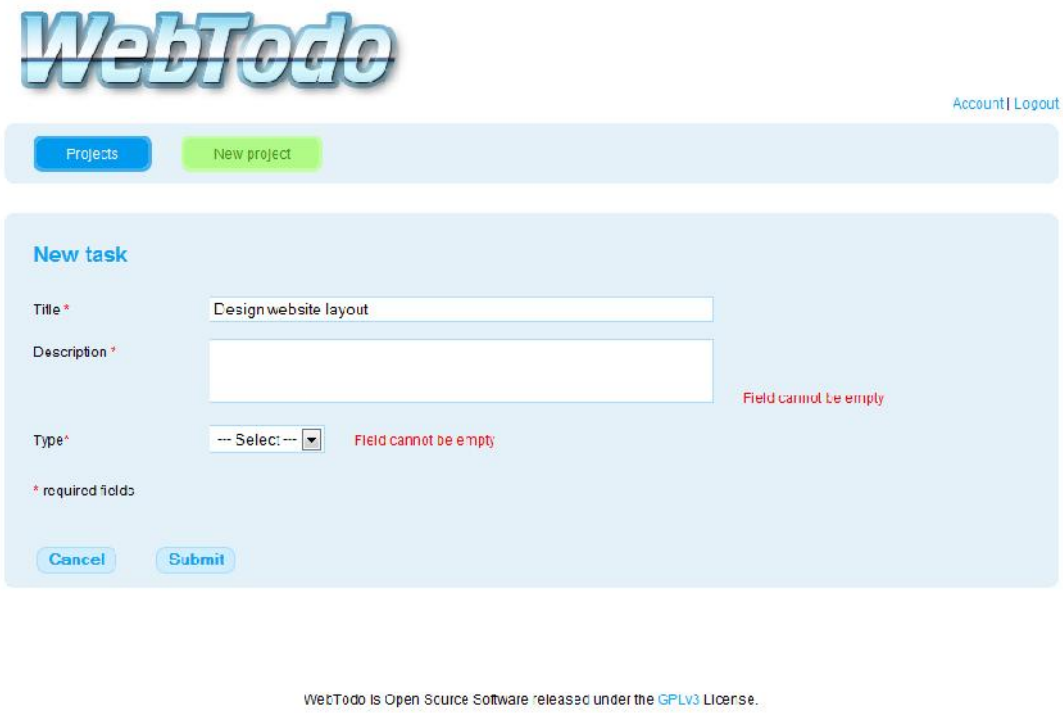
Ohjelman voi käynnistää STS:stä valitsemalla projektin, jonka valikosta valitaan Debug as ja valikon alavalikosta valitaan Web application-vaihtoehto. Kun ohjelman käynnistää STS:stä on ohjelman osoite oletuksena http://localhost:8888/ ja ohjelma ohjaa käyttäjä kirjautumissivulle. Kirjautumissivulta käyttäjän pitää rekisteröityä, minkä jälkeen ohjelmaan voi kirjautua sisään (Kuva 24).

Kuva 24 Ohjelman kirjautumissivu.

Ohjelman päänäkymä tavalliselle käyttäjälle on projektit näkymä (Kuva 25), josta voi luoda uusia projekteja ja selata kaikki projekteja sekä valita jonkin tietyn projektin tai poistaa projektin. Ohjelman ylävalikosta taas voi tarkastella ja muuttaa oman käyttäjätilinsä tietoja sekä kirjautua ulos sovelluksesta.

Kuva 25 Ohjelman projektit -näkymä.

Kaikissa ohjelman lomakkeissa on käytössä kenttien arvojen tarkastus, jos jonkin kentän jättää täyttämättä ja se on pakollinen kenttä, ohjelma antaa siitä virheilmoituksen (Kuva 26).



Kuva 26 Ohjelman uuden tehtävän lisäys -näkymä, jossa käyttäjälle on ilmoitettu puuttuvista tiedoista.

4.10 Ohjelman toiminnallisuuden testaus

WWW-sovelluksen toiminnallisuuden testaaminen eri selaimilla on tärkeää, jolloin ohjelman toiminnallisuus voidaan taata suuremmalle käyttäjäkunnalle kuin jos sovellus olisi testattu vain yhdellä selaimella. Ohjelmisto testattiin ja toteutettuun toimimaan kaikilla uusimilla Fireforx-, Safari-, Google Chrome- ja Internet Explorer-selaimilla.

Ohjelmiston toiminnallisuuden testausta voitaisiin automatisoida esimerkiksi vapaan lähdekoodin Selenium-ohjelmistolla [41], joka tukee kaikista suosituimpia selaimia ja niiden eri versioita.

5 Yhteenveto

Tuloksena syntyi WWW-sovellus, josta voi ottaa mallia ja oppia, kun alkaa kehittää ensimmäistä Google App Engine sovellustansa. Googlen App Engine on vielä beta-vaiheessa, jonka myös huomaa kehitystyökalujen ja itse pilven kypsyystasosta. Suurin ongelma Googlen App Enginessä on tietovarasto, jonne voi tehdä vain erityisen

rajoittuneita kyselyitä. Google App Engine tarjoaa hyvän pilvialustan starttiyrittäjille, jotka haluavat mahdollisimman pienillä kehityskuluilla luoda skaalautuvan WWW-palvelun.

Työn alussa WebTodo-tehtävälista WWW-sovellukselle asetetuista vaatimuksista toteutettiin kaikki. Sovelluksen toteutus oli erittäin haastavaa sillä Google App Engine on vielä uusi pilvipalvelu markkinoilla ja se kehittyy koko ajan. Sovelluskirjastojen saaminen toimimaan Googlen App Enginessä oli haasteellista, mutta kun Google App Enginen tarjoamat ongelmakohdat sai ratkaistua, oli sovelluksen toteuttaminen käytettyjen sovelluskirjastojen ansiosta melko helppoa.

LÄHTEET

1. Google App Engine. [www-dokumentti] Saatavilla: http://code.google.com/intl/fi-FI/appengine/. (Luettu 10.10.2010).
2. Spring Framework. [www-dokumentti] Saatavilla: http://www.springsource.org/. (Luettu 10.10.2010).
3. Peltomäki Juha, Silander Simo, Pekka Kosonen: Java 2 - ohjelmoinnin peruskirja. Docendo Finland Oy; 2005.
4. IBM: Java programming dynamics, Part 1: Java classes and class loading. [www-dokumentti] Saatavilla: http://www.ibm.com/developerworks/java/library/j-dyn0429/. (Luettu 10.10.2010).
5. Ortiz Ramirez Ariel: Three-Tier Architecture. [www-dokumentti] Saatavilla: http://www.linuxjournal.com/article/3508. (Luettu 10.10.2010).
6. Oracle Technology Network: Model-View-Controller Also Known As MVC. [www-dokumentti] Saatavilla: http://www.oracle.com/technetwork/java/mvc-140477.html. (Luettu 10.10.2010).
7. Eccentex Corporation: Dynamic Case Management Overview. [www-dokumentti] Saatavilla http://www.eccentex.com/platform/platform.html. (Luettu 10.10.2010).
8. Hinchcliffe Dion: Comparing Amazon's and Google's Platform-as-a-Service (PaaS) Offerings. [www-dokumentti] Saatavilla: http://www.zdnet.com/blog/hinchcliffe/comparing-amazons-and-googles-platform-as-a-service-paas-offerings/166?tag=btxcsim. (Luettu 10.10.2010).
9. Microsoft: Patterns For High Availability, Scalability, And Computing Power With Windows Azure. [www-dokumentti] Saatavilla: http://msdn.microsoft.com/en-us/magazine/dd727504.aspx. (Luettu 10.10.2010).
10. Henderson Tom, Allen Brendan, Network World: Cloud storage lives up to the hype. [www-dokumentti] Saatavilla: http://www.networkworld.com/reviews/2010/082310-cloud-storage-test.html. (Luettu 10.10.2010).

11. Microsoft: Windows Azure Platform. [www-dokumentti] Saatavilla: http://www.microsoft.com/windowsazure/. (Luettu 10.10.2010).

12. Amazon: Amazon Elastic Compute Cloud (Amazon EC2). [www-dokumentti] Saatavilla: http://aws.amazon.com/ec2/. (Luettu 10.10.2010).

13. Amazon: Amazon Elastic Block Store (EBS). [www-dokumentti] Saatavilla: http://aws.amazon.com/ebs/. (Luettu 10.10.2010).

14. Google: What Is Google App Engine? [www-dokumentti] Saatavilla: http://code.google.com/intl/fi-FI/appengine/docs/whatisgoogleappengine.html. (Luettu 10.10.2010).

15. ComputerWeekly.com: Write once, run anywhere? [www-dokumentti] Saatavilla: http://www.computerweekly.com/Articles/2002/05/02/186793/write-once-run-anywhere.htm. (Luettu 12.6.2010).

16. TIOBE Software BV: TIOBE Programming Community Index for May 2010. [www-dokumentti] Saatavilla: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. (Luettu 31.5.2010).

17. Free Software Foundation, Inc., GNU General Public License. [www-dokumentti] Saatavilla: http://www.gnu.org/licenses/gpl-3.0.html. (Luettu 12.6.2010).

18. Oracle: Java Reflection API. [www-dokumentti] Saatavilla: http://download.oracle.com/javase/tutorial/reflect/. (Luettu 10.10.2010).

19. Oracle: Java Beans. [www-dokumentti] Saatavilla: http://download.oracle.com/javase/tutorial/javabeans/. (Luettu 10.10.2010).

20. Oracle: Java 2 Platform, Enterprise Edition (J2EE) Overview. [www-dokumentti] Saatavilla: http://java.sun.com/j2ee/overview.html. (Luettu 10.10.2010).

21. Dependency Injection and Inversion of Control. [www-dokumentti] Saatavilla: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/overview.html#overview-dependency-injection . (Luettu 10.10.2010).

22. Apache: Tiles 2. [www-dokumentti] Saatavilla: http://tiles.apache.org/. (Luettu 10.10.2010).

23. Tuckey Paul: Rewrite URL's in Java Web Application Servers. [www-dokumentti] Saatavilla: http://www.tuckey.org/urlrewrite/. (Luettu 10.10.2010).

24. Spring Security. [www-dokumentti] Saatavilla: http://static.springsource.org/spring-security/site/. (Luettu 10.10.2010).

25. Google: Billing and Budgeting Resources. [www-dokumentti] Saatavilla: http://code.google.com/intl/fi-FI/appengine/docs/billing.html. (Luettu 10.10.2010).

26. Google: The JRE Class White List. [www-dokumentti] Saastavilla: http://code.google.com/appengine/docs/java/jrewhitelist.html. (Luettu 12.06.2010).

27. Google: GQL Reference. [www-dokumentti] Saatavilla: http://code.google.com/intl/fi-FI/appengine/docs/python/datastore/gqlreference.html. (Luettu 10.10.2010).

28. Google: Datastore Java API Overview. [www-dokumentti] Saatavilla: http://code.google.com/intl/fi-FI/appengine/docs/java/datastore/overview.html. (Luettu 2.6.2010).

29. Google: Bigtable: A Distributed Storage System for Structured Data. [www-dokumentti] Saatavilla: http://labs.google.com/papers/bigtable.html. (Luettu 12.6.2010).

30. Luukkainen Matti ja Nykänen Matti: 58131: Tietorakenteet. [www-dokumentti] Saatavilla: http://www.cs.helsinki.fi/u/jkivinen/opetus/tira/k07/tirakalvot.pdf. (Luettu 12.6.2010).

31. Oracle: Java Data Objects (JDO). [www-dokumentti] Saatavilla: http://www.oracle.com/technetwork/java/index-jsp-135919.html. (Luettu 10.10.2010).

32. Oracle: The Java Persistence API - A Simpler Programming Model for Entity Persistence. [www-dokumentti] Saatavilla:

http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html. (Luettu 10.10.2010).

33. Oracle: Which Persistence Specification? [www-dokumentti] Saatavilla: http://db.apache.org/jdo/jdo_v_jpa.html. (Luettu 11.10.2010).

34. Google: Issues. [www-dokumentti] Saatavilla: http://code.google.com/p/googleappengine/issues/list. (Luettu 2.11.2010).

35. Google: App Engine Product Roadmap. [www-dokumentti] Saatavilla: http://code.google.com/intl/fi-FI/appengine/docs/roadmap.html. (Luettu 2.11.2010).

36. SpringSource Tool Suite -- The Best Development Tool for Enterprise Java. [www-dokumentti] Saatavilla: http://www.springsource.com/developer/sts. (Luettu 10.10.2010).

37. Goolge: Using the Google Plugin for Eclipse. [www-dokumentti] Saatavilla: http://code.google.com/intl/fi-FI/appengine/docs/java/tools/eclipse.html. (Luettu 10.10.2010).

38. Orcale: Core J2EE Patterns - Data Access Object. [www-dokumentti] Saatavilla: http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html. (Luettu 10.10.2010).

39. Google: Java Application Configuration. [www-dokumentti] Saatavilla: http://code.google.com/intl/fi-FI/appengine/docs/java/config/appconfig.html#Enab . (Luettu 10.10.2010).

40. Oracle: JavaServer Pages Technology. [www-dokumentti] Saatavilla: http://java.sun.com/products/jsp/. (Luettu 10.10.2010).

41. Selenium: How Selenium Works. [www-dokumentti] Saatavilla: http://seleniumhq.org/about/how.html. (Luettu 11.10.2010).