Juha Lähteenmaa

Verkkosovelluksen ohjelmistoteknologiat

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

03.05.2016

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Juha Lähteenmaa Verkkosovelluksen ohjelmistoteknologiat 49 sivua 03.05.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Simo Silander Lehtori Jussi Alhorinne

Tässä opinnäytetyössä tutustun työskentelemässäni yrityksessä käyttöön otettavaan verkkosovellukseen, jonka tarkoitus on tarjota käyttäjilleen alusta, johon he voivat tallentaa omia muistojaan tarinamuodossa käyttämällä useita eri mediatyyppejä. Sovellus on toteutettu alihankintana Intiassa, ja sovelluksen jatkokehitystä ollaan siirtämässä Suomeen.

Työ etenee verkkosovelluksen tarkoituksesta tekniseen esittelyyn, jossa kerrotaan tarkemmin, mitä teknologiavalintoja on tehty, miten komponentit keskustelevat toistensa kanssa ja minkälainen arkkitehtuuri sovelluksessa on. Teknologialuvussa esitellään myös ohjelmistokoodin sisäistä toiminnallisuutta ja sitä, miten tarinat haetaan tietokannasta sovelluksen käyttöliittymään.

Työn lopputuloksena syntyi tämä dokumentti, jonka tarkoitus on toimia kirjallisena apuvälineenä sovelluksen jatkokehityksessä.

Avainsanat	Red5, Java, Spring, MySQL, Hibernate, HDFVR, Bootstrap, JSP, Three.js

Author(s) Title Number of Pages Date	Juha Lähteenmaa Software technologies of Web Application 49 pages 03 May 2016
Degree	Bachelor of Engineering
Degree Programme	Software Engineering
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Senior Lecturer Jussi Alhorinne, Senior Lecturer

In this thesis, software technologies used in a web application were studied. The purpose of the web application is to provide a platform for the users to save their memories in story form using multiple media types. The web application has been developed by a subcontractor in India but the software development of the web application is now being relocated to Finland.

The thesis begins by describing the purpose of the web application. After this, there is a chapter that describes the technical side of the web application focusing on what selections of technology have been made, how the components of the application work with one another and what the architecture of the application is. In the technology chapter there is also a presentation regarding the inner functionality of the software code and how the stories are brought from the database to the application's user interface.

The outcome of this thesis is this document the purpose of which is to work as a written resource that aids the future software development of the web application.

Keywords	Red5, Java, Spring, MySQL, Hibernate, HDFVR, Bootstrap, JSP, Three.js

Sisällys

Lyhenteet ja sanasto

AJAX	Asynchronous JavaScript And XML. Asynkroninen tiedonsiirtotekniikka, joka mahdollistaa datan välittämisen verkkosivun ja palvelimen välillä siten, ettei sivua tarvitse ladata uudestaan.
Annotaatio	Merkinnät sovellusluokissa, jonka avulla opastetaan Spring-sovelluskehystä IoC-säiliön konfiguroinnissa.
AOP	Aspect Oriented Programming. Aspektipohjainen sovelluskehitys, jolla tarkoitetaan sitä, että sovelluksen toiminnallinen logiikka ja muut siihen liittymättömät sovelluksen osat pidetään irrallaan toisistaan.
CSS	Cascading Style Sheets. Porrastetut tyyliarkit. Näiden avulla säädetään informaation esitystapaa verkkosivuilla.
dispatcher-servlet.xml	Tiedosto, jonka avulla määritellään, miten Spring Web MVC -sovelluskehys on konfiguroitu.
Enkryptaus	Tiedon salaaminen käyttäen tiedonsalausmenetelmää.
Hibernate	Sovelluskehys, jonka avulla oliopohjainen sovelluskoodi muutetaan relaatiotietokantaan sopivaksi.
HDFVR	High Definition Flash Video Recorder. Apuohjelmisto, jonka avulla voidaan nauhoittaa palvelun käyttäjän videokuvaa tai ääntä.
HTML	HyperText Markup Language. Hypertekstin merkintäkieli. Kieli jolla Internetsivut on kirjoitettu.
IoC	Inversion of Control. Spring-sovelluskehyksen käyttämä teknologia jossa sovelluksen kontrolli käännetään sovelluskehykselle.

IoC-säiliö	Spring-sovelluskehykseen kuuluva säiliö, joka antaa sovelluskehykselle kyvyn konfiguroida ja hallinnoida Java-olioita.
JAR	Java Archive. Java-sovellusalustan käyttämä tiedostoformaatti joka kokoaa useita Java-luokkia, metadataa ja resursseja (teksti, kuvat ym.) yhdeksi tiedostoksi.
JDBC	Java Database Connectivity. Teknologia, jonka avulla Hibernate ottaa yhteyden tietokantaan.
JSON	JavaScript Object Notation. Tiedostomuoto, jonka avulla tietoa voidaan siirtää attribuutti-arvo -parien avulla.
JSP	JavaServer Pages. Teknologia, jonka avulla voidaan luoda dynaamista sisältöä verkkosivuille.
JVM	Java Virtual Machine. Java-virtuaalikone jolla voidaan suorittaa Javalla tehtyjä, tavukoodiksi käännettyjä ohjelmia.
Kenttä	Tietokannan taulussa oleva osa, jonka avulla voidaan tunnistaa tietokannan sarakkeissa olevan informaation tyyppi.
Konfigurointi	Asetusten määrittäminen.
Koontiversio	Sovelluksen käännetty versio, jota voidaan ajaa palvelimella.
Kääntäjä	Tietokoneohjelma, jonka avulla kehittäjän kirjoittama sovelluskoodi käännetään tietokoneen suorittimen ymmärtämään muotoon.
Lokalisointi	Toimintatapa, jonka avulla ohjelmisto sovitetaan toimimaan vieraissa maissa (esim. päiväys, kieli, valuutta).
Maven	Koontityökalu, jonka avulla määritellään, miten sovelluskoodista luodaan koontiversio ja mitä riippuvuuksia sovelluksella on.

Mappaus	Kuvaus siitä miten asiat ovat kytketty toisiinsa.
null	Merkintä, jota käytetään, kun viitattua oliota ei ole olemassa.
Ohjelmistopalvelu	Palvelu, joka on toteutettu ohjelmiston muodossa.
Ohjelmointialusta	Digitaalinen verstas, jonka sisältämien työkalujen avulla sovelluksia voidaan rakentaa.
ORM	Object Relational Mapping. Olio-relaatiomappaus, jonka avulla automatisoidaan tiedon esitysmuodon vaihtaminen oliopohjaisesta relaatiopohjaiseen.
Olio	Sovellukseen ohjelmoidun luokan ilmentymä, joka sisältää sisäisen tietorakenteen.
POM	Project Object Model. Maven-koontityökalun käyttämä tiedosto, joka sisältää projektin yksilöllisen tunnisteen ja sovelluksen sisäiset riippuvuudet.
Rajapinta	Sovelluskirjastoon määritellyt metodit, joiden avulla sovelluksia voidaan kehittää.
Redundanssi	Informaation toisteisuus.
Renderointi	Grafiikan piirtäminen käyttäen ohjelmallista opastusta.
Riippuvuusinjektio	Menetelmä, jolla tarkoitetaan sovellusluokkien yhdistämistä toisiinsa sovelluskehyksen toimesta.
SaaS	Software as a Service. Pilvilaskennan malli jossa käyttäjälle tarjotaan ohjelmistosovellus palveluna.
Servlet	Java-teknologia, jonka avulla laajennetaan verkkopalvelimen toiminnallisuutta.

Sovelluskehys	Ryhmä luokkia, rajapintoja ja esikäännettyä koodia, jonka päällä sovelluksia voidaan kehittää.
Tag	Yksilöllinen tunniste.
Taulu	Tietokantaan tallennetun informaation rakennekuvaus. Sisältää informaatiota sekä rivi-, että sarakemuodossa.
Tietokanta	Kokoelma toisiinsa liittyvää järjestäytynyttä tietoa.
Tietue	Tietokannan taulun yksittäinen rivi.
Transaktio	Kokoelma peräkkäin suoritettavia toimenpiteitä, joiden tulee kaikkien onnistua, että transaktio voidaan luokitella onnistuneeksi.
URL	Uniform Resource Locator. Merkkijono, joka kertoo WWW-sivun sijainnin.
Verkkosovellus	Käyttöliittymällä varustettu ohjelmisto, johon loppukäyttäjällä on pääsy Internetin ylitse.
WAR	Web application ARchive. JAR-tiedosto, joka on tarkoitettu JSP-sivujen, Java Servletien, Java-luokkien, XML-tiedostojen, tagi-kirjastojen sekä staattisten Web-sivujen jakamiseen.
web.xml	Web Deployment Descriptor. Tiedosto, jonka avulla määritellään, miten verkkosovellus otetaan käyttöön palvelimella. Käytetään servlet-teknologiaan pohjautuvissa sovelluksissa.
WWW	World Wide Web. Yksi Internetin palvelumuodoista joka mahdollistaa surffaamisen verkkosivuilla.
XML	Extensible Markup Language. Tiedon kuvauskieli jossa tiedon merkitys on kuvattu tiedon sekaan.

1 Johdanto

Tässä insinöörityössä tutustun työskentelemässäni yrityksessä käyttöön otettavaan verkkosovellukseen. Yritys on lanseeraamassa uuden kaupallisen SaaS (Software as a Service) -ohjelmistopalvelun ja tarvitsee apua palvelun jatkokehityksessä. Ohjelmistopalvelu on toteutettu alihankintana Intiassa, mikä asettaa tietynlaisia rajoitteita palvelun kehittämisen suhteen. Koska yritys on alkutaipaleella ja kommunikaatioketju on pitkä, vaatii uusien ominaisuuksien määrittely, suunnittelu, toteutus, testaus ja dokumentointi suhteellisesti merkittävää resurssien käyttöä. Jotta yrityksen toimintaa voidaan tehostaa ja vasteaikoja parantaa palvelun laadun ja toiminnallisuuden kehittämisessä, on looginen askel siirtää ohjelmistokehitys hiljalleen Suomeen.

Työssä edetään ohjelmistopalvelun käyttötarkoituksesta ohjelmistopalvelussa käytettyihin komponentteihin, mitä seuraa teknisen toteutuksen ja toiminnallisuuden esittely arkkitehtuurikuvineen. Työn loppupuolella esittelen, kuinka työskentely-ympäristö määritellään siten, että verkkosovellusta voidaan jatkokehittää Eclipse-kehitysympäristössä. Lopuksi sovelluksesta kasataan koontiversio, jota voidaan ajaa palvelimella. Suurin osa työstä käsittelee kuitenkin ohjelmistopalvelun teknistä toiminnallisuutta ja arkkitehtuuria, koska se toimii tärkeänä avustavana elementtinä Suomessa tapahtuvalle ohjelmistopalvelun jatkokehitykselle.

Koska opinnäytetyö on julkinen ja palvelussa käsitellään luottamuksellisia henkilötietoja, ohjelmistopalvelun turvallisuuden lisäämiseksi työstä on jätetty pois tiettyjä asioita kuten palvelun nimi, joiden avulla palvelu voitaisiin mahdollisesti tunnistaa ja työssä esitettävää informaatiota käyttää yritystä ja / tai palvelua vastaan.

2 Ohjelmistopalvelun tarkoitus

Opinnäytetyössäni tarkastelema ohjelmistopalvelu on muisto- ja elämäntarinapalvelu, jonka tarkoituksena on tarjota käyttäjilleen alusta, johon he voivat tallentaa henkilökohtaisia muistojaan ja elämäntarinoitaan tarinamuodossa. Käyttäjälle annetaan mahdollisuus kuvailla muistojaan neljän eri mediatyypin avulla. Ne ovat video, kuva, ääni sekä teksti.

Kuhunkin tarinaan on mahdollista lisätä useita välilehtiä, joiden avulla taltioitavan muiston kuvausta voidaan tarkentaa. Jos käyttäjä haluaa esim. kuvailla lapsuutensa kotitaloon liittyvää muistoa, on hänellä mahdollisuus tehdä niin käyttämällä kaikkia edellä mainittuja mediatyyppejä. Käyttäjä voi lisätä yhdelle välilehdelle kuvan omasta kotitalostaan, selostaa puhetta toiselle, kolmannelle lisätä videon pihapiiristä ja tarkentaa vielä muiston kuvailua lisäämällä tekstiselostuksen viimeiselle välilehdelle.

Pitkän tähtäimen suunnitelmana on kasvattaa ohjelmistopalvelu kansainväliseksi, turvalliseksi palveluksi, jota voidaan käyttää useissa eri kulttuureissa ja useilla eri kielillä.

2.1 Aika ja paikka

Koska muistoihin liittyy usein sekä aika että paikka, on palvelun toiminnallisuus toteutettu tätä ajatusta vaalien. Palvelu pohjautuu kahteen päänäkymään, joista toinen on aikajana ja toinen karttanäkymä. Aikajanaa käyttämällä käyttäjä voi liikkua 3D-näkymässä syvyyssuuntaisesti menneisyyden ja nykyhetken välillä ja nähdä omat tai muiden muistot ajallisessa järjestyksessä. Aika on jaoteltu vuosikymmeniin ja muistot on tallennettu aarrearkkuihin, joita klikkaamalla voidaan aukaista kyseessä oleva muisto. Karttanäkymässä käyttäjä voi tarkastella samoja muistoja, mutta muistot ovat tällöin ripoteltu kartalle sijainnin mukaan. Hakusuodattimia käyttämällä käyttäjä voi hakea muistoja esim. tietyltä aikaväliltä ja nähdä näiden muistojen ajallisen järjestyksen lisäksi myös muistojen sijainnin.

2.2 Tarinoiden näkyvyys

Lisättävien tarinoiden näkyvyyttä voidaan säätää oman tarpeen mukaisesti tallentamalla muisto joko julkiselle, yksityiselle tai yhteisön aikajanalle. Julkiselle aikajanalle tallennetut tarinat näkyvät kaikille palvelun käyttäjille mukaan lukien myös heille, jotka eivät ole palveluun vielä rekisteröityneet. Mikäli käyttäjä haluaa rajata tarinansa näkyvyyttä, hän voi asettaa näkyvyydeksi joko yksityisen tai yhteisön. Yksityisellä aikajanalla näkyvät muistot näkyvät vain käyttäjälle itselleen, ja tätä suositellaankin henkilökohtaisten muistojen tallentamiseen. Mikäli käyttäjä kuuluu johonkin yhteisöön tai haluaa perustaa sellaisen, voidaan näkyvyysvaihtoehdoksi asettaa yhteisö. Tällöin vain tähän tiettyyn yhteisöön kuuluvat jäsenet voivat tarkastella yhteisöön tallennettuja tarinoita.

3 Verkkosovelluksessa käytetyt komponentit

Työssä tarkasteltava ohjelmistopalvelu on verkkosovellus, joka tarkoittaa käyttöliittymällä varustettua ohjelmistoa, johon loppukäyttäjällä on pääsy Internetin ylitse. Verkkosovellus koostuu useista eri ohjelmistoteknologisista komponenteista, joista kullakin on oma tärkeä tehtävänsä kokonaisuuden kannalta. Tässä luvussa avaan sovelluksessa käytettyjä komponentteja yleisellä tasolla ja pohjustan sitä, mitä tarkoitusta kukin komponentti palvelee sovelluksen toiminnassa. Tarkempi tekninen analyysi verkkosovelluksen toiminnasta löytyy luvusta 4.

3.1 Verkkosovelluksessa käytetyt komponentit

Verkkosovelluksessa käytetyt teknologiat kuvauksineen on esitetty taulukossa 1.

Taulukko 1. Verkkosovelluksessa käytetyt teknologiat ja niiden kuvaus

Teknologia	**Kuvaus**
Red5 Media Server	Mediapalvelin, jossa ajetaan ohjelmistopalvelun koontiversiota.
Java EE	Ohjelmointialusta, joka tarjoaa rajapinnan yritystason ohjelmistojen luontiin.
Spring-sovelluskehys	Sovelluskehys, jonka päälle koko ohjelmistopalvelu on rakennettu.
MySQL	Relaatiotietokantaohjelmisto, johon tallennetaan sovelluksen pysyvä data.
Hibernate	ORM (Object-Relational Mapping) -kehys, jonka avulla sovitetaan Javan oliopohjainen tiedon esitysmuoto relaatiotietokantaan sopivaksi.
HDFVR	High Definition Flash Video Recorder. Kolmannen osapuolen apuohjelmisto, jonka avulla käyttäjän lähetykset (video / audio) voidaan tallentaa palvelimelle.
Bootstrap	Työkalu, jonka avulla voidaan muokata tiedon esitystapaa sen mukaan, millä päätelaitteella sivustoa tarkastellaan.
JSP	Java Server Pages. Teknologia dynaamisten verkkosivujen luontiin.
Three.js	JavaScriptillä toteutettu grafiikkakirjasto, jonka avulla helpotetaan 3D-grafiikan luontia ja käsittelyä.

3.1.1 Red5 Media Server

Red5 Media Server on avoimen lähdekoodin mediapalvelin, joka mahdollistaa mm. eri medioiden suoratoiston ja muita ominaisuuksia kuten käyttäjän lähetysten nauhoittamisen (video / audio). Suoratoistolla tarkoitetaan käytännössä sitä, että esim. ladattaessa video- tai äänitiedostoa verkkopalvelusta voi käyttäjä aloittaa tämän mediatiedoston toistamisen jo ennen kuin se on kokonaan siirretty palvelimelta asiakkaalle. Tällä pyritään vähentämään palvelimeen kohdistuvaa kuormitusta, ja se onnistuukin tilanteissa, joissa mediatiedoston lataaminen keskeytyy syystä tai toisesta. Tällainen tilanne voi tapahtua esim. silloin, kun käyttäjä lopettaa mediatiedoston toiston ennen kuin se on kokonaan latautunut. Suoratoisto onkin erittäin tärkeä ominaisuus nykyajan mediapalvelimissa, koska se auttaa minimoimaan palvelimen ja asiakkaiden välillä tapahtuvaa tiedonsiirtokuormaa. Red5:n ainoa käyttö ei kuitenkaan ole ainoastaan suoratoiston mahdollistaminen, vaan se toimii myös palvelimena itse ohjelmistopalvelulle. Red5 sisältää itsessään Apache Tomcat -palvelimen, joka asentamisen jälkeen tarkastelee asennushakemistonsa sisällä olevaa webapps-hakemistoa ja etsii sieltä WAR (Web application ARchive) -tiedostoja. Mikäli se löytää tästä hakemistosta WAR-tiedoston, se rakentaa tästä tiedostosta tiedostonnimeä vastaavan hakemiston sisältöineen, jota palvelin alkaa sitten tarjoilla. [1; 2.]

Red5 Media Server on valittu verkkosovelluksen palvelimeksi, koska se on avoimen lähdekoodin ilmainen vaihtoehto kaupallisille suoratoistopalvelimille. Vaihtoehtoiset kaupalliset suoratoistopalvelimet kuten Wowza Media Server ja Adobe Flash Media Server ovat hinnaltaan vastaavasti 880 € ja 4000 €. Red5:n ikävämpi puoli on kuitenkin se, että se on heikosti dokumentoitu ja vaatii ainakin perustason ymmärryksen Javasta ja tietoverkoista, jotta sen toimintaa voi ymmärtää. Minkäänlaista tukipalvelua ei myöskään ole lukuun ottamatta postituslistaa, jossa ihmiset voivat keskustella toistensa kanssa vastaan tulleista ongelmista ja etsiä ratkaisua yhteisön avulla. [3; 4.]

3.1.2 Java / Java EE

Java on oliopohjainen ohjelmointikieli, jonka yksi tunnistettavimmista piirteistä on alustariippumattomuus, eli Javalla tuotettua koodia on mahdollista ajaa millä tahansa alustalla, joka tukee Java-virtuaalikonetta (JVM). Java onkin yksi suosituimmista ohjelmointikielistä etenkin asiakas-palvelin-malliin perustuvissa sovelluksissa, kuten verkkosovelluksissa. Javan standardiversio (Java SE) sisältää Java-kielen ydintoiminnallisuuden eli oh-

jelmointirajapinnan, jonka avulla Java-sovelluksia voidaan luoda. Java EE eli Java Enterprise Edition on Java SE:n päälle rakennettu sovellusalusta, joka tarjoaa oman ohjelmointirajapintansa ja ajonaikaisen ympäristön suuren mittakaavan, monikerroksisten, skaalautuvien, luotettavien ja turvallisten verkkosovellusten luontiin ja suorittamiseen. [5; 6.]

Java on valittu verkkosovelluksen ohjelmointikieleksi, koska se erittäin yleinen ohjelmointikieli ja täten hyvin tuettu. Nämä kaksi seikkaa lisäävät todennäköisyyttä sille, että sovellukseen liitettävät teknologiat ovat helposti integroitavissa itse sovellukseen. Koska Javaa voidaan ajaa useissa eri ympäristöissä, mahdollistaa se myös sen, että sovelluksen alla oleva käyttöjärjestelmä voidaan tarvittaessa vaihtaa toiseen.

3.1.3 Spring-sovelluskehys

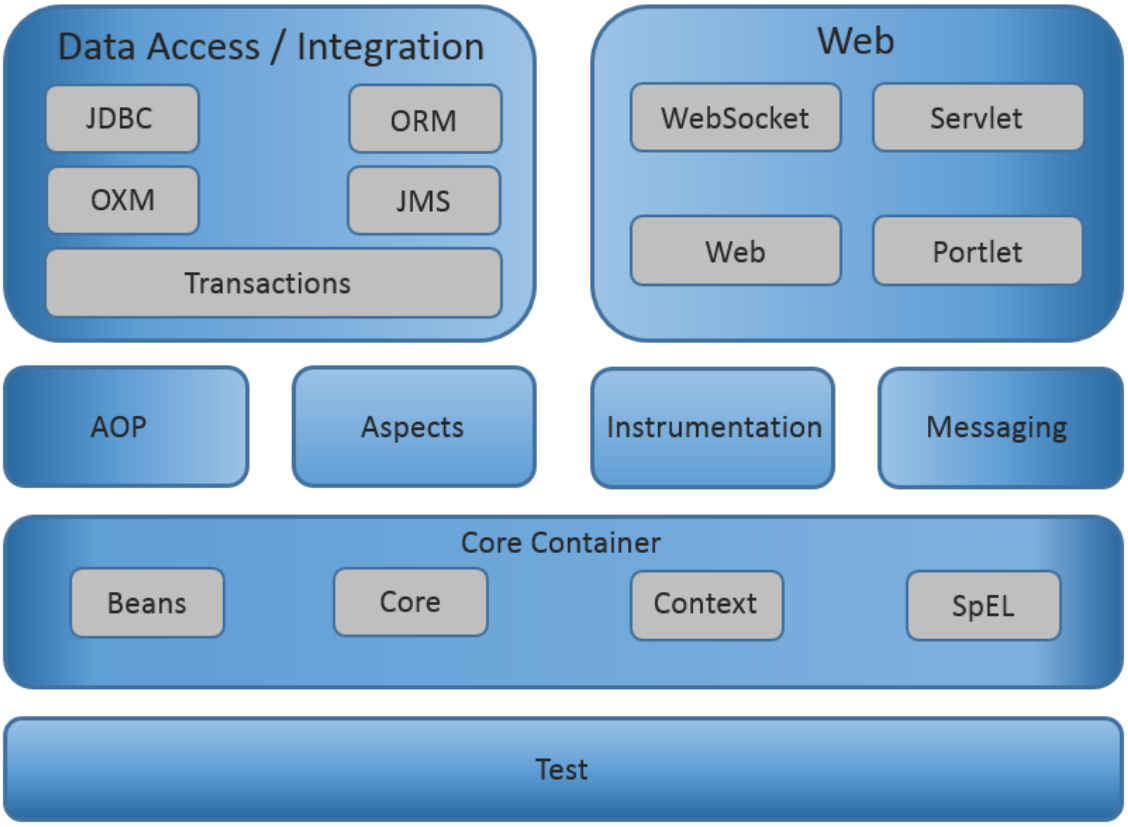
Ohjelmistopalvelu on toteutettu Spring-sovelluskehyksellä, joka on työn kirjoitushetkellä suosituin yritystason sovelluskehys Java-pohjaisille sovelluksille. Tietokoneasiantuntija Rod Johnson loi sovelluskehyksen ensimmäisen version vuonna 2002 helpottaakseen sovelluskehittäjien työtä tarjoamalle heille valmiin aihion eli sovelluskehyksen sovellusten luontiin. Spring kannustaa ja opastaa käyttämään ohjelmistoteollisuudessa hyväksi havaittuja käytäntöjä sovellusten luonnissa. Näitä ovat mm. modulaarisuus ja IoC (Inversion of Control) eli kontrollin kääntäminen sovelluskehykselle. [7.]

Kontrollin kääntäminen sovelluskehykselle on toteutettu käyttämällä riippuvuusinjektiota, jolla tarkoitetaan sitä, että Java-luokat eivät itsessään saa luoda toisia olioita vaan näiden toisiinsa kytkeminen annetaan sovelluskehyksen hoidettavaksi. Kun ohjelmistokoodin koko ja luokkien määrä kasvaa mittavaksi, on tärkeää pitää luokat irrallaan toisistaan, koska pieni muutos toisessa luokassa voi aiheuttaa sen, että koodimuutoksia joudutaan tekemään satoihin muihin luokkiin. Käyttämällä riippuvuusinjektiota ohjelmakoodista saadaan modulaarisempi ja esim. koodin testaus helpottuu, koska luokkia voidaan yksikkötestata itsenäisesti. [7; 8.]

Spring-sovelluskehyksen keskiössä on sen IoC-säiliö, joka antaa sovelluskehykselle kyvyn konfiguroida ja hallinnoida Java-olioita. IoC-säiliö on vastuussa olioiden koko elinkaaresta sekä näiden kytkemisestä toisiinsa. Säiliö voidaan konfiguroida joko tarjoamalla sille ulkoista XML (Extensible Markup Language) -muotoista dataa tai määrittelemällä

sovellusluokkiin Java-annotaatioita, joiden avulla sovelluskehykselle välitetään tieto siitä, kuinka nämä oliot tulee luoda ja kytkeä toisiinsa.

Spring koostuu valmiista moduuleista, joita ohjelmistonkehittäjä voi ottaa käyttöön kehittämänsä sovelluksen vaatimusten mukaisesti. Ohjelmistokehyksen moduulit on esitetty kuvassa 1.



Kuva 1. Spring-sovelluskehyksen moduulit

Verkkosovelluksessa on otettu käyttöön moduulit JDBC (Java Database Connectivity), ORM, Servlet, Web, AOP, Beans, Core ja Context. Käyttöön otetut moduulit ja näiden tarkoitus on koottu taulukkoon 2.

Taulukko 2.

Moduuli	**Tarkoitus**
JDBC	Moduuli jonka avulla verkkosovellus muodostaa yhteyden tietokantaan.
ORM	Tämä moduuli on otettu käyttöön olio-relaatio muunnosta varten. Tätä muunnosta tarvitaan, jotta verkkosovelluksen oliopohjainen tiedon esitysmuoto voidaan muuntaa relaatiotietokantaan sopivaksi.
Servlet	Otettu käyttöön, koska kyseessä on verkkosovellus. Tämä moduuli sisältää MVC-kehyksen verkkosovelluksien laatimista varten.
Web	Tarjoaa toiminnallisuuden, jossa tiedostoja voidaan ladata palvelimelle. Määrittää myös alkuasetukset Springin IoC-säiliölle käyttäen servlet-kuuntelijoita ja verkko-orientoitunutta sovelluskontekstia (web-oriented application context).
AOP	Aspect Oriented Programming. Tarjoaa mahdollisuuden aspektipohjaiseen ohjelmistonkehitykseen. Tällä tarkoitetaan sitä, että sovelluksen toiminnallinen logiikka ja muut siihen liittymättömät sovelluksen osat pidetään irrallaan toisistaan. Yksi esimerkki tästä on mm. se, että sovellusluokkiin ei tarvitse kirjoittaa erillisiä lokilausekkeita lokin muodostamiseksi, vaan tämä hoidetaan määrittelemällä ulkoinen konfiguraatiotiedosto, jossa määritellään, miten loki muodostuu. Näin toimimalla koodi pysyy puhtaampana ja helpommin hallittavampana.
Core ja Beans	Tarjoavat Spring-sovelluskehyksen ydintoiminnallisuuden mukaan lukien kontrollin kääntämisen sovelluskehykselle ja riippuvuusinjektion.
Context	Tämän moduulin avulla tarjoutuu mahdollisuus päästä käsiksi luotuihin olioihin sovelluskehyksen opastamalla tavalla. Se tarjoaa tuen myös sovelluksen lokalisaatioon käyttäen esim. ulkoista resurssitiedostoa jonne maakohtaiset tekstipohjaiset merkkijonot voidaan määritellä.

3.1.4 MySQL

Jotta ohjelmistopalveluun rekisteröityneiden käyttäjien tietoja, tallennettuja tarinoita ja muuta ohjelmistopalvelun toiminnan kannalta tärkeää dataa voidaan käsitellä ja varastoida, tarvitaan tietokannan hallintajärjestelmä. Etenkin järjestelmissä, joissa käsiteltävä datamäärä kasvaa suureksi, tulee tallennetun tiedon olla organisoitu, jotta sitä on mahdollista käsitellä järkevällä tavalla. MySQL on avoimen lähdekoodin tietokannan hallintajärjestelmä, jonka avulla tietoa voidaan tallentaa, hakea ja muuttaa tehokkaasti. MySQL-tietokannat perustuvat ns. tauluihin ja näiden välisiin yhteyksiin, joihin määritellään luontivaiheen aikana mm., minkälaista tietoa voidaan syöttää tietokantaan, miten tietoa voidaan päivittää ja miten sitä voidaan poistaa.

3.1.5 Hibernate

Olio-ohjelmointikielissä data esitetään toisiinsa kytkeytyneinä olioina, kun taas relaatiotietokannoissa se esitetään taulukkomuodossa. Hibernaten avulla helpotetaan ohjelmoijan työtä tarjoamalla korkean tason sovelluskehys, jonka avulla voidaan käyttää useita eri tietokantoja samalla ohjelmistokoodilla muokkaamalla pelkästään konfiguraatiotiedostoa. Hibernatea käytetäänkin verkkosovelluksessa eräänlaisena sovittimena sovelluskoodin ja käytössä olevan MySQL-tietokannan välissä. Käyttämällä kymmenientuhansien Java-koodaajien hyväksi todettua ratkaisua voi ohjelmoija keskittyä ohjelmansa koodauksessa olennaisimpiin asioihin ja jättää merkittävän osa koodista sovelluskehyksen hoidettavaksi. [9; 10.]

3.1.6 HDFVR

HDFVR on kolmannen osapuolen apuohjelmisto, jonka avulla voidaan nauhoittaa käyttäjän videokuvaa tai ääntä. W3C (World Wide Web Consortium) ylläpitää ja kehittää WWW:n (World Wide Web) standardeja ja suosituksia. Se on tehnyt kaksi virallista aloitetta, jonka tavoitteena on pyrkiä saamaan videolähetyksien nauhoitus suoraan verkkoselaimiin käyttäen HTML5 (HyperText Markup Language) -merkintäkieltä. Ensimmäinen näistä aloitteista eli HTML Media Capture on kandidaattivaiheessa, mikä tarkoittaa sitä, että standardin määrittelyssä merkittävimmät ominaisuudet on jo päätetty, mutta sitä testataan vielä kehittäjäyhteisön toimesta. Saadun palautteen perusteella joidenkin ominaisuuksien toteutus voi vielä vaihdella. Toinen näistä aloitteista eli MediaStream Recording on luonnosvaiheessa, joten videolähetysten nauhoittamiseen tarvitaan tänä päivänä vielä erillinen kolmannen osapuolen apuohjelmisto. Eräs syy, miksi HDFVR on kehittäjien suosiossa, liittyy siihen, että sen mukana toimitetaan myös palvelinpuolen sovellusrajapinta tapahtumien käsittelyyn. Rajapinnan avulla käyttäjän lähetykset voidaan vaivattomasti tallentaa tietokantaan. Toinen syy on se, että lähetykset saadaan tallennettua MP4-muotoon, joka on todella suosittu videoformaatti ja toimii lähes kaikilla verkkoselaimilla (ainoana poikkeuksena Opera Mini). [11; 12.]

3.1.7 Bootstrap

Bootstrap on avoimen lähdekoodin kokoelma työkaluja ja aihioita, jonka tarkoitus on helpottaa responsiivisten verkkosivustojen luontia. Responsiivisella verkkosivustolla tarkoitetaan sivustoa, joka tarjoilee näkymänsä eri tavalla riippuen siitä, minkälaisella päätelaitteella sivustoa tarkastellaan. Bootstrap koostuu pääasiassa esimääritellyistä HTML-

sivupohjista, CSS (Cascading Style Sheets) -tyylitiedostoista ja JavaScript-laajennuksista. [13.]

3.1.8 JSP

JSP (JavaServer Pages) on teknologia, jonka avulla voidaan luoda dynaamista sisältöä verkkosivuille. Sen julkaisi Sun Microsystems vuonna 1999, ja se käyttää alustanaan Java-ohjelmointikieltä. Jotta JSP-sivuja voidaan käyttää verkkosovelluksissa, tarvitaan verkkopalvelin, joka sisältää servlet-säiliön (esim. Red5). JSP-sivuja voidaan käyttää joko itsenäisesti tai MVC-mallin View-komponenttina. [14.]

JSP onkin otettu käyttöön verkkosovelluksessa juuri edellä mainitusta syystä. Se toimii informaation esityskerroksena (View) Spring-sovelluskehyksen Web MVC -mallissa. MVC-mallista ja sovelluksen teknisestä toteutuksesta löytyy lisää informaatiota luvussa 5. [14.]

3.1.9 Three.js

Three.js on JavaScriptillä toteutettu grafiikkakirjasto verkkoselaimessa suoritettavien 3D-animoitujen sovellusten luontiin. Kirjasto käyttää WebGL-rajapintaa grafiikan renderointiin ja mahdollistaa näin grafiikkaprosessorin käyttämisen sovellusten suorittamisessa ilman ylimääräisiä verkkoselainten lisäosien asennuksia. Puhtaan WebGL:n koodaaminen on tavattoman työlästä verrattuna Three.js-apukirjaston käyttöön. Useiden satojen rivien mittainen WebGL-koodi yksinkertaisen kuution renderoimiseksi pelkistyy murto-osaan tästä käyttämällä tätä apukirjastoa. [15; 16.]

Three.js-grafiikkakirjastoa on käytetty verkkosovelluksen käyttöliittymän toteutuksessa. Grafiikkakirjaston avulla sovellukseen on luotu 3D-ympäristö, jota voidaan navigoida aika-akselilla vierittämällä hiiren rullaa tai liikuttamalla ruudulla näkyvää liukusäädintä. Tarkoitus on ollut luoda intuitiivinen tapa selata eri vuosikymmenille ajoittuvia muistoja. Käyttäjä kulkeekin palvelussa vuosikymmenten ohi nähden kullekin vuosikymmenelle ajoittuvat muistot.

4 Verkkosovelluksen ohjelmistoteknologiat ja toiminta

Tässä luvussa avaan verkkosovelluksen teknistä toiminnallisuutta aloittaen siitä, miten palvelin luo vastauksen asiakkaan lähettämiin pyyntöihin. Tämän jälkeen esittelen, miten palvelin ja verkkosovellus ovat konfiguroitu käyttäen ulkoisia konfiguraatiotiedostoja. Sovelluksen käyttämää tietokantaa, rakennetta ja toiminnallisuutta käydään läpi luvussa 4.4, jota seuraa verkkosovelluksen yleisarkkitehtuurin esittely.

4.1 Spring Web MVC

Verkkosovelluksen ydintoiminnallisuus perustuu Spring-sovelluskehyksen servlet-moduulissa sijaitsevaan MVC-mallin toteutukseen, joka on suunniteltu verkkosovellusten luontiin. Springin MVC-mallin toteutusta kutsutaan Spring Web MVC -sovelluskehykseksi. Web MVC -sovelluskehys pohjautuu Javan servlet-teknologian ympärille, jonka avulla voidaan laajentaa verkkopalvelimen toiminnallisuutta tarjoamalla sovelluskehittäjälle mahdollisuus luoda asiakkaan lähettämään pyyntöön vastaus käyttämällä apuna Java-ohjelmointikielen tarjoamia työkaluja.

MVC on ohjelmistotekniikassa yleisesti käytössä oleva suunnittelumalli, joka jakaa sovelluksen toiminnallisuuden kolmeen kerrokseen.

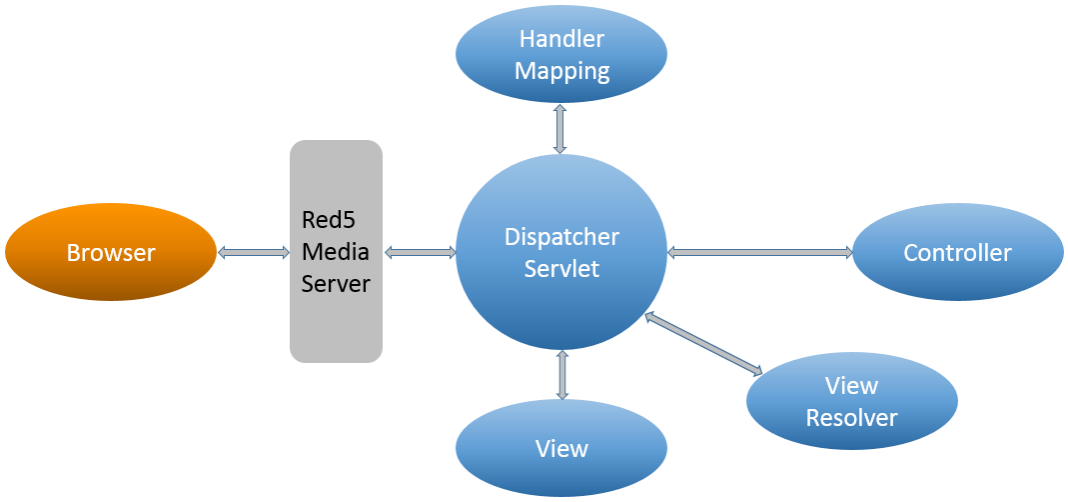
- Malli / Model – sisältää sovellusdatan. Yleensä koostuu POJO (Plain Old Java Object) -luokista.
- Näkymä / View – sisältää esityskerroksen eli tavan, jolla informaatio esitetään. Tässä tapauksessa JSP-sivu, jonne malli eli sovellusdata välitetään.
- Ohjain / Controller – vastuussa pyyntöjen käsittelystä, mallin rakentamisesta ja sen viemisestä oikeaan näkymään.

MVC-mallin ideana on se, että jakamalla ohjelman koodi kolmeen kerrokseen voidaan kutakin kerrosta kehittää itsenäisesti, eikä esim. käyttöliittymäsuunnittelijan kehitystyö häiritse henkilöä, joka on keskittynyt sovelluksen toiminnallisuuden kehittämiseen. MVC-malli täten lisää kehitettävän ohjelmiston modulaarisuutta ja vähentää ohjelmistonkehitykseen ja testaukseen tarvittavaa aikaa.

Springin Web MVC -sovelluskehys perustuu front controller -suunnittelumalliin, jolla tarkoitetaan sitä, että pyyntöjen käsittely on keskitetty yhden keskeisen ohjelmistokom-

ponentin ympärille. Spring MVC -kehyksessä tämä komponentti on nimeltään dispatcher-servlet, ja se sääteleekin koko verkkosovelluksen kontrollivirtaa. Tämän vuoksi se on erittäin olennaisessa roolissa sovelluksen sisäisen logiikan ymmärtämisessä. [17; 18.]

Dispatcher-servletillä onkin ensisijainen vastuu pyynnön käsittelyssä ja vastauksen tuottamisessa. Se ei kuitenkaan yksinään pysty tuottamaan vastausta, vaan se konsultoi käytössä olevia apureitaan, jotka on esitetty kuvassa 2.



Kuva 2. Spring Web MVC -sovelluskehyksen perustoiminnallisuus

Kun loppukäyttäjä eli asiakas syöttää internetselaimen osoiteriville URL:n (Uniform Resource Locator), joka kohdistuu yrityksen verkkotunnukseen, pyynnön vastaanottaa Red5-mediapalvelin, joka tarkkailee palvelimella aukaistuja portteja. Mikäli pyyntö kohdistuu porttiin 80 (HTTP, HyperText Transfer Protocol), välitetään se eteenpäin Spring-sovelluskehyksen dispatcher-servletille.

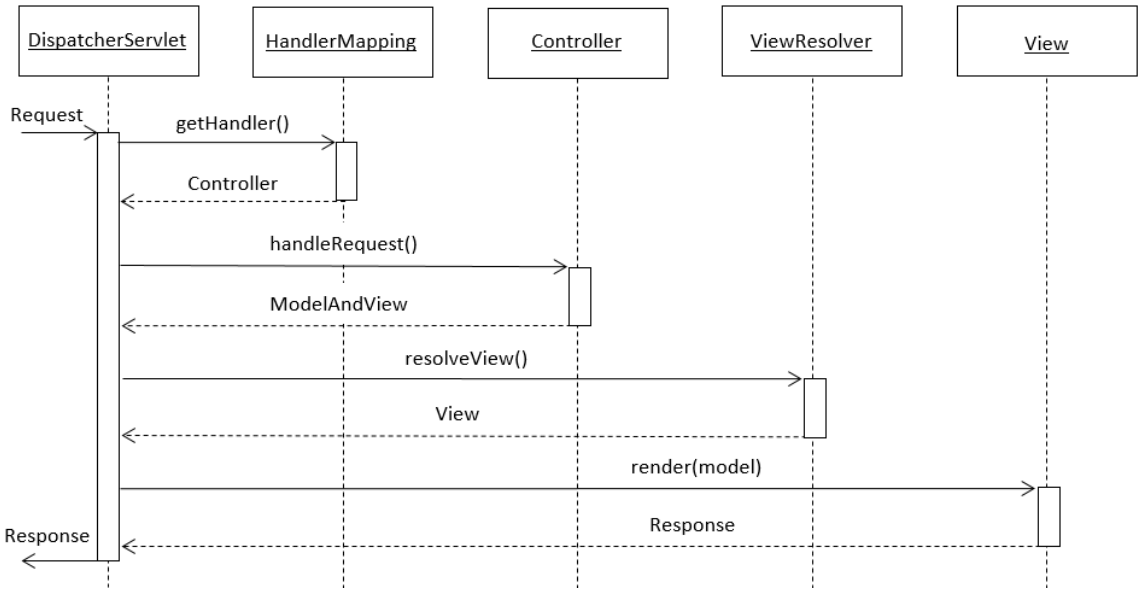
Kun dispatcher-servlet vastaanottaa asiakkaan lähettämän pyynnön, se ei vielä tiedä, mikä ohjain on vastuussa tämän pyynnön käsittelystä, joten se konsultoi handler mapping -komponenttia. Handler mappingin tehtävä onkin skannata pyynnön mukana vastaanotettu URL-osoite ja selvittää, mikä ohjain on vastuussa tämän pyynnön käsittelystä. Kun handler mapping on löytänyt oikean ohjaimen, se lähettää tiedon tästä ohjaimesta takaisin dispatcher-servletille. [18.]

Kun dispatcher-servlet on vastaanottanut tiedon oikeasta ohjaimesta, se välittää pyynnön eteenpäin tälle ohjaimelle. Ohjaimien tehtävänä on suorittaa verkkosovelluksen toiminnallisuuden mukaiset toimenpiteet ja tallentaa saatu data malli-olioon. Ohjain palauttaa lopuksi mallin ja tiedon esittämisestä vastuussa olevan näkymän nimen dispatcher-servletille. [18.]

Kun dispatcher-servlet on vastaanottanut mallin ja näkymän nimen, tulee saatu malli lähettää tälle näkymälle. Dispatcher-servlet ei kuitenkaan tiedä, missä tämä näkymä sijaitsee, joten se konsultoi view resolver -nimistä komponenttia. View resolverin tehtävänä on selvittää, missä oikea näkymä sijaitsee. View Resolver vastaanottaa näkymän nimen, selvittää sen sijainnin ja palauttaa sijainnin dispatcher-servletille. [18.]

Kun dispatcher-servlet on vastaanottanut näkymän sijainnin, se välittää mallin tälle näkymälle. Näkymä koostuu sivupohjasta, johon lisätään dynaamisesti sisältöä mallissa olevan datan avulla. Näkymä rakentaa sivupohjasta ja datasta valmiin vastaussivun ja palauttaa sen dispatcher-servletille. [18.]

Kuva 3 havainnollistaa toimintojen ajallista järjestystä siitä eteenpäin, kun asiakkaan pyyntö on välitetty Red5-mediapalvelimelta dispatcher-servletille.



Kuva 3. Sekvenssikaavio kontrollin etenemisestä sovelluksessa

Kun dispatcher-servlet on vastaanottanut lopullisen vastaussivun, se välittää tämän sivun asiakkaalle, ja asiakkaan pyyntöön on näin luotu vastaus. [18.]

4.2 Verkkosovelluksen konfiguraatiotiedostot

Jotta Red5-mediapalvelin ja Spring-sovelluskehys tietävät, kuinka verkkosovellus otetaan käyttöön palvelimella, tulee näille tarjota tämä informaatio ulkoisissa konfiguraatiotiedostoissa. Nämä seuraavissa luvuissa esiteltävät kaksi tärkeää tiedostoa sijaitsevat verkkosovelluksen juurihakemiston /WEB-INF-hakemistossa.

4.2.1 web.xml

Kaikissa servlet-teknologiaan pohjautuvissa sovelluksissa tarvitaan web deployment descriptor -tiedosto (web.xml), joka määrittelee, miten verkkosovellus otetaan käyttöön palvelimella. Kuvassa 4 on esitetty verkkosovelluksen web.xml-tiedoston konfiguraatio, jonka sisältöä on tarkemmin avattu kuvan alapuolella olevassa taulukossa 3.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://java.sun.com/xml/ns/javaee"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
 http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <display-name>Story Management System</display-name>

   <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class> org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.html</url-pattern>
    </servlet-mapping>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
    </context-param>

    <listener>
        <listener-class>
                    org.springframework.web.context.ContextLoaderListener
                </listener-class>
    </listener>

    <filter>
        <filter-name>EncodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter
        </filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <init-param>
            <param-name>forceEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>EncodingFilter</filter-name>
        <url-pattern>*</url-pattern>
    </filter-mapping>

</web-app>
```

Kuva 4. web.xml-konfiguraatiotiedosto

Taulukko 3. web.xml-konfiguraatiotiedoston asetukset ja niiden merkitys

Rivi(t)	**Merkitys**
1 - 5	Sisältää määrittelyt siitä, millainen rakenne tällä dokumentilla on.
7	Määrittää nimen, joka näkyy kehittäjälle ohjelmiston kehitysvaiheessa.
9 - 11	Määrittelee, mikä sivu näytetään, jos käyttäjä syöttää URL:n, jolle ei ole määritelty erillistä ohjainta (controller).
14	Määrittelee, minkä niminen servlet luodaan front controlleriksi (dispatcher-servlet.xml).
15	Määrittää, mitä luokkaa käytetään front controllerin toteuttavana luokkana.
17	Asetus, jonka mukaan servlet tulee ladata siinä vaiheessa, kun palvelin käynnistetään / projekti välitetään palvelimelle (muutoin servlet ladattaisiin vasta siinä vaiheessa, kun ensimmäinen pyyntö saapuu palvelimelle).
21	Määrittelee minkä nimiselle servletille rivillä 21 määritellyt URL-pyynnöt ohjataan (dispatcher-servlet.xml).
22	Määrittelee tarkemmin, mitkä URL-pyynnöt ohjataan rivillä 20 määritellylle servletille. Tässä tapauksessa kaikki .html-päätteiset pyynnöt ohjataan dispatcher-servletille.
26	Määritys, jonka avulla kerrotaan palvelimelle, että käytössä on myös muita ulkoisia konfiguraatiotiedostoja (dispatcher-servlet.xml).
27	Määritellään tämän toisen ulkoisen konfiguraatiotiedoston sijainti.
32	Konfiguroi sovelluskontekstin (web-oriented application context).
37	Määrittelee, minkä niminen luokka luodaan käyttäjän lähettämien pyyntöjen suodattamiseksi.
38	Määrittelee, mitä luokkaa käytetään tämän suodattimen toteuttavana luokkana.
41 - 42	Määrittelee merkistön koodaus-asetukset, jotka määrittelevät miten käyttäjältä tulevat pyynnöt koodataan (käytetään UTF-8 merkistöä).
45 - 46	Asetetaan EncodingFilter-suodatus päälle.
50 - 51	Määrittelee, että kaikki käyttäjältä tulevat pyynnöt lähetetään EncodingFilter-suodattimelle ennen pyynnön käsittelyä.

4.2.2 dispatcher-servlet.xml

Tieto siitä, miten Spring Web MVC -sovelluskehys on konfiguroitu, sijaitsee dispatcher-servlet.xml-tiedostossa, jonne on määritelty verkkosovelluksessa käyttöön otettavat

komponentit. Verkkosovellus käyttää ns. annotaatiopohjaista konfiguraatiota, jolloin sovellukseen koodattuihin luokkiin lisätään tiettyjä avainsanoja eli annotaatioita. Annotaatiopohjaisessa konfiguraatiossa Spring skannaa verkkosovelluksen luokat läpi ja konfiguroi itsensä luokkiin määriteltyjen annotaatioiden avulla. Dispatcher-servletin konfiguraatiotiedot on koottu kuviin 5 ja 6, joista tarkemmat tiedot ovat taulukoissa 4 ja 5.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd ">

    <context:component-scan base-package="com.tatva.story" />

    <import resource="spring-aop.xml"/>

    <mvc:annotation-driven />

    <mvc:interceptors>
        <bean id="localeChangeInterceptor"
            class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
            <property name="paramName" value="lang" />
        </bean>
    </mvc:interceptors>

    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/story/**" />
            <mvc:mapping path="/admin/**" />
            <mvc:mapping path="/videoInformation*" />
            <bean class="com.tatva.story.interceptor.LoginInterceptor" />
        </mvc:interceptor>
    </mvc:interceptors>

    <mvc:resources mapping="/hdfvr/**" location="/hdfvr" />

    <bean id="sessionFactory"
    class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
      <property name="dataSource" ref="dataSource" />
      <property name="packagesToScan" value="com.tatva.story.entity" />
      <property name="hibernateProperties">
         <props>
            <prop key="hibernate.hbm2ddl.auto">validate</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
         </props>
      </property>
   </bean>
```

Kuva 5. Dispatcher-servletin konfiguraatio osa 1.

Taulukko 4. Dispatcher-servletissä määritetyt asetukset ja niiden merkitys (rivit 1 – 49)

Rivi(t)	**Merkitys**
1 - 12	Sisältää määrittelyt siitä, millaista dataa tämä dokumentti pitää sisällään.
14	Kerrotaan Spring-sovelluskehykselle, että etsitään annotaatioita luokista, jotka sijaitsevat tähän määritellyn pakkauksen sisällä (com.tatva.story).
16	Otetaan mukaan resurssi, jossa on määritelty tarkemmin AOP:n asetuksia.
18	Kerrotaan Spring-sovelluskehykselle, että käytetään annotaatiopohjaista konfiguraatiota.
20 - 25	Määritellään interceptor eli sieppaaja, jonka avulla voidaan siepata tietyt pyynnöt palvelimelle. Tässä on määritelty kielen valintaan liittyvä sieppaaja. Mikäli kieltä on vaihdettu, vaihdetaan käyttöliittymän teksti toisen kieliseksi, ennen kuin sivu esitetään.
27 - 34	Määritellään sieppaaja, jonka tarkoituksena on tarkistaa, onko pyynnön lähettäneellä käyttäjällä oikeuksia päästä tiettyihin resursseihin käsiksi.
36	Otetaan käyttöön kolmannen osapuolen apuohjelmisto mediatiedostojen nauhoittamista varten (HDFVR).
38 - 49	Määritellään tietokantaoperaatioihin liittyvät asetukset.

```
51    <bean id="dataSource"
52    class="org.springframework.jdbc.datasource.DriverManagerDataSource">
53      <property name="driverClassName" value="com.mysql.jdbc.Driver" />
54      <property name="url" value="jdbc:mysql://localhost:3306/*****" />
55      <property name="username" value="*****" />
56      <property name="password" value="*****" />
57   </bean>
58
59   <bean id="transactionManager"
60    class="org.springframework.orm.hibernate4.HibernateTransactionManager">
61      <property name="sessionFactory" ref="sessionFactory" />
62   </bean>
63
64    <bean id="multipartResolver"
65        class="org.springframework.web.multipart.commons.CommonsMultipartResolver" />
66
67    <bean id="viewResolver"
68        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
69        <property name="prefix" value="/WEB-INF/pages/" />
70        <property name="suffix" value=".jsp" />
71    </bean>
72
73     <bean id="messageSource"
74     class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
75          <property name="basename" value="classpath:messages" />
76    </bean>
77
78    <bean id="localeResolver"
79    class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
80        <property name="defaultLocale" value="fi"/>
81        <property name="cookieName" value="Cookie"></property>
82        <property name="cookieMaxAge" value="3600"></property>
83    </bean>
84
85    <bean id="flashScopeInterceptor"
86    class="com.tatva.story.interceptor.FlashScopeInterceptor" />
87
88    <bean id="handlerMapping"
89    class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping">
90        <property name="interceptors">
91            <list>
92                <ref bean="flashScopeInterceptor" />
93            </list>
94        </property>
95    </bean>
96
97 </beans>
```

Kuva 6. Dispatcher-servletin konfiguraatio osa 2.

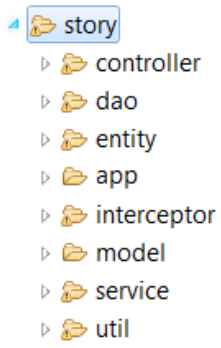
Taulukko 5. Dispatcher-servletissä määritetyt asetukset ja niiden merkitys (rivit 51 – 97)

Rivi(t)	**Merkitys**
51 - 57	Määritellään komponentti, jonka avulla voidaan ottaa yhteys tietokantaan. Property-attribuuttien avulla määritellään, minkälaista tietokanta-ajuria käytetään, mitä porttia ja mitä tietokantaa käytetään, mikä on käyttäjätunnus ja salasana tähän tietokantaan.
59 - 62	Määrittää transaktioiden hallinnoijan, jonka tarkoitus on pitää huolta siitä, että tietokantaoperaatiot sujuvat asianmukaisesti. Tällä var-

	mistutaan siitä, että vain yksi käyttäjä pääsee kerrallaan muokkaamaan tiettyä tietuetta ja kaikki muutokset joko toteutetaan tai perutaan riippuen siitä, onnistuiko transaktio.
64 - 65	Määritellään komponentti, jonka avulla mahdollistetaan tiedostojen siirto palvelimelle. Toteuttaa Jakarta Commons FileUpload 1.2 -rajapinnan.
67 - 71	Määritellään MVC-mallin näkymänselvittelijä sovellukselle. Tämän avulla tarkennetaan missä hakemistossa näkymät sijaitsevat ja mikä on näkymien tiedostotyyppi (.jsp).
73 - 76	Määritellään resurssien sijainti koskien lokalisaatio-asetuksia. Kirjoitushetkellä sovellus sisältää lokalisaatio-asetukset kahdelle kielelle (Suomi ja Englanti)
78 - 83	Määrittää evästeen, joka lähetetään käyttäjälle, mikäli vaihdetaan kieltä. Käyttäjälle lähetettyä evästettä käytetään automaattisesti kielen vaihdon jälkeen, jolloin kieltä ei tarvitse asettaa jokaisella pyynnöllä palvelimelle.
85 - 86	Määritellään sieppaaja, jonka tarkoituksena on joko sitoa flashScope nykyiseen sessioon tai poistaa se nykyisestä sessiosta. Jos flashScope on jo sidottu, lomitetaan se olemassa olevan flashScopen kanssa.
88 - 95	Määritellään käsittelijä, joka luo luokissa olevien annotaatioiden pohjalta mappaukset sille, mitkä ohjaimet ovat vastuussa kustakin URL-pyynnöstä.

4.3 Verkkosovelluksen luokkarakenne

Kirjoitushetkellä verkkosovellus sisältää 68 erilaista luokkaa, jotka on jaoteltu toimintojen mukaisesti eri hakemistoihin (kuva 7). Tässä luvussa esittelen, mitä kukin hakemisto pitää sisällään ja mikä niiden tarkoitus on. Esittelen, miten tarinat haetaan tietokannasta käyttäen hakemistoihin määriteltyjä luokkia. Luokkien esittely tapahtuu samassa järjestyksessä, miten kontrolli sovelluksessa etenee.



Kuva 7. Verkkosovelluksen luokkarakenne

4.3.1 Controller

Controller-hakemisto pitää sisällään sovelluksessa käytetyt ohjainluokat, joille kontrolli käännetään dispatcher-servletin toimesta. Kullekin verkkotunnukseemme kohdistuvalle URL-pyynnölle on määritelty ohjain, joka on vastuussa kyseessä olevan URL-pyynnön käsittelystä. Kuvassa 8 on esitelty ohjain, joka on vastuussa /story/list-pyynnön käsittelystä.

```
379     // List of stories
380     @RequestMapping(value = "/story/list", method = RequestMethod.POST)
381     public ResponseEntity<String> getAllStories()
382     {
383         String json="false";
384         List<Story> listStories=storyService.getAllStories();
385         if(listStories!=null && listStories.size()>0){
386             ObjectMapper mapper = new ObjectMapper();
387             try {
388                 json=mapper.writeValueAsString(listStories);
389             } catch (Exception e) {
390                 e.printStackTrace();
391             }
392         }
393 
394         HttpHeaders responseHeaders = new HttpHeaders();
395         responseHeaders.add("Content-Type", "text/html; charset=utf-8");
396         return new ResponseEntity<String>(json, responseHeaders, HttpStatus.OK);
397     }
```

Kuva 8. StoryController-ohjainluokan metodi, joka vastaa /story/list URL-pyynnön käsittelystä

Springin MVC-sovelluskehyksessä määritellään, että mikäli se löytää sovellusluokista annotaation nimeltä @Controller, se käsittelee tätä luokkaa ohjaimena. @RequestMapping-annotaation (rivi 380) avulla määritellään tarkemmin, minkälaista URL-pyyntöä tämä ohjain käsittelee (/story/list). ResponseEntity-metodin tarkoituksena (rivit 381 – 397) on hakea kaikki julkiset tarinat tietokannasta ja palauttaa vastaus JSON (JavaScript

Object Notation) -formaatissa. Vastaukset palautetaan tässä muodossa, koska sovelluksen 3D-näkymän renderöivä clouds.js-tiedosto kutsuu tätä metodia AJAX (Asynchronous JavaScript And XML) -teknologiaa käyttäen ja haluaa vastauksen JSON-muotoisena. Rivillä 384 käytetään service-hakemistossa määriteltyä StoryService-luokkaa tarinoiden hakemiseen tietokannasta. Palautettujen tarinoiden tyyppi on Story (malli, joka on määritelty model-hakemistossa). StoryService-luokka on injektoitu automaattisesti StoryController-luokkaan samassa luokassa määritellyn @Autowired-annotaation avulla. Jos haettujen tarinoiden lista ei ole tyhjä ja haettuja tarinoita on enemmän kuin nolla (rivi 385), haetaan tarinat JSON-formaattiin (rivi 388) rivillä 386 määritellyn ObjectMapper-olion avulla. Jos tässä ei onnistuta, tulostetaan virheloki (rivit 389 – 391). Rivillä 394 – 395 kirjoitetaan vastauksen otsikkokenttään (responseHeaders) tieto siitä, millaista dataa ollaan palauttamassa. Sisällön tyypiksi on asetettu text/html ja merkistönä käytetään UTF-8 -merkistöä. Lopuksi palautetaan uusi HttpEntity, jonka vartalossa (body) on koottu JSON-merkkijono, otsikkotietona responseHeaders ja HTTP:n tilakoodina OK.

Taulukkoon 6 on koottu ohjainluokissa käytetyt annotaatiot ja niiden merkitys.

Taulukko 6. Controller-luokissa käytetyt annotaatiot ja niiden merkitys

Annotaatio	**Merkitys**
@Controller	Merkitsee luokan ohjaimeksi. Kun Spring-sovelluskehys skannaa sovellukseen koodatut luokat läpi etsien annotaatioita, se kirjaa tällä annotaatiolla merkityt luokat MVC-mallin ohjaimiksi.
@RequestMapping	Tätä annotaatiota käytetään ohjainluokassa oleville metodeille. Sen avulla määritellään minkälaiseen URL-pyyntöön tällä annotaatiolla merkitty metodi vastaa (ks. esimerkki kuvasta 9).
@RequestParam	Tämän annotaation avulla määritellään, mitä arvoja (parameters) halutaan lukea lähetetyltä lomakkeelta.
@SessionAttributes	Tämän annotaation avulla määritellään olio(t), johon tallennetaan kuhunkin olioon liittyvä istuntokohtainen data.

4.3.2 Model

Model-hakemisto sisältää verkkosovelluksessa käytetyn MVC-kehyksen malli-oliot (model). Tässä hakemistossa sijaitsevat luokat ovat POJO (Plain Old Java Object) -luokkia, jotka on kapseloitu. Ohjaimet käyttävät näiden luokkien ilmentymiä datan tallentamiseen.

Kuvassa 9 on esitelty Story-luokan konstruktori, jonka avulla tarinoiden data tallennetaan luokan sisäiseen tietorakenteeseen.

```
73    public Story(long id,
74                 String title,
75                 String city,
76                 String country,
77                 Date storyDate,
78                 String publishing,
79                 char ready,
80                 Date created,
81                 String lattitude,
82                 String longitude,
83                 User user)
84    {
85        this.id = id;
86        this.title = title;
87        this.city = city;
88        this.country = country;
89        this.storyDate = storyDate;
90        this.publishing = publishing;
91        this.ready = ready;
92        this.created = created;
93        this.lattitude=lattitude;
94        this.longitude=longitude;
95        this.user = user;
96    }
```

Kuva 9. Story-luokka (model), johon tallennetaan haettavien tarinoiden sisältämä data

4.3.3 Service

Service-luokat määrittelevät, minkälaisia palveluita sovellukseen on koodattu. Aiemmassa luvussa (4.3.1) esitelty ohjainluokka käyttää StoryService-luokkaa (kuva 10) tarinoiden hakemiseen tietokannasta.

```
882    public List<Story> getAllStories(){
883        List<Story> listStory = null;
884        try {
885            List<com.tatva.story.entity.Story> listStoryEntity = storyDao.getAllStories();
886            listStory = new ArrayList<Story>();
887            for(com.tatva.story.entity.Story story: listStoryEntity)
888            {
889                listStory.add(ModelEntityMappingUtils.getStoryModel(story, false));
890            }
891
892            log.info("All story list loaded");
893        } catch (SQLException e) {
894            // TODO Auto-generated catch block
895            log.error("Failed to load story list");
896            e.printStackTrace();
897        }
898
899        return listStory;
900    }
```

Kuva 10. StoryService-luokan metodi, jonka avulla tarinat voidaan hakea tietokannasta

Rivi 882 aloittaa metodin määrityksen, joka palauttaa listan Story-tyyppisistä olioista (Story-luokan toteutus löytyy model-hakemistosta). Haettaville tarinoille muodostetaan listStory-muuttuja, jonka alkuarvoksi asetetaan null (rivi 883). Tarinat haetaan tietokannasta ensin entiteetti-hakemistosta löytyvään Story-entiteettiin storyDao-olion avulla (rivi 885), jonka jälkeen listStory-muuttuja määritellään Javan ArrayList-tyyppiseksi (rivi 886). Haetut entiteetit lisätään for-loopissa listStory-olioon käyttämällä utils-hakemistosta löytyvää ModelEntityMappingUtils-luokan getStoryModel-metodia, joka muuntaa entiteetti-olion Story-luokan olioksi (malli). Siltä varalta, että jokin menee pieleen tietojen haussa, riveillä 893 – 896 on varauduttu tällaiseen poikkeukseen. Jos kaikki on mennyt hyvin, palautetaan listStory-olio, joka sisältää kaikki julkisella aikajanalla olevat tarinat.

Taulukkoon 7 on koottu service-luokissa käytetyt annotaatiot.

Taulukko 7. Service-luokissa käytetyt annotaatiot ja niiden merkitys

Annotaatio	**Merkitys**
@Service	Kerrotaan Spring-sovelluskehykselle, että kyseessä on palvelu-luokka.
@Autowired	Annotaatio jonka avulla dispatcher-servletissä määritellystä komponentista luodaan ilmentymä IoC-säiliön toimesta viittaamalla sovellusluokissa komponentin tunnisteessa (id) määriteltyyn nimeen.

4.3.4 DAO

DAO-luokkien tarkoitus on erottaa datan nouto-operaatiot itse dataresurssista eli verkkosovelluksen tietokannasta. Kuva 11 esittelee nouto-operaatiot, joiden avulla tietokannasta haetaan kaikki sinne tallennetut julkiset tarinat.

```
94    public List<Story> getAllStories() throws SQLException {
95
96        List<Story> listStories = null;
97
98        try {
99            Session session = sessionFactory.getCurrentSession();
100           Criteria criteria = session.createCriteria(Story.class);
101           criteria.addOrder(Order.desc("storyDate"));
102           criteria.add(Restrictions.like("publishing", "Public", MatchMode.ANYWHERE));
103           listStories = criteria.list();
104       } catch (Exception e) {
105           e.printStackTrace();
106       }
107
108       return listStories;
109
110   }
```

Kuva 11. StoryDao-luokan metodi julkisten tarinoiden hakemiseksi tietokannasta

Rivillä 94 määritellään metodi, joka palauttaa kaikki julkiset tarinat tietokannasta. Jos jokin menee pieleen, heitetään poikkeus. Haettaville tarinoille muodostetaan listStories-muuttuja, jonka alkuarvoksi asetetaan null (rivi 96), jonka jälkeen tarinat haetaan tietokannasta (rivit 98 – 104). Ensin muodostetaan Hibernaten avulla yhteys tietokantaan (rivi 99, katso lisätietoa kuvasta 5, rivit 38 - 49), jonka jälkeen määritellään palautettavien olioiden tyyppi (rivi 100) (Story-entiteettejä). Muokataan hakutuloksien järjestystä siten, että haetut tarinat järjestetään tapahtuma-ajan mukaan laskevaan järjestykseen (rivi 101). Tarkennetaan hakukriteereitä määrittelemällä haettavaksi vain julkiset tarinat (rivi 102). Rivillä 103 haetaan tiedot tietokannasta ennalta määriteltyjen kriteerien mukaisesti. Jos tarinoiden haku jostain syystä epäonnistuu, tulostetaan virheloki (rivit 104 – 106). Lopuksi rivillä 108 palautetaan haetut tarinat.

Taulukkoon 8 on koottu DAO-luokissa käytetyt annotaatiot.

Taulukko 8. DAO-luokissa käytetyt annotaatiot ja niiden merkitys

Annotaatio	**Merkitys**
@Repository	Annotaatio, jonka avulla Spring määrittelee luokan DAO-repositori-oksi.
@Resource	Annotaatio, jonka avulla Spring osaa injektoida dispatcher-servletissä määritellyn dataSourcen tähän luokkaan kiinni.
@Autowired	Annotaatio jonka avulla dispatcher-servletissä määritellystä kom-ponentista luodaan ilmentymä IoC-säiliön toimesta viittaamalla sovel-lusluokissa komponentin tunnisteessa (id) määriteltyyn nimeen.

4.3.5 Entity

Entiteetit ovat kevytrakenteisia Java-olioita, joita käytetään sovelluksen ja tietokannan väliseen kommunikointiin. Jokainen entiteetti-luokan instanssi edustaa yhtä tietuetta tietokannan taulussa ja niiden avulla määritellään tietokannan taulun rakenne sovelluskoodiin. Kuvassa 12 on esitetty story-entiteetin toteutusta.

```
1  package com.tatva.story.entity;
2
4  * TatvaSoft - Added Community entity
6  import java.util.HashSet;
23
24 @Entity
25 @Table(name="community")
26 public class Community
27 {
28     @Id
29     @GeneratedValue(strategy = GenerationType.IDENTITY)
30     @Column(name="ID")
31     private long id;
```

Kuva 12. Story-entiteetin sovelluskoodia

Annotaatiolla @Entity (rivi 24) ilmaistaan Spring-sovelluskehykselle, että kyseessä on entiteetti-luokka. @Table-annotaatio (rivi 25) viittaa tiettyyn tietokannan tauluun. @Id-annotaation (rivi 28) avulla entity-olioon generoidaan valitun tietokantataulun tietuetta vastaava pääavaimen arvo. @GeneratedValue-annotaation (rivi 29) avulla kerrotaan Spring-sovelluskehykselle, että pääavaimen arvo generoidaan automaattisesti tietokannasta saadun arvon perusteella. @Column-annotaatio (rivi 30) määrittää, mitä tietokantataulun kenttää käytetään annotaation alapuolella määritellyn jäsenmuuttujan (rivi 31) asettamiseen.

Taulukkoon 9 on koottu entity-luokissa käytetyt annotaatiot.

Taulukko 9. Entity-luokissa käytetyt annotaatiot ja niiden merkitys

Annotaatio	**Merkitys**
@Entity	Määritellään, että tämä luokka sisältää entiteetin.
@Table	Määritellään mihin tietokannan tauluun tällä luokalla viitataan.
@Id	Tämän annotaation alla olevalla luokan jäsenmuuttujalla viitataan tämän entiteetin pääavaimeen (primary key).

@GeneratedValue	Tällä merkitään jäsenmuuttuja, jonka arvoa kasvatetaan automaattisesti tietokannasta saadun uuden arvon perusteella.
@Column	Tällä annotaatiolla määritellään sen alapuolella oleva luokan jäsenmuuttuja tietokantataulun tiettyyn kenttään.
@OneToMany	Yhden suhde moneen -yhteyden määritys.
@ManyToOne	Monen suhde yhteen -yhteyden määritys.
@JoinTable	Kytkös välitauluun, jonka avulla määritellään monen suhde moneen -yhteys.
@JoinColumn	Määrittelee mihin kohdetaulun sarakkeeseen yhteys määritetään.

4.3.6 App

App-hakemisto sisältää ainoastaan yhden luokan, joka laajentaa Red5-palvelimen applicationadapter-luokkaa. Hakemistossa olevan luokan tehtävänä on toimia perusluokkana koko verkkosovellukselle ja käynnistää sovellus palvelimella.

4.3.7 Interceptor

Interceptor-hakemisto sisältää kaksi interceptor-luokkaa eli sieppaajaa, joiden tarkoitus on siepata sovelluksen kontrollivirta tietyissä tilanteissa. Ensimmäinen näistä on LoginInterceptor, jonka tarkoitus on estää käyttäjän pääsy tiettyihin resursseihin, mikäli hän ei ole kirjautunut sisään.

Toinen näistä luokista on FlashScopeInterceptor, jonka tarkoitus on sitoa flashScope nykyiseen sessioon tai poistaa se nykyisestä sessiosta. Mikäli se on jo sidottu, lomitetaan se olemassa olevan flashScopen kanssa.

4.3.8 Util

Util-hakemisto sisältää apuluokkia (utilities) sovelluksessa usein käytettyjen toiminnallisuuksien toteuttamiseksi. Kuvassa 13 on esitetty erään apuluokan toteutus, jonka tarkoituksena on muuntaa sille välitetty Story-entiteetti Story-malliksi.

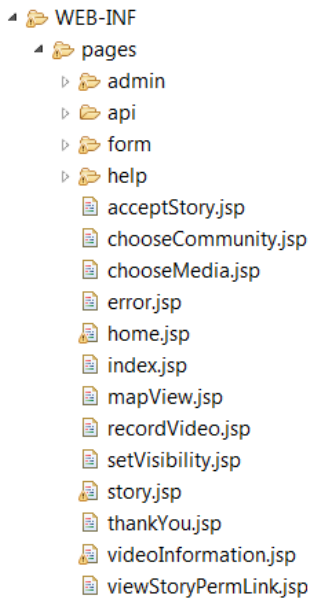
```
169    public static com.tatva.story.model.Story getStoryModel(Story entity, boolean readUserCommunities)
170
171        com.tatva.story.model.Story model = new com.tatva.story.model.Story();
172
173        model.setId(entity.getId());
174        model.setTitle(entity.getTitle());
175        model.setCity(entity.getCity());
176        model.setCountry(entity.getCountry());
177        model.setStoryDate(entity.getStoryDate());
178        model.setPublishing(entity.getPublishing());
179        model.setReady(entity.getReady());
180        model.setCreated(entity.getCreated());
181        model.setLattitude(entity.getLattitude());
182        model.setLongitude(entity.getLongitude());
183        model.setYear(entity.getYear());
184        model.setUser(getUserModel(entity.getUser(), readUserCommunities));
185        model.setStoryType(entity.getStoryType());
186        model.setPermanentLinkId(entity.getPermanentLinkId());
```

Kuva 13. Apuluokka Story-entiteetin muuttamiseksi Story-malliksi

Näiden luokkien tarkoitus on hajauttaa verkkosovelluksessa usein käytetyt toiminnallisuudet omaan hakemistoonsa, jotta redundanssin määrä sovelluskoodissa voidaan minimoida.

4.4 Pages-hakemisto

Verkkosovellus sisältää vielä yhden olennaisen hakemiston, joka on irrallaan sovelluksen luokkarakenteesta. Pages-kansio sisältää JSP-sivut, joiden avulla informaatio esitetään lopullisessa muodossaan (kuva 14). Springin Web MVC -mallissa JSP-sivut ovat näkymiä (Views).



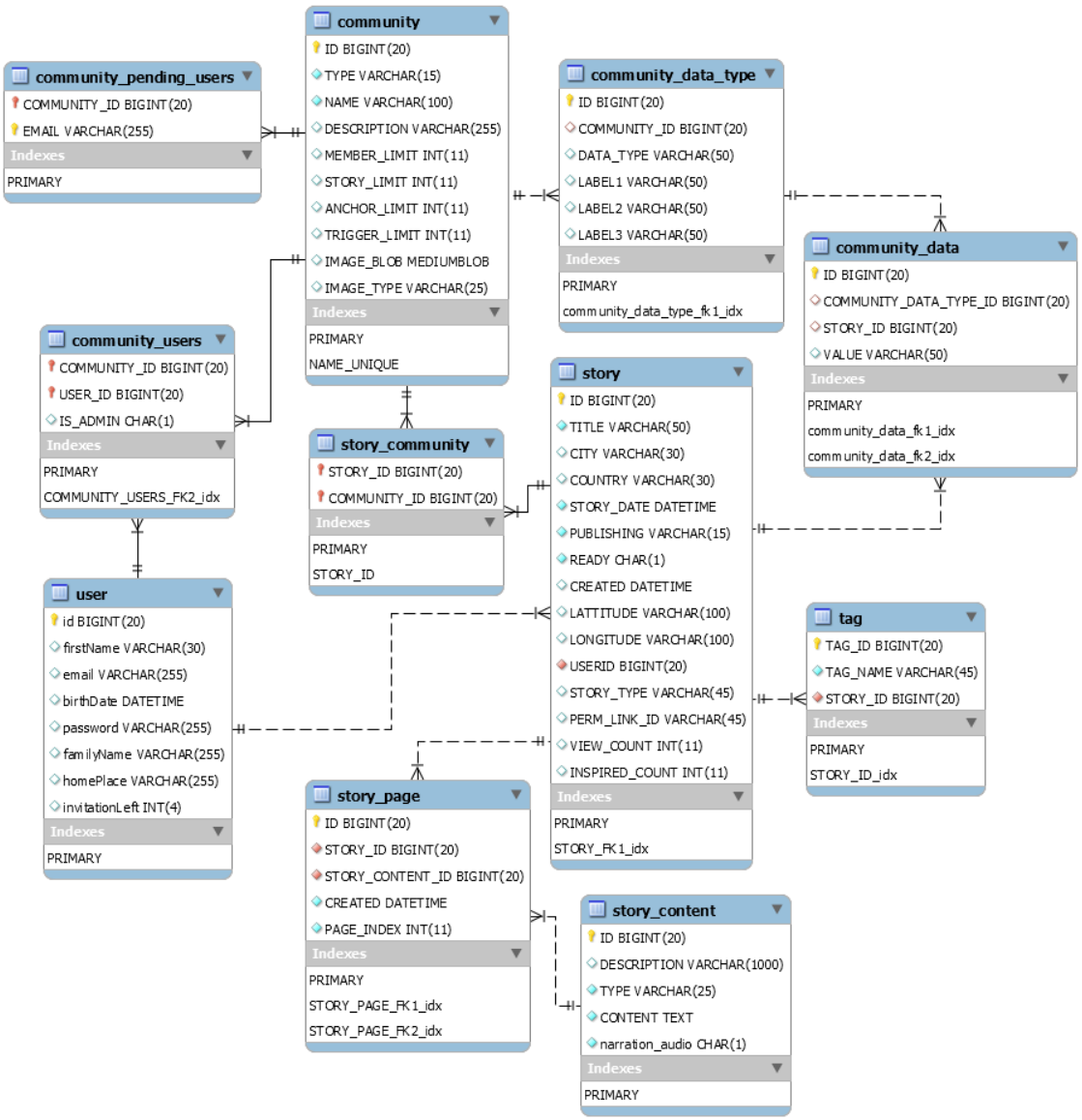
Kuva 14. JSP-sivut, jotka vastaavat MVC-mallin View-osaa

4.5 Verkkosovelluksen tietokanta

Avaan tässä kappaleessa verkkosovelluksen tietokannan teknistä toteutusta ja selvennän ER (Entity Relationship) -mallinnuksen termein, millaisia yhteyksiä tietokannan taulujen välille on luotu.

4.5.1 Yleiskuva

Verkkosovelluksessa käytetty tietokanta pohjautuu kuvassa 15 esitettyihin tauluihin.



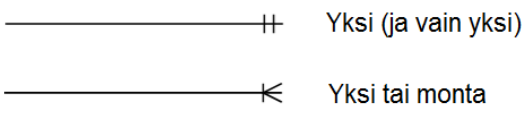
Kuva 15. Tietokannan rakenne

Kuvassa näkyvät tietokannan taulujen nimet ja niiden selitteet on koottu taulukkoon 10.

Taulukko 10. Tietokannassa olevat taulut ja niiden selite

Taulu	**Selite**
story	Sisältää tarinoiden otsikkotiedot, päivämäärät, koordinaatit.
story_page	Sisältää viittaukset kuhunkin tarinaan lisättyihin välilehtiin.
story_content	Sisältää tiedot mediaresursseista ja näiden sijainnista palvelimella liittyen kuhunkin tarinaan.
story_community	Välitaulu, jonka avulla muodostetaan monen suhde moneen yhteys yhteisö ja tarina-taulun välille.
community	Sisältää tiedot palveluun lisätyistä yhteisöistä.
community_data	Sisältää vain tietylle yhteisölle käytössä olevat lisäkentät, joilla tarinan kuvausta voidaan tarkentaa.
community_data_type	Sisältää määrittelyt siitä millaisia lisäinformaatiokenttiä on määritelty tälle tietylle yhteisölle.
community_users	Sisältää tiedot ketkä kuuluvat kuhunkin yhteisöön ja tiedon siitä onko yhteisöön kuuluvalla jäsenellä admin-ominaisuuksia.
community_pending_users	Sisältää tiettyyn yhteisöön kutsuttujen henkilöiden sähköpostiosoitetiedot.
user	Sisältää tiedot palveluun rekisteröityneistä käyttäjistä, heidän sähköpostiosoitteistaan ja salasanoistaan (enkryptattuja).
tag	Sisältää tiedot tarinoihin lisätyistä tunnisteista eli tageista.

Tietokannan kaikki taulut on kytketty toisiinsa yhden suhde moneen -yhteyksillä lukuun ottamatta yhteyttä tarina- ja yhteisö-taulun sekä yhteisö- ja käyttäjä-taulun välillä (kuva 15). Näiden em. taulujen välille on luotu monen suhde moneen -yhteys käyttäen välitaulua apuna. Taulujen kytköksissä käytetyt symbolit on esitetty kuvassa 16.



Kuva 16. Tietokannassa käytetyt yhteystyypit

Symboleiden avulla viitataan siihen, millainen yhteys tietokannassa olevien taulujen välille on luotu. Kuvaa 12 katsottaessa voidaan havaita esim., että story- ja story_page-taulujen välille on luotu yhteys. Story-taulun päässä yhteystyyppi on yksi (ja vain yksi) kun taas story_page-taulun päässä yhteystyyppi on yksi tai monta. Alla olevassa kappaleessa esittelen, kuinka kuvaa voidaan tulkita. Esittelen, miten asiat kytkeytyvät toisiinsa

tarina- ja yhteisö-taulun näkökulmista katsottuna, koska ne ovat merkittävimmät taulut tietokannan toiminnan kannalta.

Tietokanta on rakennettu tarina (story)-taulun ympärille. Kukin tarina voi kuulua moneen yhteisöön (community) ja kukin yhteisö voi sisältää monta tarinaa. Kullakin tarinalla on käyttäjä (user), joka on lisännyt tarinan ja tämä kyseinen henkilö voi olla lisännyt useita tarinoita. Kukin tarina voi sisältää useita sivuja (story_page) ja kullakin näistä sivuista on yksi sisältö (story_content). Kukin tarina voi sisältää myös useita tunnisteita (tag) ja useita eri lisätietokenttiä (community_data).

Kukin yhteisö voi sisältää useita käyttäjiä, joille on lähetetty kutsu palveluun mutta jotka eivät ole vielä rekisteröityneet (community_pending_users). Kukin käyttäjä, jolle kutsu on lähetetty, voi toisaalta olla kutsuttuna vain yhteen yhteisöön (community) kerrallaan. Kukin yhteisö voi sisältää useita käyttäjiä, ja kukin käyttäjä voi kuulua moneen yhteisöön. Kukin yhteisö voi sisältää useita tälle yhteisölle määriteltyjä lisäinformaatiokenttiä (community_data_type).

4.5.2 Indeksointi

Kun tietokannan koko kasvaa ja taulujen tietueiden määrä lisääntyy, tietokantahakuihin kuluva aika lisääntyy. Tietokantahakujen tehokkuutta voidaan kuitenkin parantaa käyttämällä indeksointia. Indeksoinnissa muodostetaan tietokannan tauluun ns. aputietorakenne, jonka avulla vältetään se, että tietokantahaussa jokainen tietue tulisi käydä läpi. Menetelmä toimii hieman samaan tapaan kuin tietyn nimen etsiminen puhelinluettelosta. Etsittäessä s-kirjaimella alkavia sukunimeä ei ole tarpeen käydä koko puhelinluetteloa läpi vaan tiedetään, että s-kirjaimella alkavat nimet sijaitsevat tietyssä kohtaa puhelinluetteloa. Indeksoinnin avulla mahdollistuu tiedon etsiminen pienemmästä datajoukosta. Verkkosovelluksen tietokantaan on luotu indeksi kuuteen tauluun (community_users, community_data_type, community_data, story, story_page, tag).

4.5.3 Turvallisuus

Työn kirjoitushetkellä tietokantaan ei ole määritelty erillisiä varmuuskopiotoimenpiteitä, jolloin vastuu varmuuskopioista on tietokantaa hallinnoivalla henkilöllä. Pelkkä tietokannasta otettu varmuuskopio ei kuitenkaan ole riittävä toimenpide tämän verkkosovelluk-

sen tapauksessa, koska palvelimelle tallentuu käyttäjien lähettämiä mediatiedostoja, joihin viitataan tietokannassa. Mikäli mediatiedostoja ei ole varmuuskopioitu, tietokannassa olevat viittaukset kohdistuvat tiedostoihin, joita ei ole olemassa.

Eräs ratkaisu tähän voisi olla se, että palvelimelle kirjoitetaan skripti, jonka avulla voidaan suorittaa varmuuskopiointi sekä tietokannasta että palvelimella sijaitsevista mediatiedostoista. Palvelun alkutaipaleella tämä voitaisiin toteuttaa ns. kylmänä varmuuskopiona, jolloin tietokanta otetaan pois käytöstä varmuuskopioinnin ajaksi. Tällä varmistetaan se, että tietokannasta saadaan luotua varmuuskopio, joka on sisäisesti ristiriidaton. Jos varmuuskopio tehdään tietokannan ollessa käytössä, kopioon voi tallentua dataa, joka on muuttunut varmuuskopioinnin aikana ja täten aiheuttaa sisäisen ristiriitatilanteen. Kun palvelua käyttävien henkilöiden määrä kasvaa ja palvelun saavutettavuuden merkitys täten korostuu, kannattanee siirtyä kuumaan varmuuskopiointiin, jolloin data kopioidaan tietokannan ollessa käytössä. Näin toimimalla käyttökatkokset palvelussa ovat minimaalisia. Toisaalta tällöin tulee myös varautua tilanteisiin, joissa varmuuskopio on sisäisesti ristiriitaisessa tilassa, jolloin ristiriitatilanteiden ratkaisemiseen tulee myös varautua.

Tällä hetkellä tietokannassa operoidaan root-tunnuksella, jolla on täydet oikeudet suorittaa tietokannassa mitä tahansa operaatioita. Palvelun turvallisuuden lisäämiseksi tietokantaan tulisi lisätä ainakin käyttäjätili, jolla on vähäisemmät oikeudet tietokantaoperaatioihin. Näin toimimalla palvelun turvallisuus lisääntyisi, koska tiettyjen operaatioiden suorittaminen tietyillä tunnuksilla ei olisi ylipäätään mahdollista.

4.5.4 Eheyden hallinta

Tietokannassa oleville tauluille on määritelty eheysrajoitteita, joilla pyritään varmistumaan siitä, että tietokantaan pääsee mahdollisimman vähän virheellistä tietoa ja että tietokanta säilyy eheänä. Eheysrajoitteita on kolmea tyyppiä, jotka on esitelty taulukossa 11.

Taulukko 11. Eheysrajoitteet ja niiden määritelmä

Eheysrajoitteen tyyppi	**Määritelmä**
Avainrajoite	Kaikilla taulun tietueilla tulee olla uniikki perusavaimen arvo, jolla kyseessä oleva tietue voidaan tunnistaa yksilöllisesti.
Viite-eheysrajoite	Tietokannan taulussa olevan viiteavaimen arvon on esiinnyttävä kohdetaulun jossain tietueessa perusavaimen arvona.
Arvoaluerajoite	Kunkin tietokannan taulun tietueen kenttään syötettävän arvon tulee olla sallituiksi määriteltyjen arvojen joukossa.

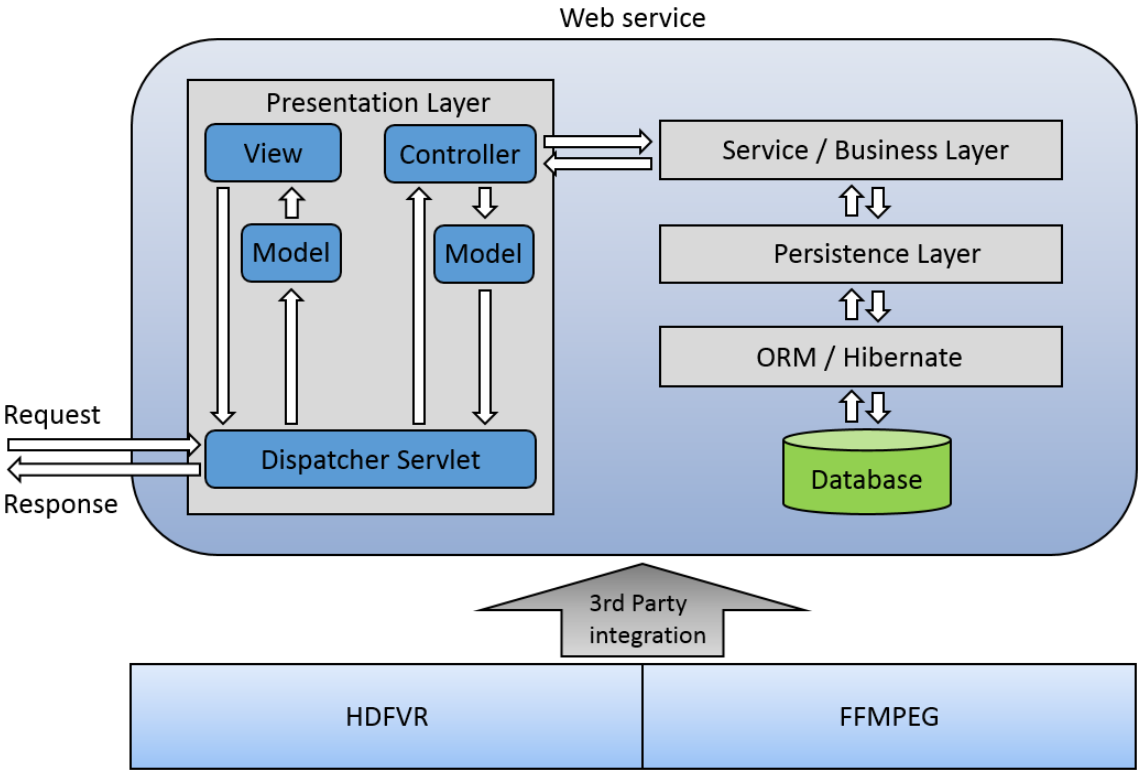
Kukin tietokannassa oleva taulu koostuu tietueista, jotka voidaan yksilöllisesti tunnistaa tietueisiin merkityn kentän (perusavain = id) arvon avulla. Jokainen tietokannassa oleva perusavain koostuu enintään 20-merkkisestä kokonaisluvusta, joka mahdollistaa $10^{20} - 1$ uniikkia arvoa.

Tietyt päivitys- tai poisto-operaatiot tietokantaan on estetty. Esim. tarina-taulusta ei voida suoraan poistaa tietueita, koska tarina-taulun ja käyttäjä-taulun (user) välille on luotu viite-eheysrajoite, joka rikkoontuisi poiston myötä.

Kullekin tietokannan taulussa olevalle kentälle on asetettu arvoaluerajoite, jonka avulla rajoitetaan sitä, millaista dataa kyseessä olevaan kenttään voidaan asettaa. Esim. jokaisella tarinalla oleva otsikkotieto (TITLE) voi olla enintään 50 merkkiä pitkä sarja alfanumeerisia merkkejä.

4.6 Verkkosovelluksen yleisarkkitehtuuri

Verkkosovelluksen yleisarkkitehtuuri on esitetty kuvassa 17.



Kuva 17. Verkkosovelluksen arkkitehtuurikaavio

Kun pyyntö saapuu dispatcher-servletille, se välittää sen eteenpäin pyynnöstä vastuussa olevalle ohjaimelle. Pyynnön vastaanottaneeseen ohjaimeen määritelty metodi ottaa vastuun pyynnön käsittelystä ja käyttää tarvittaessa palvelu-/bisneskerrokseen ohjelmoituja toimintoja luodessaan vastauksen. Palvelu-/bisneskerros voi hakea tai viedä tietoa tietokantaan käyttäen apuna pysyvyyskerrosta. Pysyvyyskerrokseen kuuluu mm. entiteetit, joiden avulla tietokannan tilaa mallinnetaan sovelluskoodissa. Tiedot viedään / haetaan tietokannasta käyttäen apuna Hibernatea, jonka avulla oliopohjainen sovelluskoodi muutetaan relaatiotietokantaan sopivaksi ja päinvastoin. Kun tarvittavat operaatiot on suoritettu tietokannassa, kontrolli palautuu ohjaimelle, joka palauttaa saamansa tiedon mallin välityksellä dispatcher-servletille. Kun dispatcher-servlet on vastaanottanut mallin, se välittää sen siitä vastuussa olevalle näkymälle, jonka tehtävä on tuottaa lopullinen vastaussivu. Kun vastaussivu on luotu, se välitetään takaisin dispatcher-servletille, joka palauttaa sen vastauksena luotuun pyyntöön.

Kolmannen osapuolen integraatio on arkkitehtuurin ulkopuolella, koska nämä ovat ulkoisia hakemistoja, jotka otetaan resursseina käyttöön verkkosovelluksessa.

5 Verkkosovelluksen jatkokehitys Eclipsessä

Tämä luku sisältää ohjelmiston jatkokehityksen kannalta oleelliset järjestelmä- ja kehitysympäristön konfiguraatiot. Alkuvaiheessa verkkosovelluksen käyttöönotto tuotti merkittäviä ongelmia ja paljon päänvaivaa, koska kehitysympäristö oli konfiguroitu väärin ja aiheutti tästä johtuen erikoisia poikkeuksia, joihin oli todella vaikeaa löytää ratkaisu. Tämä luku onkin kirjoitettu ajatellen, että tämänkaltaiset ongelmat voitaisiin tulevaisuudessa välttää käyttämällä tätä ohjeistusta. Ohjeistus on kirjoitettu Windows-ympäristöön.

5.1 JDK (Java Development Kit) -asennus ja ympäristömuuttujien asettaminen

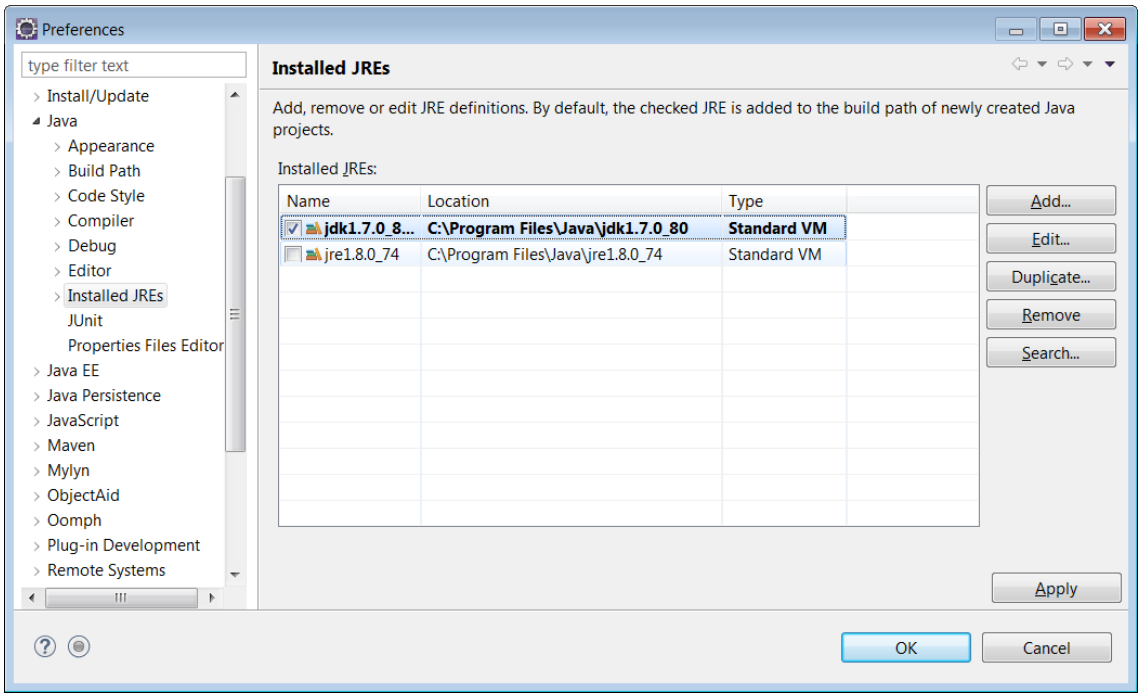
Ennen kuin verkkosovellusta voidaan lähteä jatkokehittämään, tulee varmistua siitä, että käyttöjärjestelmään on asennettu riittävän uusi JDK-versio, jonka avulla Java-pohjaisia sovelluksia voidaan kehittää ja kääntää. Verkkosovellus on luotu Java JDK -versiolla 7, joten tässä esimerkissä käytetään samaa versiota.

Mikäli JDK ja ympäristömuuttujat on jo asetettu asianmukaisesti, voidaan hypätä lukuun 6.2. Mikäli JDK:ta ei ole asennettu, tulee se ladata osoitteesta:

http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

Kun JDK on ladattu, tulee se asentaa koneelle, minkä jälkeen tulee asettaa ympäristömuuttujat, jotta Javaa voidaan ajaa työskentelyhakemistosta riippumatta. Tärkeä ympäristömuuttuja on JAVA_HOME, jonka arvoksi tulee määrittää hakemisto, johon JDK asennettiin.

Eclipsen Java-kääntäjäksi tulee asettaa juuri ladattu Javan JDK. Java-kääntäjän asetukset löytyvät valikon Window -> Preferences alta (kuva 18).



Kuva 18. Oikean JDK-version asettaminen

Kun nämä esitoimenpiteet on suoritettu, voidaan siirtyä seuraavaan lukuun.

5.2 Red5-palvelimen asennus ja konfigurointi

Jotta verkkosovellusta voidaan ajaa palvelimella, tulee sille ladata palvelinohjelmisto. Verkkosovellus käyttää palvelimenaan Red5-palvelinta, joka voidaan ladata osoitteesta:

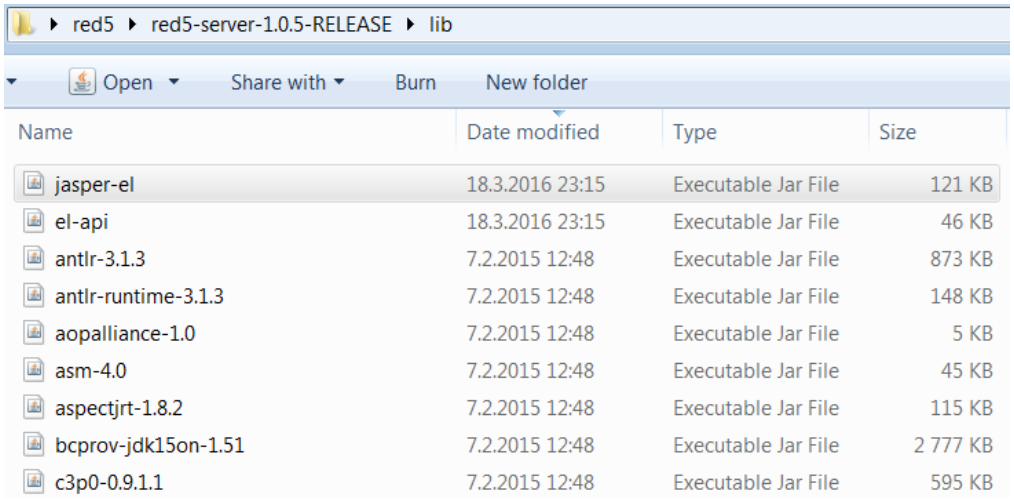
https://github.com/Red5/red5-server/releases/download/v1.0.5-RELEASE/red5-server-1.0.5-RELEASE-server.zip

Kun Red5 on ladattu, tulee ZIP-tiedosto purkaa haluttuun hakemistoon, joka toimii palvelimen käynnistyshakemistona. Jotta poikkeuksilta liittyen puuttuviin luokkakirjastoihin voidaan välttyä, palvelimelle tulee ladata kaksi lisäkirjastoa, jotka ovat nimeltään jasper-el.jar ja el-api.jar. Nämä tiedostot löytyvät osoitteesta:

http://mvnrepository.com/artifact/org.apache.tomcat/jasper-el

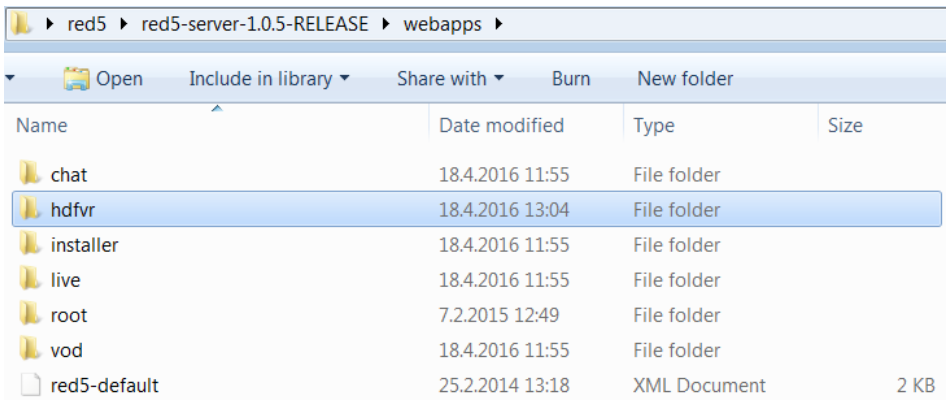
http://mvnrepository.com/artifact/javax.el/javax.el-api

Kun tiedostot on ladattu, tulee ne siirtää palvelinhakemiston juurihakemistossa sijaitsevaan /lib-hakemistoon (kuva 19).



Kuva 19. Kaksi lisättyä JAR (Java Archive) -tiedostoa palvelimen kirjastohakemistossa

Edellä mainittujen kirjastojen lisäksi palvelimen /webapps-hakemistoon tulee siirtää lähdekoodin mukana tullut HDFVR-hakemisto, joka sisältää medioiden nauhoitukseen tarvittavan kolmannen osapuolen apuohjelmiston (kuva 20).



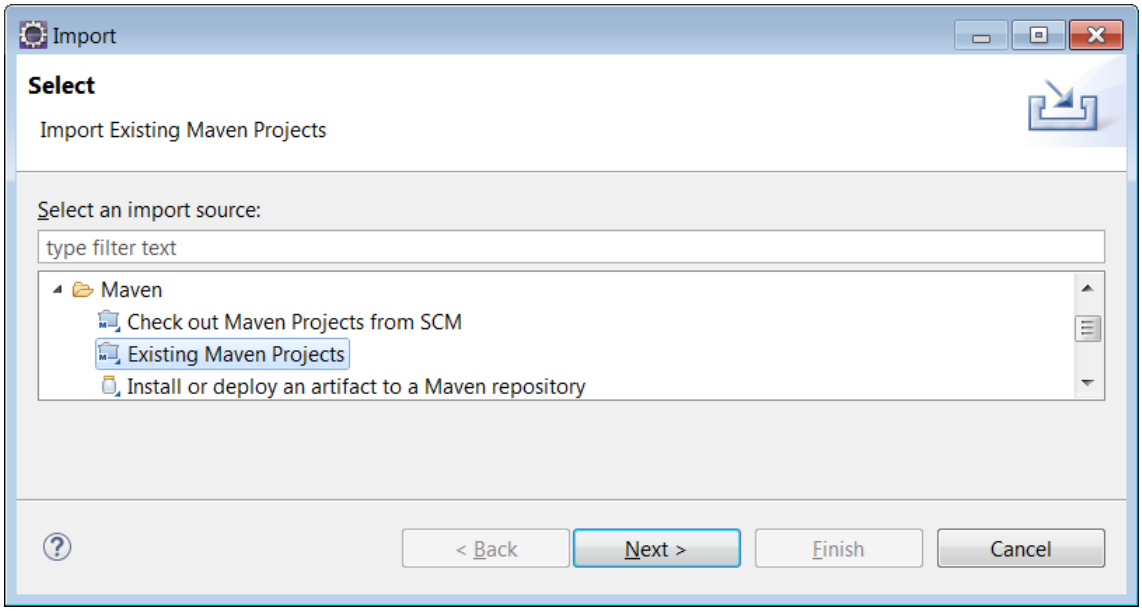
Kuva 20. HDFVR-apuohjelmisto palvelimen webapps-hakemistossa

Kun nämä palvelimen konfigurointiin liittyvät toimenpiteet ovat suoritettu, voidaan siirtyä kappaleeseen 6.3.

5.3 Projektin tuonti Eclipseen

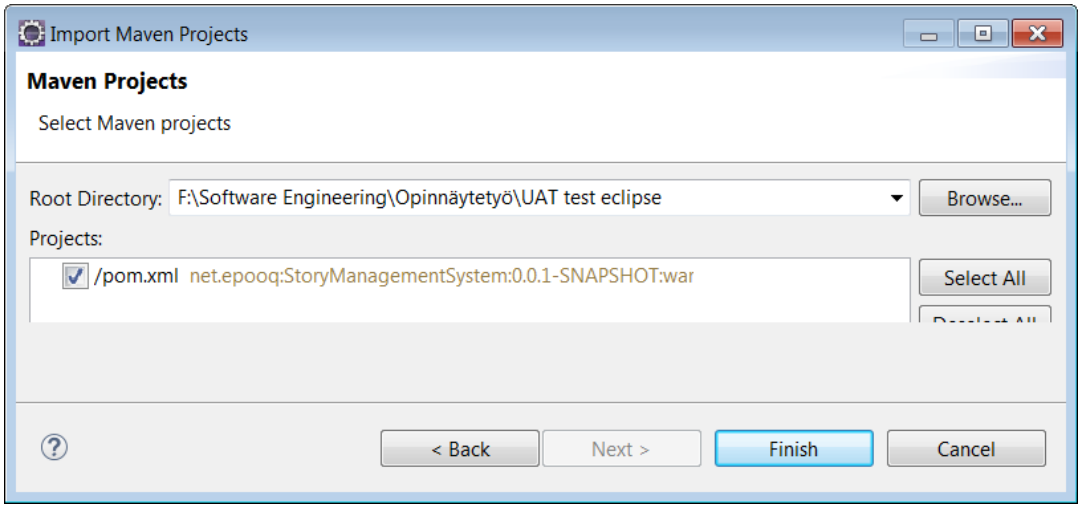
Verkkosovellus on koottu käyttäen Maven-koontityökalua, joten sen tuominen Eclipseen tapahtuu seuraavasti (kuva 21).

File -> Import -> Maven -> Existing Maven Projects



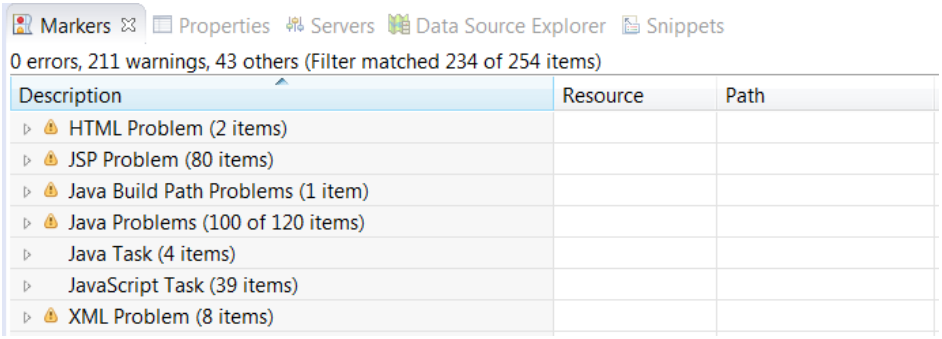
Kuva 21. Projektin tuominen Eclipseen (1)

Kuvan 21 kohdan jälkeen klikataan Next, jonka jälkeen avautuu kuvan 22 mukainen näkymä.



Kuva 22. Projektin tuominen Eclipseen (2)

Kuvan 16 kohdassa tulee valita projektin root directoryksi hakemisto, jossa projektin lähdekoodi sijaitsee. Kun oikea hakemisto on valittu, Eclipse löytää projektin POM (Project Object Model) -tiedoston, jonka avulla Maven-projekti voidaan aukaista Eclipsessä. Klikataan Finish, ja projekti latautuu Eclipseen. Kuvassa 23 on tilanne sen jälkeen, kun projekti on tuotu Eclipseen.

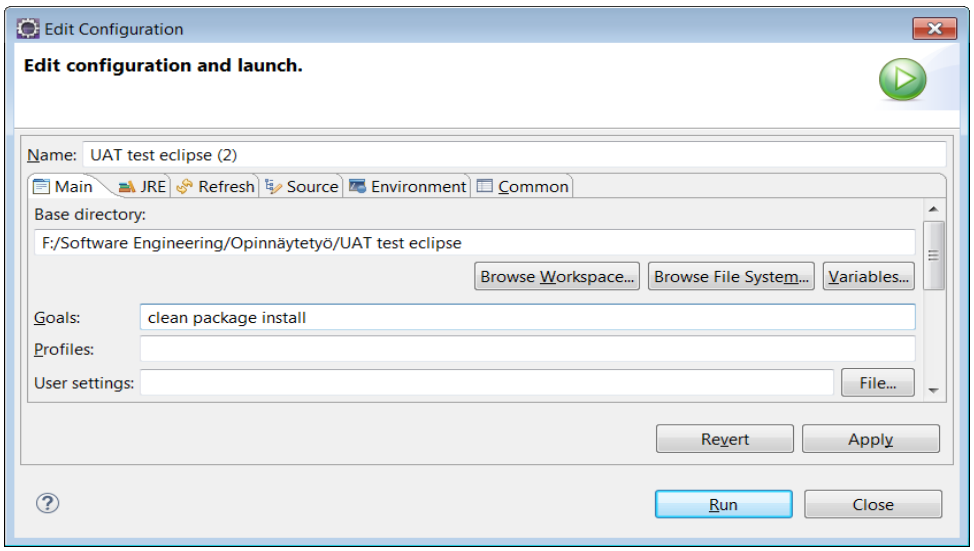


Kuva 23. Eclipsen viestit koskien projektia

Jos kaikki on mennyt oikein, projekti sisältää 0 error-viestiä, warnings-viestejä voi olla.

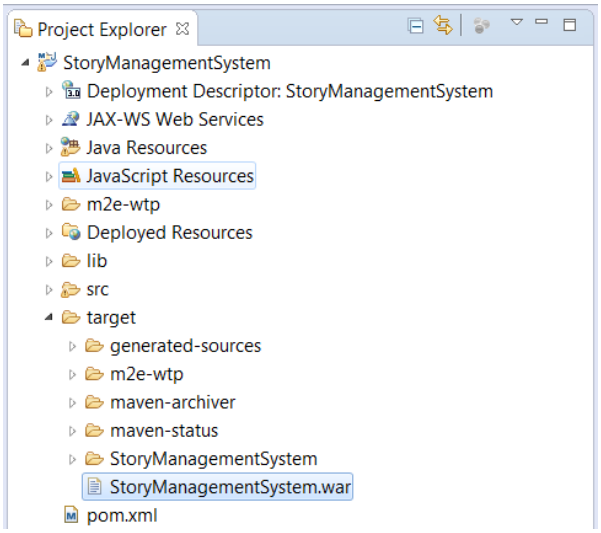
5.4 Koontiversion luonti

Kun projekti on tuotu Eclipseen, voidaan sovelluskoodista luoda koontiversio, jota voidaan ajaa palvelimella. Klikataan projektia oikealla hiiren napilla ja valitaan Run As -> 6 Maven Build, jonka jälkeen avautuu kuvan 24 mukainen näkymä.



Kuva 24. Koontiversion luominen käyttäen Maven-koontityökalua

Mavenin goalseiksi asetetaan clean package install ja painetaan Run, jonka jälkeen Eclipse luo tarvittavan WAR-tiedoston projektihakemiston /target-hakemistoon (kuva 25).

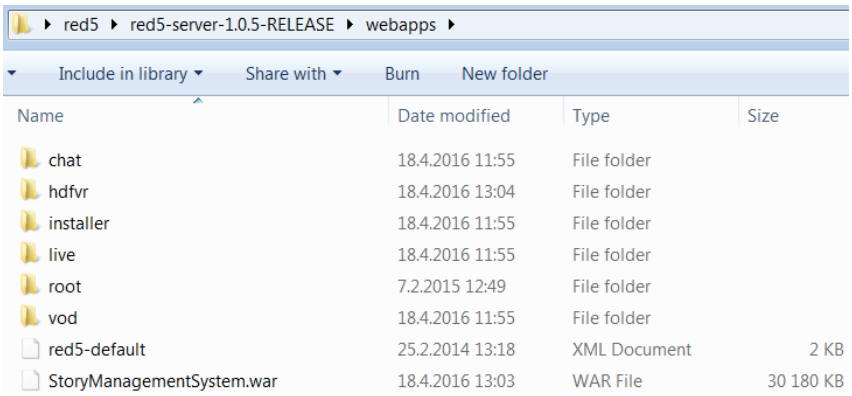


Kuva 25. Luotu koontiversio Eclipsessä

Kun koontiversio on luotu, voidaan projekti ottaa käyttöön Red5-palvelimella.

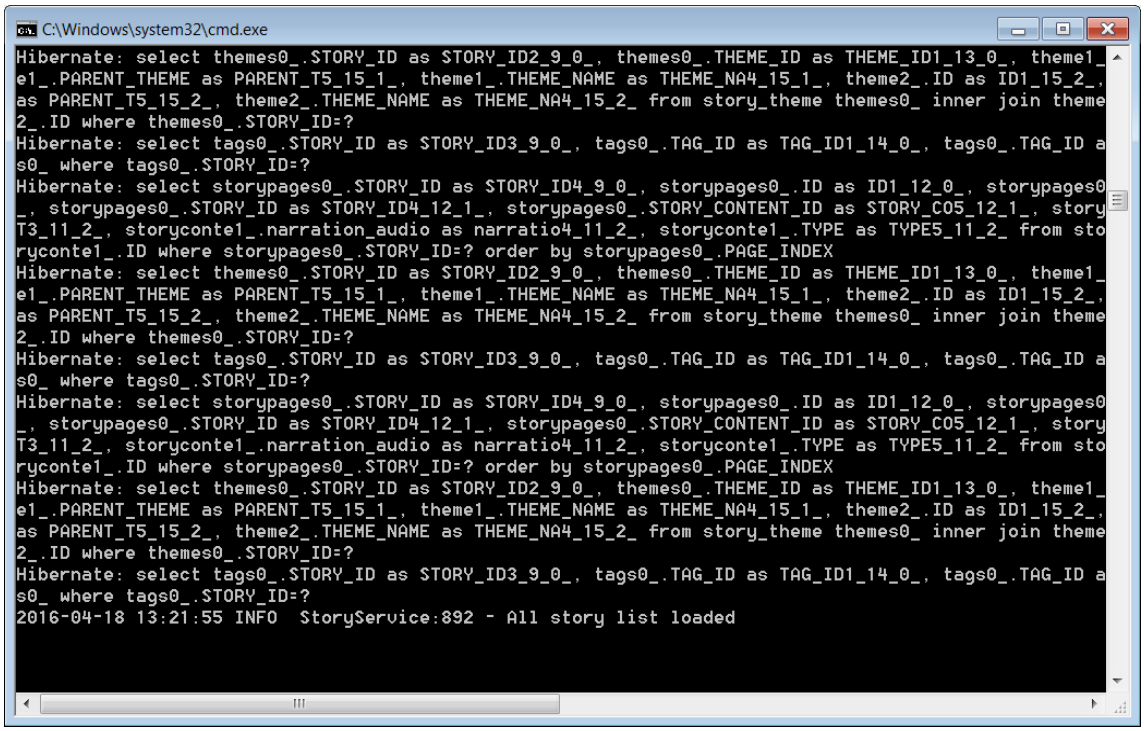
5.5 Koontiversion käyttöönotto palvelimella

Koontiversion käyttöönottamiseksi palvelimella tulee se siirtää Red5:n webapps-hakemistoon (kuva 26).



Kuva 26. Koontiversion siirto Red5-palvelimelle

Kun koontiversio on siirretty webapps-hakemistoon, palvelin voidaan käynnistää. Tämä tapahtuu Red5:n juurihakemistossa klikkaamalla red5.bat-tiedostoa. Palvelimen käynnistyksen jälkeen /webapps-hakemistoon ilmestyy WAR-tiedostonnimeä vastaava hakemisto, jota palvelin alkaa tarjoilla. Kuvassa 27 nähdään palvelimen tila siinä vaiheessa, kun sovellus on aktiivisessa tilassa.



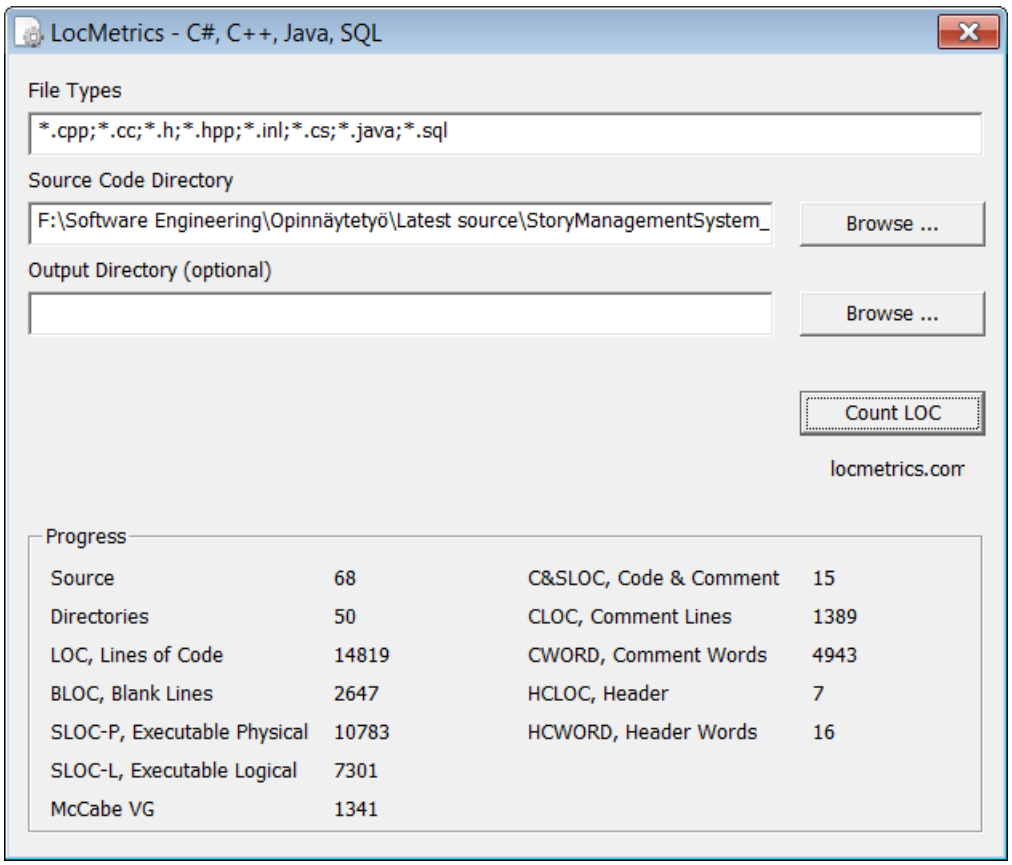
Kuva 27. Koontiversion ajaminen palvelimella

6 Yhteenveto

Insinöörityön tavoitteena oli tutustua verkkosovelluksessa käytettyihin ohjelmistoteknologioihin. Työn tuloksena syntyi tämä dokumentti, joka sisältää olennaisimmat asiat verkkosovelluksen toiminnan ymmärtämiseksi. Toiminnan kannalta keskeisimmässä roolissa on Spring-sovelluskehys, ja etenkin Spring Web MVC, jonka ympärille koko sovellus on rakennettu.

Koska verkkosovellus sisältää itsessään n. 11 000 koodiriviä (kuva 28), laajamittaisempi läpikäyminen ja toiminnan dokumentointi vaatisi huomattavan paljon enemmän aikaa, kuin käytössäni oli. Muut työtehtävät ja aikataululliset rajoitukset aiheuttivat sen, että työstä jäi pois Three.js-grafiikkakirjaston käyttö sovelluksessa ja JSP-sivujen syvällisempi käsittely. Nämä ovat itsessään kuitenkin sen verran laajoja aihealueita (clouds.js,

2151 riviä koodia), että niiden toiminnasta saisi tehtyä oman opinnäytetyön. Three.js ja JSP ovat kuitenkin erittäin hyvin dokumentoitu, ja oppaat näiden teknologioiden käyttöön on helposti saatavilla Internetistä.



Kuva 28. Verkkosovelluksen koodianalyysi käyttäen LocMetrics-työkalua

Verkkosovelluksen kehitystyö jatkuu tämän insinöörityön dokumentoinnin jälkeen, ja dokumenttia voidaan käyttää apuvälineenä sovelluksen jatkokehityksessä. Etenkin yritykseen rekrytoitaville uusille työntekijöille dokumentti on hyödyllinen, koska se sisältää tiivistettynä olennaisimmat asiat sovelluksen toiminnasta.

Lähteet

1 World Wide Web. 2015. Verkkodokumentti. <https://fi.wikipedia.org/wiki/World_Wide_Web>. Päivitetty 15.02.2015. Luettu 09.03.2016.

2 Red5 – Reference Documentation. 2015. Verkkodokumentti. <http://red5.googlecode.com/svn/doc/trunk/reference/pdf/red5-reference-1.0.pdf>. Luettu 09.03.2016.

3 Red5 vs. Wowza. Verkkodokumentti. 2015. <https://www.red5server.com/red5-vs-wowza/>. Luettu 11.04.2016.

4 Comparison Red5 vs. Wowza vs. Adobe Media Server. 2014. Verkkodokumentti. <http://hosting-marketers.com/news/2014/01/12/comparison-between-wowza-media-server-vs-red5-and-vs-adobe-flash-media-server/>. Päivitetty 12.02.2014. Luettu 11.04.2016.

5 Java (programming language). 2016. Verkkodokumentti. <https://en.wikipedia.org/wiki/Java_(programming_language)>. Päivitetty 22.04.2016. Luettu 09.03.2016.

6 Differences between Java EE and Java SE. 2016. Verkkodokumentti. <http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html>. Luettu 09.03.2016.

7 Introduction to Spring Framework. 2016. Verkkodokumentti. <http://docs.spring.io/spring/docs/4.0.2.RELEASE/spring-framework-reference/html/overview.html>. Luettu 11.04.2016.

8 Spring Framework – Overview. 2015. Verkkodokumentti. <http://www.tutorialspoint.com/spring/spring_overview.htm>. Päivitetty 01.03.2015. Luettu 09.03.2016.

9 Hibernate ORM. 2016. Verkkodokumentti. <http://hibernate.org/orm/>. Luettu 09.03.2016.

10 What is Object/Relational Mapping? 2016. Verkkodokumentti. <http://hibernate.org/orm/what-is-an-orm/>. Luettu 12.04.2016.

11 HTML5 Video Recording. 2016. Verkkodokumentti. <https://hdfvr.com/html5-video-recording>. Luettu 23.03.2016.

12 Can I use? 2016. Verkkodokumentti. <http://caniuse.com/#feat=mpeg4>. Luettu 23.03.2016.

13 Bootstrap (front-end framework). 2016. Verkkodokumentti. <https://en.wikipedia.org/wiki/Bootstrap_%28front-end_framework%29>. Päivitetty 25.04.2016. Luettu 23.03.2016.

14 JavaServer Pages. 2016. Verkkodokumentti. <https://en.wikipedia.org/wiki/JavaServer_Pages>. Päivitetty 26.04.2016. Luettu 11.04.2016.

15 Three.js. 2016. Verkkodokumentti. <https://en.wikipedia.org/wiki/Three.js>. Luettu 09.03.2016.

16 Three.js. Creating a scene. 2016. Verkkodokumentti. <http://threejs.org/docs/index.html#Manual/Introduction/Creating_a_scene>. Luettu 28.03.2016.

17 Front controller. 2016. Verkkodokumentti. <https://en.wikipedia.org/wiki/Front_controller>. Päivitetty 15.04.2016. Luettu 19.04.2016.

18 Spring Web MVC framework. 2016. Verkkodokumentti. <http://docs.spring.io/spring/docs/4.0.2.RELEASE/spring-framework-reference/html/mvc.html>. Luettu 11.04.2016.