HAAGA-HELIA
ammattikorkeakoulu

REST-rajapinta Java Spring-kehikolla
Case: Raka Tuki Oy TQM-Controller

Simo-Pekka Kerkelä

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2014

HAAGA-HELIA ammattikorkeakoulu

Tiivistelmä

14.5.2014

Tietojenkäsittelyn koulutusohjelma

Tekijä tai tekijät Simo-Pekka Kerkelä	**Ryhmä tai aloitusvuosi** 2011 Syksy
Opinnäytetyön nimi REST-rajapinta Java Spring-kehikolla Case: Raka Tuki Oy TQM-Controller	**Sivu- ja liitesivumäärä** 26 + 9
Ohjaaja tai ohjaajat Ismo Harjunmaa	

RakaTuki Oy on Helsinkiläinen ohjelmistoyritys, jonka tärkein tuote on Raka-Kehikko. Raka-Kehikko on ohjelmiston elinkaaren-, versioinnin- ja muutostenhallintaan luotu järjestelmä. Yksi osa tätä järjestelmää on TQM (Task Queue Manager). TQM on yhteydessä Raka-Kehikon tietokantaan ja aloittaa prosessijonoja lukemansa tiedon perusteella.

TQM on nykymuodossaan kiinteästi sidottu Raka-Kehikkoon ja sen tietokantaan. Toimeksiantaja haluaa kuiteinkin erottaa TQM:n omaksi ohjelmistokokonaisuudekseen, jota RakaTuki voi jatkossa tarjota erillisenä tuotteena. Toimeksiantaja haluaa toteuttaa erottamisen luomalla uuden ohjelman toimimaan REST-rajapintana TQM ja Raka-Kehikon välillä.

Tässä raportissa kuvaan tämän REST-rajapinnan toteuttavan TQM-Controller-ohjelman suunnittelua ja toteutusta. Raportti on osa HAAGA-HELIA ammattikorkeakoulun Tietojenkäsittelyn koulutusohjelmaa varten tehtävää toteutustyyppistä opinnäytetyötä. TQM-Controller toteutetaan Java Spring-kehikolla palvelinohjelmana, joka ylläpitää omaa tietokantaa. Kuvaan raportissa myös REST-arkkitehtuurimallia, sekä TQM-Controllerissa käytettyjä teknologioita ja ratkaisuja.

Raportin lopussa käyn läpi huomioita REST-arkkitehtuurin toteuttamisesta käytännössä, sekä käytettyjen teknologioiden etuja TQM-Controller-projektissa. Kuvaan lisäksi ehdotuksia TQM-Controllerin jatkokehitystä varten.

Asiasanat
Ohjelmointi, Java, XML, MySQL, REST

HAAGA-HELIA
University of Applied Sciences

Abstract

14.5.2014

Degree programme in Information Technology

Author(s) Simo-Pekka Kerkelä	**Group or year of entry** 2011 Autumn
The title of thesis **Creating a REST interface using the Spring-framework** **Case: RakaTuki Ltd TQM-Controller**	**Number of report pages and attachment pages** 26 + 9
Advisor(s) Ismo Harjunmaa	

RakaTuki Ltd is a software development company based in Helsinki. Its primary product is Raka-Kehikko, which is a solution for handling the lifecycle, versioning and change management of software. One part of Raka-Kehikko is a program called TQM (Task Queue Manager). TQM connects to a database controlled by Raka-Kehikko and initiates process queues based on the data read.

In its current form, TQM is tightly coupled with Raka-Kehikko and the Raka-Kehikko database. The employer wants to decouple TQM from Raka-Kehikko into a separate program that can in the future be offered as a stand-alone product. To achieve this separation the employer wants to produce a new program to provide a REST interface between TQM and Raka-Kehikko.

In this report I describe the planning and implementation of TQM-Controller. This report is a part of an implementation-style thesis for the HAAGA-HELIA University of Applied Sciences degree in Information Technology. TQM-Controller provides a REST interface for interacting with TQM. TQM-Controller is implemented with the Java Spring-framework and handles a local database. I also describe the REST software-architecture, and various technologies and solutions used in TQM-Controller.

In the conclusion portion of the report I go over the various advantages of the technologies used, in addition to observations and conclusions on the REST model and its application. Finally, I provide suggestions for future development of TQM-Controller.

Key words
Programming, Java, XML, MySQL, REST

Sisällys

1 Johdanto ... 1
1.1 Projektin tarpeellisuus ... 2
2 Arkkitehtuurimallit ja ohjelmistokehikot ... 3
2.1 Kolmitasoarkkitehtuuri ... 3
2.2 REST-arkkitehtuurimalli ... 4
2.2.1 Arkkitehtuurimallin osat ... 4
2.2.2 Arkkitehtuurin ominaisuudet ... 5
2.2.3 Asiakas-palvelin ... 5
2.2.4 Tilattomuus ... 6
2.2.5 Välimuistin hyödyntäminen ... 6
2.2.6 Kerrostettu järjestelmä ... 6
2.2.7 Suoritettavan koodin lähettäminen ... 6
2.2.8 Yhtenäinen rajapinta ... 7
2.3 Spring-kehikko ... 7
2.4 iBatis-kirjasto ... 8
3 TQM-Controllerin suunnittelu ... 9
3.1 Toimintaympäristö ... 9
3.2 Tietokanta ... 9
3.3 REST arkkitehtuuri käytännössä ... 9
4 TQM-Controller toteutus ... 11
4.1 Ympäristön toteutus ... 11
4.2 Tietokanta ... 11
4.3 Domain-luokat ... 11
4.4 Web-taso ... 14
4.4.1 Jonojen käsittely ... 15
4.4.2 Jonopalvelimien käsittely ... 16
4.4.3 Tilausten käsittely ... 17

4.5 Service-taso 18
4.5.1 Jonopalvelimien käsittely 18
4.5.2 Jonojen käsittely 19
4.5.3 Jonojen ja jonopalvelimien yhteiskäyttö 19
4.5.4 Tilausten käsittely 19
4.6 Dao-taso 19
4.7 Testaaminen 20
5 Pohdinta .... **Virhe. Kirjanmerkkiä ei ole määritetty.**
5.1 Projektin haasteet ja onnistumiset 22
5.2 Arkkitehtuurimallit, kehikot ja kirjastot 22
5.3 Jatkokehittäminen 23
Lähteet 24
Liitteet 26

Käsitteitä

Eclipse

Eclipse on avoimen lähdekoodin ohjelmistokehitystyökalu. Se tukee useita ohjelmointikieliä, kuten esim. Java ja SQL. (Eclipse Foundation 2014.)

HTML

HTML eli Hypertext Markup Language on standardisoitu tapa esittää web-sivulla sisältöä (W3C 1999).

JAXB

JAXB eli Java Architecture for XML Binding on ohjelmisto ja Java-kirjasto mikä mahdollistaa Java-luokkien luomisen XML-tiedostoista (Ort, Mehta 2003).

JSON

JSON eli JavaScript Object Notation on standardoitu tapa esittää jäsenneltyä tietoa, perustuen JavaScript-kielen syntaksiin (Ecma International 2013, 4).

JSP

JSP eli JavaServer Pages on Javaan perustuva menetelmä luoda HTML- ja XML muotoisia websivuja (Oracle 2014).

MySQL

MySQL on laajalti käytössä oleva relaatiotietokanta (MySQL 2014).

XML

XML on standardisoitu tapa esittää jäsenneltyä tietoa (W3C 2008).

1 Johdanto

Tämän raportin on tarkoitus esitellä HAAGA-HELIA ammattikorkeakoulun Tietojenkäsittelyn Koulutusohjelman (TIKO) opiskelijan koulutusaikana kerääntynyttä tietämystä ja ammattitaitoa. Opinnäytetyötä pidetään opiskelijan viimeisenä tehtävänä ennen Tradenomitutkinnon suorittamista.

RakaTuki Oy on sovelluskehitysyritys Helsingissä, joka on erikoistunut tuotteenhallinnan järjestelmien määrittely-, suunnittelu- ja toteutustyöhön. Yrityksen tärkein tuote on Raka-Kehikko. Raka-Kehikko on järjestelmä, joka yhdistää sovellusten elinkaaren-, version- ja muutostenhallinnan yhteen toimivaksi kokonaisuudeksi. Oleellinen osa Raka-Kehikkoa on sen käyttämä tietokanta, joka sisältää Kehikolla hallittavien sovellusten ja asennuspakettien versiotiedot. Kun uusi asennuspaketti kirjautuu versiokatalogiin, Kehikon tietokantaan kirjautuu myös asennuspyyntö. Asennuspyyntöihin reagoi Task Queue Manager -palvelu (TQM).

TQM on ohjelma, joka lukee Kehikon tietokannasta asennuspyyntöjä ja käynnistää pyyntöjen mukaisia ohjelmia, kuten esimerkiksi Java-pakettien asennuksia. Se järjestää ensin pyynnöt useisiin rinnakkaisiin jonoihin. Samassa jonossa olevat prosessit käynnistyvät peräkkäin yksi kerrallaan odottaen edellisen prosessin päättymistä ennen seuraavan käynnistämistä. TQM ja Raka-Kehikon yhteistoiminta projektin aloitusvaiheessa on kuvattu liitteessä 5.

Opinnäytetyöprojektin tavoitteena on suunnitella ja toteuttaa REST-rajapinta ja eriytetty tietokanta RakaTuki Oy:n TQM-sovellukseen. Tuloksena on erillinen TQM-Controller-ohjelma, joka vastaanottaa REST-pyyntöjä ja reagoi niihin määritellyllä tavalla. Tavoitetila on kuvattu liitteessä 6.

TQM-Controllerin tulee käsitellä seuraavat REST-pyynnöt:

- Jonopalvelimien luettelointi (sekä yksittäisen että kaikkien)
- Jonopalvelimen päivittäminen
- Tilausten luettelointi (sekä yksittäisen että kaikkien)
- Tilauksen tekeminen

– Tilauksen päivittäminen
– Jonojen luettelointi (sekä yksittäisen että kaikkien)

Lisäksi tavoitteena on suorittaa HAAGA-HELIAN Tietojenkäsittelyn Koulutusohjelman mukainen opinnäytetyö.

1.1 Projektin tarpeellisuus

TQM-ohjelma kutsuu nykytilassa Kehikon tietokantaa useita kertoja minuutissa, mikä rasittaa tietokantaa tarpeettomasti ja aiheuttaa ylimääräisiä kuluja. Projektin tuloksena toteutettu TQM-Controller vähentäisi huomattavasti Kehikon tietokannan rasitusta, sillä TQM-Controller ylläpitäisi omaa tietokantaansa, johon se päivittäisi tarpeellisia tietoja, sekä päivittäisi niitä REST-rajapinnan kautta tulevien pyyntöjen perusteella. TQM-Controller myös mahdollistaa jatkokehityksellä koko TQM-ohjelman irrottamisen Raka Kehikko-ympäristöstä omaksi tuotteekseen, mitä RakaTuki Oy voi tulevaisuudessa tarjota omana tuotteenaan.

Minulle opinnäytetyön tekijänä projektista kertyy käytännön kokemusta ohjelmistokehityksestä työelämässä. Lisäksi saan suoritettua Tietojenkäsittelyn Koulutusohjelmaan kuuluvan oppimista ja opitun soveltamista esittelevän opinnäytetyön.

2 Arkkitehtuurimallit ja ohjelmistokehikot

Ohjelmistokehitystä varten on kehittynyt useita arkkitehtuurimalleja ja ohjelmistokehikkoja tyypillisten ja usein toistuvien ongelmien ratkaisuiksi. Arkkitehtuurimallit ovat ohjeita ja säännöksiä ohjelmien rakenteen suunnittelua varten. Niiden etu on edesauttaa ohjelmiston kehitettävyyttä, testattavuutta, turvallisuutta ja useita muita kriittisiä osa-alueita ja ne kehittyvät usein työelämässä todettujen parhaiden käytäntöjen ympärille. Arkkitehtuurimallit ovat usein toteutuskielestä ja ympäristöstä riippumattomia, eivätkä yleensä ota kantaa tarkempiin kirjastoratkaisuihin.

Ohjelmistokehikot mahdollistavat kehittyneempien ohjelmien nopeamman kehittämisen, sillä kehikot tarjoavat valtaosan pohjatyöstä valmiina, eikä ohjelmoinnin tarvitse alkaa tyhjästä. Esimerkiksi useimmat Web-ohjelmoinnin kehikot sisältävät valmiit luokat ja metodit HTTP-pyyntöjen käsittelemiseen, tietokantoihin yhdistämiseen jne. Ohjelmistokehikot ovat usein hyvin muokattavissa, kuten esim. Spring-kehikko. On tyypillistä, että ohjelmistokehikot toteuttavat jonkin arkkitehtuurimallin, kuten esim. kolmitaso-arkkitehtuurin.

2.1 Kolmitasoarkkitehtuuri

Kolmitasoarkkitehtuurissa ohjelman toiminnallisuus on jaettu kolmeen eri tasoon, joista jokainen käyttää itseään alempaa tasoa oman tehtävänsä suorittamiseen. Päällimmäinen tavoite arkkitehtuurimallilla on eriyttää ohjelman ulkoinen rajapinta (esim. ulkosivu, rajapinta), logiikkataso eli tiedon tarkempi käsittely, sekä tiedon fyysinen tallennus (esim. tietokannat). Tässä projektissa käytän tasoista termejä Web-taso, Service-taso ja Dao-taso.

Web-taso vastaanottaa kyselyt ja lähettää ne käsiteltäväksi kyselylle tarkoitettuun Service-tason luokkaan. Tämä on ainoa taso joka on ulospäin avoin, eli sovelluksen ulkoinen rajapinta. Käytännössä Java-sovelluksessa Web-taso tarkoittaa ulkoasua, eli HTML ja JSP-sivuja, sekä niitä ohjaavia ”Controller”-luokkia.

Service-taso käsittelee tietoa käsitteinä. Service-luokat päättävät mitä tiedolle tehdään, mitä tietoa palautetaan. Service-taso käyttää tiedon muokkaamiseen, tallentamiseen tai poistamiseen Dao-tasoa. Java-ohjelmassa Service-tasoon kuuluu ”Service” ja ”Manager” luokat, sekä niiden käyttämät tietoa kuvaavat Java-luokat, kuten esimerkiksi ”Ticket” tai ”Queue”.

Dao-taso käsittää ohjelman rajapinnan sisäiseen tallennustilaan, eli esimerkiksi tietokantaan. Dao-taso saa nimensä lyhenteestä DAO (Data Access Object). Kullekin tallennettavalle tietotyypille luodaan oma Dao-luokkansa, mikä hoitaa kyseisen tietotyypin tallentamisen, hakemisen, päivittämisen ja poistamisen.

2.2 REST-arkkitehtuurimalli

REST on lyhenne käsitteelle **Representational State Transfer**. Kyseessä on arkkitehtuurimalli ohjelmointirajapinnoille. Se on erityisesti web-palveluiden kehityksessä käytetty malli ja on vaihtoehto muille hajautetun järjestelmän arkkitehtuurimalleille.

Termi REST on alun perin määritelty vuonna 2000 Roy Fieldingin tohtorin väitöskirjassa. Väitöskirjan tarkoituksena oli kartoittaa Web-arkkitehtuurin nykytilaa ja ehdottaa yhtenäistä uutta arkkitehtuurimallia, mikä sopisi yhteen samaan aikaan kehitettyjen muiden Web-standardien kanssa. (Fielding 2000, 116-117.)

2.2.1 Arkkitehtuurimallin osat

REST -arkkitehtuuri perustuu kolmeen keskeiseen osaan ja niiden välisten suhteiden rajauksiin. Osat ovat (Fielding 2000, 123.):

- Komponentit (Components)
- Liittimet (Connectors)
- Data

Arkkitehtuurimalli ei ota tekijöiden implementointiin kantaa, kunhan ne toteuttavat määritellyt vaateet.

Komponentti on ohjelmiston yksikkö joka mahdollistaa rajapintansa avulla tiedon muutoksen. Esimerkkejä komponenteista ovat mm. lähtöpalvelimet ja verkkoselaimet. (Fielding 2000, 129.)

Liitin on mekanismi joka koordinoi komponenttien välistä yhteistoimintaa ja viestintää. Käytännön esimerkkeinä mm. palvelin-, työpöytä- ja mobiilisovellukset sekä verkkoselainten välimuistit. (Fielding 2000, 127.)

Data on informaation yksikkö jota siirretään yhden tai useamman liittimen välityksellä komponentilta toiselle. REST arkkitehtuurissa dataa käsitellään termillä resurssi. Esimerkkejä ovat HTML-sivut, ääni- ja kuvatiedostot, paikalliset säätiedot jne. (Fielding 2000, 125.)

2.2.2 Arkkitehtuurin ominaisuudet

REST -arkkitehtuurin tärkeimmät ominaisuudet saavutetaan valvomalla rajauksia tekijöiden välisessä vuorovaikutuksessa. Rajausten toteutuminen tuo ohjelmassa esiin useita haluttuja ominaisuuksia, kuten korkea suorituskyky, yksinkertaisuus, muokattavuus ja luotettavuus. (Fielding 2000, 120.)

Rajaukset ovat:

- Asiakas-palvelin
- Tilattomuus
- Välimuistin hyödyntäminen
- Kerrostettu järjestelmä
- Suoritettavan koodin lähettäminen
- Yhtenäinen rajapinta

2.2.3 Asiakas-palvelin

Asiakkaat ja palvelimet on erotettu yhtenäisellä rajapinnalla. Asiakas-sovellukset eivät esim. tallenna dataa tietokantaan, vaan pyytävät sitä palvelimelta. Palvelin ei sen sijaan puutu asiakas-sovelluksen ulkoasuun tai käyttäjäasetuksiin. Asiakas-palvelin rajaus mahdollistaa asiakas-sovelluksissa usean alustan tukemisen vaivatta. Palvelimet puoles-

taan pysyvät yksinkertaisempina ja helpommin laajennettavina. Asiakas- ja palvelinsovelluksia voidaan myös kehittää toisistaan irrallisina, olettaen että niiden käyttämää rajapintaa ei muuteta. (Fielding 2000, 121.)

2.2.4 Tilattomuus

Asiakkaan ja palvelimen rajauksiin kuuluu myös ohjelman tilattomuus: palvelin ei tallenna asiakkaista tietoa kyselyiden välillä. Jokainen asiakkaan lähettämä kysely sisältää kaiken palvelimen vastaukseen tarvitseman tiedon. Niin sanottuja sessiotietoja ei tallenneta palvelimelle, vaan esim. kirjautumistiedot tulee lähettää edelleen siihen tarkoitetulle palvelulle, kuten esim. tietokantaan. (Fielding 2000, 121.)

2.2.5 Välimuistin hyödyntäminen

Ohjelmien nopeuttamiseksi välimuistia hyödyntäen on palvelimen lähettämien vastausten määriteltävä, joko implisiittisesti tai eksplisiittisesti, onko lähetetty tieto mahdollista varastoida välimuistiin. Välimuistin tehokas hyödyntäminen edesauttaa skaalautuvuutta ja suorituskykyä. (Fielding 2000, 121.)

2.2.6 Kerrostettu järjestelmä

Asiakasohjelma ei näe onko se yhteydessä suoraan päätepalvelimeen vai esim. välipalvelimeen. Palvelimet voidaan järjestää kerroksittain hierarkiaan, jossa kukin kerros näkee vain itsensä kanssa kommunikoivat kerrokset. Järjestelmän monimutkaisuus pysyy hallinnassa ja esim. vanhat tai harvoin käytetyt osat järjestelmää on mahdollista eristää omiin kerroksiinsa. Välipalvelimilla edesautetaan skaalautuvuutta ja järjestelmän vakautta. (Fielding 2000, 122.)

2.2.7 Suoritettavan koodin lähettäminen

Palvelin voi lähettää asiakasohjelmalle ohjelmakoodia tarkoituksena tilapäisesti parantaa tai täydentää asiakasohjelman suoritusta. Tyypillisin esimerkki on web-sivujen Javascript-ohjelmien lähettäminen. Suoritettavan koodin lähettäminen (engl. Code on De-

mand) on ainoa REST -arkkitehtuurin rajauksista joka on valinnanvarainen. (Fielding 2000, 123.)

2.2.8 Yhtenäinen rajapinta

REST -arkkitehtuurin oleellisimpia osia on yhtenäisen rajapinnan rajaus. Asiakasohjelmat ja palvelin käyttävät kyselyihin ja vastauksiin yhteistä, ennalta määritettyä tietomuotoa. (Fielding 2000, 122.)

Asiakasohjelma lähettää kyselyssä aina tarvittaessa pyytämänsä yksittäisen resurssin yksilöllisen tunnuksen. Web-pohjaisissa REST järjestelmissä tämä voi tapahtua esimerkiksi URI:n välityksellä. Palvelin ei palauta suoraan tietokantaansa, vaan muuttaa ensin halutun tiedon sisäisesti yhtenäisen rajapinnan mukaisesti sopivaan muotoon. Näitä muotoja ovat esim. HTML, XML, tai JSON. (Fielding 2000, 124-125.)

Asiakas voi myös niin ikään manipuloida palvelimen hallitsemia resursseja lähettämällä palvelimelle rajapinnan mukaisessa muodossa kuvauksia muutoksista. Palvelin käsittelee pyynnön kuvauksen mukaisesti, mahdollisesti muokaten tai poistaen tietoa tietokannastaan, ja lähettää lopuksi asiakkaalle kuvauksen käsittelyn lopputuloksesta, tai virhetilanteessa virheilmoituksen. (Fielding 2000, 126.)

Kyselyiden tulee aina sisältää kaiken tiedon prosessoimiseen vaadittavan tiedon. Esim. XML tai JSON-muodossa olevien viestien tulee sisältää tarvittava metadata, jotta palvelin osaa reagoida oikein tietoa parsiessaan.

2.3 Spring-kehikko

Spring-kehikko on erityisesti Web-kehittämistä varten luotu laaja ohjelmistokehikko. Se mahdollistaa monipuolisen palvelinpuolen ohjelmoinnin ja yksinkertaistaa huomattavasti useita Web-kehittämisen osa-alueita, kuten tietokantayhteyksien luomista, käyttäjienhallintaa, käyttöturvallisuutta, testaamista jne.

Tärkeimpiä Spring-kehikon ominaisuuksia ovat **Riippuvuusinjektio**, **Model-View-Controller-malli (MVC),** sekä laaja tietokantatuki (Spring Documentation 2014).

Riippuvuusinjektio (eng. Dependency Injection) tarkoittaa arkkitehtuurimallia, missä Java-luokkien attribuuttien (riippuvuuksien) esittely ja alustaminen toteutetaan luokan ulkopuolella, tyypillisesti esim. XML-tiedostossa. Riippuvuusinjektio mahdollistaa helpomman yksikkötestaamisen ja järjestelmän asetusten määrittämisen Java-luokkien ulkopuolelta. (Spring Documentation 2014.)

MVC-malli on suosittu arkkitehtuurimalli, missä tiedon sisäinen mallinnus esitystapa ja käsittely ovat eriytetty. Tieto mallinnetaan Java-luokkana, jota käsitellään Controller-luokassa ja lopuksi esitetään esim. HTML-sivulla. MVC-malli on esimerkki kolmitasoarkkitehtuurista. (Spring Documentation 2014.)

Spring-kehikko toimi opinnäytetyöprojektin perustana, tarjoten ratkaisut esim. HTTP-kyselyiden käsittelyyn, tietokantaan yhdistämiseen sekä ohjelmakoodin jäsentelyyn.

2.4 iBatis-kirjasto

IBatis-kirjasto on tietokantarajapinnan tarjoava kirjasto. Sen avulla SQL-haut ja tietotyypit on mahdollista kuvata erillisessä XML-tiedostossa. Hakuja on mahdollista nimetä ja kutsua Java-koodista. IBatis mahdollistaa hakulogiikan eriyttämisen täysin ohjelmakoodista. Opinnäyteprojektissa iBatis helpotti suuresti valmiin tietokannan kanssa toimimista, sillä TQM-ohjelman käyttämät SQL-lauseet oli valmiiksi kuvattu XML-tiedostossa. Kolmitasoarkkitehtuurissa iBatis asettuu alimmalle eli Dao-tasolle. (iBatis.)

3 TQM-Controllerin suunnittelu

Projekti alkoi ohjelmiston suunnittelulla yhdessä toimeksiantajan kanssa. Koska TQM-Controller tulisi toimimaan olemassa olevan TQM-ohjelmiston kanssa, oli tärkeää että yhteiset rajapinnat olivat yhtenäisiä. Toimeksiantaja toimitti tietokantakuvauksia, esimerkkikyselyitä sekä käsittelysääntöjä. Suunnittelussa pyrin valitsemaan arkkitehtuurimalleja ja teknologioita jotka olivat sekä toimeksiantajalle että minulle tuttuja, sekä myös yleisesti käytettyjä ja luotettavia.

3.1 Toimintaympäristö

TQM-ohjelma on suunniteltu toimimaan Windows-ympäristössä koska RakaTuen asiakaskunnasta valtaosa käyttävää Windowsia, joten myös TQM-Controller keskittyi yksinomaan Windows-ympäristössä toimimiseen. Toteutuskielen ansiosta toimiminen muissakin ympäristöissä on pienellä lisäkehityksellä kuitenkin mahdollista.

3.2 Tietokanta

TQM-Controller käyttää omaa tietokantaansa, johon se päivittää kyselyiden pyytämät muutokset ja josta se palauttaa vastaukset kyselyiden pyytämiä resursseja. Tietokannan rakenne oli ennalta määritelty ja se vastasi olemassa olevan TQM-ohjelman käyttämää tietokantaa. Tietokannan osalta suunnittelutyön tarve jäi siksi vähäiseksi. Tietokannan taulujen kuvaukset ovat liitteessä 1.

3.3 REST arkkitehtuuri käytännössä

REST-arkkitehtuurimallin mukaisesti TQM-Controller toimii palvelimella komponenttina ja vastaa asiakasohjelmien kyselyihin määritelmien mukaisella tavalla. Kyselyt lähetetään ja vastaanotetaan XML-muotoisena. XML on yleisimpiä tiedonsiirtoformaatteja ja projektissa käytettävä Spring-kehikko osaa käsitellä XML-muotoista tietoa suoraan. Asiakasohjelmaa ei tämän opinnäytetyöprojektin puitteissa toteuteta. Tämä on mahdollista REST arkkitehtuurin asiakas – palvelin rajauksen ansiosta. Asiakasohjelmana TQM-Controllerille tulee tulevaisuudessa toimimaan Raka Kehikko.

REST-rajapinta käsittelee seuraavaksi kuvattuja tietoja (REST resursseja):

- **Jono** (Queue) pitää sisällään listaa tilauksista, joita se käsittelee järjestyksessä.
- **Tilaus** (Ticket) pitää sisällään yhden asennus/ajopyynnön, minkä TQM:n halutaan suorittaa.
- **Jonopalvelin** (Server) pitää sisällään kuvauksen yksittäisestä TQM palvelininstanssista.

Toimeksiantaja toimitti resursseista tietokantakuvaukset ja XML-skeemat.
TQM-Controller tukee seuraavanlaisia kyselyitä:

- Listaus jonoista
- Yksittäisen jonon kysely
- Listaus tilauksista
- Yksittäisen tilauksen kysely
- Tilauksen lähettäminen
- Listaus jonopalvelimista
- Yksittäisen jonopalvelimen tiedot
- Jonopalvelimen tietojen päivitys

3.4 Testaaminen

Projektissa suunniteltiin alustavia automaattisia testejä, mutta toimeksiantajan muiden kiireiden vuoksi erillistä testaussuunnitelmaa ei luotu. TQM-Controllerin testaus on hyvin kiinteästi sidottu tulevien asiakasohjelmien toimintaan, sekä TQM-ohjelman toimintaan ja ympäristöön, joten ilman toimeksiantajan tarvittavaa panosta testien suunnittelu osoittautui haasteelliseksi. Projektin toteutusvaiheessa tämä testausvaiheen suunnittelun vähäisyys johti manuaaliseen testaamiseen.

4 TQM-Controller toteutus

TQM-Controller on toteutettu palvelimella toimivana Java-ohjelmana. Toteutukseen on käytetty useita valmiita kirjastoja ja kehikkoja. Tässä osiossa on kuvattu vaihe vaiheelta TQM-Controllerin toteutus. TQM-Controllerin arkkitehtuuri on kuvattu liitteessä 8.

4.1 Ympäristön toteutus

Projektin toteutus alkoi kehitysympäristön pystyttämisestä toimeksiantajan toimittamalle tietokoneelle. Koneelle asennettiin Windows 7 Professional-käyttöjärjestelmä, MySQL-tietokanta, Eclipse-kehitystyökalu, Subversion-versionhallintatyökalu ja TomCat 7-kehityspalvelin. Eclipseen asennettiin lisäksi Spring-sovellusten kehittämiseen tarkoitettu lisäosa Spring Tool Suite (STS). REST-rajapinnan testaamiseen ladattiin RestClient 3.1. Java-luokkien luomiseen XML-tiedostoista ladattiin JAXB-ohjelma. Ympäristön pystyttäminen onnistui ilman komplikaatioita ja toimi koko projektin ajan moitteetta.

4.2 Tietokanta

Projektissa käytettiin MySQL-tietokantaa. Alkuperäinen TQM-ohjelma käytti IBM DB2-tietokantaa, joten tietokannan pystyttämiseen tehtyjä luomistiedostoja jouduttiin muokkaamaan. Tietokannan kuvaus löytyy liitteestä 1. SQL-lauseet joilla tietokanta pystytettiin löytyvät liitteestä 2.

4.3 Domain-luokat

Domain luokat sisältävät ohjelman käsittelemän tiedon sisäiset kuvaukset ja parametrit, mutta eivät toimintalogiikkaa. Valtaosa Domain-luokista generoitiin JAXB-ohjelmalla suoraan toimeksiantajan toimittamista XML-tiedostoista.

JAXB-ohjelmalla generoituihin luokkiin lisätään automaattisesti annotaatiot, joiden avulla Spring-kehikko pystyy automaattisesti muuttamaan kyseisestä luokasta luodun ilmentymän takaisin XML-muotoon. Lisäksi luokassa on annotaatiot, jotka kuvaavat

kenttien pakollisuutta. Näiden avulla Spring-kehikko kykenee automaattisesti validoimaan käsittelemänsä ilmentymän.

Taulukko 1. JAXB:lla generoidut Java-luokat

Luokan nimi	**Parametri**	**Tietotyyppi**
Server	serverId	int
	serverName	String
	serverStatus	int
ServerList	server	List<Server>
TqmcMessage	messageId	String
	message	String
TqmcMessages	tqmcMessage	List<TqmcMessage>
TqmcOrderRequest	customer	String
	orderId	String
	description	String
	queueName	String
	queueMaxTime	int
TqmcOrderResponse	tqmcRequestResult	TqmcRequestResult
	customer	String
	orderId	String
	orderNumber	int
	ticketId	int
TqmcParameter	parameterName	String
	parameterValue	String
TqmcRequestResult	tqmcResult	TqmcResult
	tqmcMessages	TqmcMessages
TqmcResult	resultCode	String
	explanation	String
TqmQueue	queueId	int
	serverId	int
	serverName	String
	queueServerStatus	int
	queueName	String
TqmQueueList	tqmQueue	List<TqmQueue>
TqmTicket	customer	String
	orderId	String
	orderNumber	int
	queueName	String
	ticketId	int
	orderStatus	int
	message	String

JAXB:lla generoitujen Domain-luokkien lisäksi TQM-Controller käyttää neljää erikseen luotua Java-luokkaa sisäisesti iBatis-tietokantarajapinnan yhteydessä.

Taulukko 2. iBatis-tietokantarajapintaa varten luodut luokat

Luokan nimi	Parametri	Tietotyyppi
QueueServer	queueId	int
	serverId	int
	serverName	String
	queueServerStatus	int
	queueName	String
Queue	queueId	int
	ticketStatus	int
	queueName	String
Parameter	objectId	int
	objectType	int
	parameterNumber	int
	parameterName	String
	parameterValue	String
Ticket	ticketId	int
	orderId	String
	orderNumber	int
	initTimeStamp	String
	startTimeStamp	String
	message	String
	logfile	String
	description	String
	queueName	String
	queueId	int
	queueStatus	int
	queueStartTimeStamp	String
	queueStopTimeStamp	queueMaxTime
	queuePid	int

4.4 Web-taso

Kolmitasoarkkitehtuurin korkein taso on sovelluksessa Web-taso. Web-taso koostuu Controller-luokista jotka vastaanottavat kyselyitä eri muodoissa ja vastaavat niihin kyselyn perusteella. Controller-luokat on annotoitu Spring-kehikon tarjoamilla annotaatioilla, joilla pystytään yksilöimään mihin metodiin kukin kysely päätyy. Annotaatiolla voidaan myös automaattisesti parsia vastaanotettu XML-tiedosto Domain-luokan ilmentymäksi, sekä vastaavasti palauttaa kyselyn lähettäjälle XML-muotoinen vastaus. UML-kuvaukset Web-tason luokista löytyvät liitteestä 3.

4.4.1 Jonojen käsittely

QueueController vastaanottaa pyynnöt liittyen jonoihin. Se pitää sisällään ilmentymän QueueServerManager-luokasta.

Tuetut kyselyt ovat:

– yksittäisen jonon tiedot
– kaikkien jonojen tiedot
– yksittäisen jonon jonopalvelimen tila.

Kaikkien jonojen tiedot saadaan lähettämällä `GET`-muotoinen kysely osoitteeseen `/tqmqueues`. Metodi, joka käsittelee kyselyn, on nimeltään `getTqmQueues`. Metodi hakee QueueServerManager-ilmentymää hyödyntäen tietokannasta listaan jonot ja muuttaa listan TqmQueueList-luokan ilmentymäksi. Lopuksi kysely palauttaa TqmQueueList-ilmentymän, minkä Spring-kehikko jälleen muuttaa automaattisesti XML-muotoon ennen eteenpäin lähettämistä.

Yksittäisen jonon tiedot saadaan lähettämällä `GET`-muotoinen kysely osoitteeseen `/tqmqueues/X`, jossa `X` on halutun jonon yksilöllinen tunnusnumero, eli ID. Kysely käsitellään t`qmQueue`-metodissa. Metodi hakee QueueServerManager-ilmentymää hyödyntäen tietokannasta listaan jonot ja etsii jonoista halutun ID:n omaavan jonon. Mikäli jono löytyy, palauttaa metodi jonon TqmQueue-luokan ilmentymän. Muussa tapauksessa metodi palauttaa tyhjän (`null`). TqmQueue-luokka on JAXB:lla generoitu, minkä ansiosta Spring-kehikko pystyy muuttamaan sen automaattisesti XML-muotoon ennen asiakkaalle palautusta.

Yksittäisen jonon jonopalvelimen tila saadaan lähettämällä `GET`-muotoinen kysely osoitteeseen `/tqmqueues/X/serverstatus`, missä X on halutun jonon ID. Metodin nimi on `queueServerStatus` ja se toimii muuten samoin kuin yksittäisen jonon tietoja hakeva metodi, mutta palauttaa vain jonon jonopalvelimen tilan. Tila palautetaan kokonaislukuna.

4.4.2 Jonopalvelimien käsittely

ServerController vastaanottaa pyynnöt, jotka käsittelevät jonopalvelimia. Se sisältää ilmentymän ServerService-luokasta.

Tuetut kyselyt ovat:

- Kaikkien jonopalvelimien tiedot
- Yksittäisen jonopalvelimen tiedot
- Yksittäisen jonopalvelimen tietojen päivitys

Kaikkien jonopalvelimien tiedot saadaan lähettämällä `GET`-muotoinen kysely osoitteeseen `/servers`. Kyselyn käsittelevän metodin nimi `on getServers`. Metodi hakee ServerService-luokan ilmentymän avulla listaan kaikki palvelimet ja luo niistä ServerList-luokan ilmentymän. Lopuksi metodi palautta ServerList-ilmentymän, minkä Spring-kehikko muuttaa XML-muotoiseksi ja lähettää takaisin kyselyn lähettäjälle.

Yksittäisen jonopalvelimen tiedot saadaan lähettämällä `GET`-muotoinen kysely osoitteeseen `/servers/X`, jossa `X` on halutun jonopalvelimen yksilöllinen tunnus, eli ID. Kyselyn käsittelevän metodin nimi on `getServer`. Metodi hakee ID:n perusteella ServerService-luokan ilmentymän avulla yksittäisen jonopalvelimen tiedot ja palauttaa kyseisen Server-luokan ilmentymän.

Yksittäisen jonopalvelimen tietojen muutos tehdään lähettämällä `POST`-muotoinen kysely osoitteeseen `/servers/update`. `POST`-muotoinen kysely lähettää mukanaan XML-muodossa muutettavat jonopalvelimen tiedot. Tietojen muutoksen käsittelee metodi nimeltä `updateServer`. Spring-kehikko parsii XML-tiedoston ja luo sen perusteella Server-luokan ilmentymän, jonka metodi lähettää ServerService-luokan ilmentymälle. Metodi luo TqmcRequestResult-luokan ilmentymän ja, riippuen onnistuiko jonopalvelimen tietojen muutos vai ei, asettaa siihen joko tiedon onnistumisesta tai virheilmoituksen. Lopuksi TqmcRequestResult lähetetään kyselyn lähettäjälle. Spring-kehikko muuttaa palautettavan tiedon XML-muotoiseksi.

4.4.3 Tilausten käsittely

TicketController vastaanottaa pyynnöt, jotka liittyvät tilauksiin. Se sisältää ilmentymän TicketManager- luokasta.

Tuetut kyselyt ovat:

- Kaikkien tilausten tiedot
- Yksittäisen tilauksen tila
- Uuden tilauksen luominen
- Yksittäisen tilauksen tilan päivittäminen

Kaikkien tilausten tiedot saadaan lähettämällä `GET`-tyyppinen kysely osoitteeseen `/tqmcorders`. Tilausten tietojen hakemisen käsittelee metodi nimeltä `list-Tickets`. Metodi hakee TicketManager-luokan ilmentymän avulla kaikki tilaukset, asettaa ne TqmTicketList-luokan ilmentymään ja palauttaa tämän ilmentymän. Spring-kehikko muuttaa palautettavan TqmTicketList:n XML-muotoon.

Yksittäisen tilauksen tila saadaan lähettämällä `GET`-tyyppinen kysely osoitteeseen `/tqmcorderstatus/X`, missä `X` on halutun tilauksen yksilöllinen tunnus, eli ID. Tilauksen tilan haun käsittelee metodi nimeltä `orderstatus`. Metodi luo ilmentymän TqmcRequestResult-luokasta, hakee TicketManager-luokan ilmentymällä ID:n perusteella halutun tilauksen ja asettaa TqmcRequestResult:iin viestinä tilauksen tilan. Tila on kokonaisluku. Mikäli halutulla ID:llä ei löydy tilausta, asetetaan TqmcRequestResult:iin viestinä virheilmoitus.

Uuden tilauksen luominen saadaan lähettämällä `POST`-tyyppinen kysely osoitteeseen `/tqmcorder`. `POST`-muotoinen kysely lähettää mukanaan XML-muodossa uuden tilauksen tiedot. Uuden tilauksen luomisen käsittelee metodi nimeltä `receiveOrder`. Spring-kehikko parsii vastaanotetusta XML-tiedostosta TqmcOrderRequest-luokan ilmentymän. Käyttäen `isValid`-metodia varmennetaan ensin, että tilaus ei ole puutteellinen. Seuraavaksi tilaus yritetään tallentaa. Mikäli tallennus onnistuu, asetetaan TqmcRequestResult:iin viesti onnistumisesta. Mikäli tallennus epäonnistuu, asetetaan

viestiin virheilmoitus. Lopuksi metodi TqmcRequestResult-luokan ilmentymän, minkä Spring-kehikkoa muuttaa XML-muotoon.

Tilauksen tilan päivitys tapahtuu lähettämällä `PUT`-muotoinen pyyntö osoitteeseen `/updateorderstatus/X/Y`, missä `X` on halutun tilauksen yksilöllinen tunnus, eli ID ja `Y` on haluttu tila kokonaislukuna. Tilauksen tilan päivityksen käsittelee metodi nimeltä `updateOrderStatus`. Metodi hakee halutun tilauksen TicketManager-luokan ilmentymällä, asettaa tilaukselle uuden tilan ja lopuksi päivittää tilauksen. Riippuen päivityksen onnistumisesta, asetetaan TqmcRequestResult-luokan ilmentymälle joko viesti onnistumisesta, tai virheilmoitus. Metodi palauttaa TqmcRequestResultin, minkä Spring-kehikko muuttaa XML-muotoiseksi vastaukseksi.

4.5 Service-taso

Kolmitasoarkkitehtuurin keskimmäinen taso on Service-taso. Service-taso koostuu Service- ja Manager-luokista, jotka käyttävät Domain-luokkia ja Dao-luokkia tiedon prosessointiin ja tallentamiseen. Service-taso vastaa ohjelmiston päälogiikasta, kuten laskutoimista, tiedon rajaamisesta jne.

Sovelluksessa käytin Spring-kehikossa yleistä käytäntöä luoda Service-luokat toteuttamaan erillistä Java-rajapintaluokkaa. Java-rajapintaluokkien käyttö mahdollistaa helpomman yksikkötestaamisen, sekä vaihtoehtoisten toteutusluokkien luonnin tulevaisuudessa. Yleisen jatkokehitettävyyden kannalta rajapintaluokkien käyttö on usein järkevää. Service-tason luokkien UML-kaaviot löytyvät liitteestä 4.

4.5.1 Jonopalvelimien käsittely

Jonopalvelimien käsittelyyn keskittyy ServerService-rajapinta ja sen toteuttava SimpleServerService-luokka. SimpleServerService sisältää ilmentymän ServerDao-rajapinnan toteuttavasta luokasta.

Jonopalvelimien käsittelyssä ei tarvittu suurempaa logiikkaa, joten metodit palauttavat Dao-luokan ilmentymän avulla tietokannasta jonopalvelimien tietoja. Tiedon siirtoon käytetään Server-luokan ilmentymiä.

4.5.2 Jonojen käsittely

Jonojen käsittelystä vastaa QueueService-rajapintaluokan toteuttava SimpleQueueService. Luokka käyttää QueueDao-luokan ilmentymää Queue-luokan ilmentymien tallentamiseen. Luokka on kaikista Service-tason luokista yksinkertaisin, sisältäen vain kaksi metodia, yksittäisen jonon tallentamisen ja yksittäisen jonon hakemisen ID:n avulla.

4.5.3 Jonojen ja jonopalvelimien yhteiskäyttö

QueueServerManager-rajapintaluokka ja SimpleQueueServerManager-luokka käsittelevät jonoja ja jonopalvelimia yhdessä. Käsittelylogiikka on hyvin yksinkertaista ja metodit palauttavat joko yksittäisiä QueueServer-luokan ilmentymiä, tai listoja näistä ilmentymistä. Hakuihin käytetään QueueServerDao-rajapinnan toteuttavaa luokkaa. Jonojen ja jonopalvelimien yhteiskäyttö on yleistä. QueueServerManager mahdollistaa molempien tietojen haun kerralla, vähentäen näin kokonaisuudessa tarvittavia tietokantahakuja.

4.5.4 Tilausten käsittely

Tilausten käsittelystä vastaa TicketManager-rajapintaluokan toteuttava SimpleTicketManager. Yksittäisten tilausten sekä tilauslistojen haun lisäksi TicketManager-rajapinnan toteuttavat luokat sisältävät myös tilauksen luontimetodin. Metodi vastaanottaa listan Parameter-luokan ilmentymistä, sekä Ticket-luokan ilmentymän ja tallentaa nämä TicketDao-luokan ilmentymällä tietokantaan.

4.6 Dao-taso

Dao-taso on ohjelman liittymätaso tietokantaan. TQM-Controllerissa Dao-taso ei kuitenkaan käytä suoraan esim. SQL-lauseita, vaan kyselyiden tekeminen tietokantaan tehdään iBatis-kirjastoa käyttäen. SQL-lauseet ovat määritelty erillisessä XML-tiedostossa.

XML-tiedostoon on kuvattu kaikki tarvittavat tietotyypit, sekä niihin kohdistuvat kyselyt. Kyselyt on nimetty yksilöllisin tunnuksin, joita käyttämällä XML-tiedostossa kuvattuja SQL-lauseita voidaan kutsua Java-koodista. Jotta iBatis-kirjastoa voi käyttää, on Dao-luokan perittävä SqlMapClientDaoSupport-luokasta, mikä on iBatis-kirjastoon kuuluva luokka. Kaikki metodit käyttävät hyväkseen SqlMapClientTemplate-luokan ilmentymää, minkä kautta iBatis-kirjastoa käytetään. UML-kaaviot Dao-luokista löytyvät liitteestä 5.

Kuten Service-tasolla, myös Dao-tasolla käytetään usein rajapintaluokkia testattavuuden ja ylläpidettävyyden vuoksi. Service-luokkien tapaan jokainen Dao-luokka on määritelty ensin rajapintaluokkana.

Jonoja käsittelevä rajapintaluokka on QueueDao. Rajapinnan toteuttaa QueueDaoImpl. Luokka sisältää yhden metodin, jonka avulla tietokannasta haetaan yksittäisen jonon tiedot.

Jonopalvelimia käsittelevä rajapintaluokka on ServerDao. Rajapinnan toteuttaa ServerDaoImpl. Luokka sisältää kolme metodia: yksittäisen jononpalvelimen tiedot, kaikkien jonopalvelimien tiedot, sekä yksittäisen jonopalvelimen tietojen muuttaminen.

Jonoja ja jonopalvelimia yhdessä käsittelee QueueServerDao. Rajapinnan toteuttaa QueueServerDaoImpl. Luokka sisältää useita metodeja sekä jonojen, että jonopalvelimien tietojen hakemiseen, päivittämiseen ja lisäämiseen.

Tilauksia käsittelevä rajapintaluokka on TicketDao. Rajapinnan toteuttaa TicketDaoImpl. Luokka sisältää useita metodeja tilausten luomiseen, muuttamiseen, yksilöllisen tunnuksen eli ID:n luomiseen, tilausten hakemiseen sekä tilauksen mukana tulevien Parameter-luokan ilmentymien tallentamiseen.

4.7 Testaaminen

Projektiin kuului alun perin automaattisten testien luominen. Automaattiset testit kuitenkin rajattiin ulos projektista projektin liiallisen laajuuden ja ajanpuutteen vuoksi.

Testaaminen ohjelmaa kehittäessä on toteutettu HTTP-kyselyiden lähettämiseen ja REST-rajapintojen testaamiseen tarkoitetulla RESTClient-ohjelmalla.

RESTClient-ohjelmassa on mahdollista tallentaa kyselyitä ja lähettää niitä uudelleen useita kertoja. Automaattisten testien sijaan loin jokaiselle kyselytyypille oman tallennetun kyselyn. Kyselyt on mahdollista ajaa peräkkäin. (RESTClient 2014.)

5 Tulokset ja johtopäätökset

Tässä osiossa pohdin projektin kulkua, sekä projektissa käytettyjä teknologioita sekä menetelmiä.

5.1 Projektin haasteet ja onnistumiset

Projektin suunnittelussa painotettu valmiin ja tunnetun teknologian hyödyntäminen kannatti ohjelman toteutusvaiheessa. Suurempia kompastuskiviä ei teknologian takia esiintynyt. Projektissa ainoastaan iBatis oli minulle ennestään tuntematon teknologia, mutta kyseinen kirjasto oli erittäin johdonmukainen ja siksi helppo omaksua. Lisäksi toimeksiantajalla oli erinomaiset esimerkit iBatiksen käytöstä.

Yleisesti projektin suunnittelu ja ohjelmointityö kävi toimeksiantajan kanssa sujuvasti. Valmiit tietokantakuvaukset ja XML-skeemat mahdollistivat nopean siirtymisen suunnitteluvaiheesta toteutusvaiheeseen.

Projektissa minulle haasteellisimmaksi osoittautuivat projektinhallinnolliset asiat ja korrektin raportointikäytännön omaksuminen.

5.2 Arkkitehtuurimallit, kehikot ja kirjastot

REST-arkkitehtuurimalli oli minulle ennestään vain etäisesti tuttu. Projektin edetessä malli kuitenkin kirkastui. Selkein etu mallissa on palvelinpuolen ohjelmoinnin yksinkertaistaminen. Kun asiakas ja palvelinohjelmat ovat eriytettyjä ja keskustelevat vain esim. XML tai JSON muodossa olevan tiedon välityksellä, on ne vaivatonta kehittää täysin toisistaan erillään kunhan kehittäjät sopivat käytetystä viestintätyypistä. Tässä projektissa käytetty XML on yleisesti tuettu ja tunnettu jäsennellyn tiedon muoto, mitä olemassa oleva TQM-ohjelma tuki jo ennestään. Spring-kehikko mahdollisti XML-muotoisen tiedon automaattisen parsimisen ja luomisen.

REST-arkkitehtuurimallin toteuttaminen osoittautui yllättävän helpoksi. Vaikka itse teoria on varsin abstrakti, käytännössä tärkeimmiksi ominaisuuksiksi huomasin asiakas ja palvelinohjelmien työnjaon, sekä palvelimen tilattomuuden.

Kolmitasoarkkitehtuuria olen käyttänyt useissa sovelluksissa aikaisemminkin. Tässä projektissa kirjoitin ehkä hieman liikaa logiikkaa Web-tasolle, mutta Controller-luokat pysyivät silti yksinkertaisina. Service-luokista tuli paljon tyypillistä Java sovellusta yksinkertaisempia REST-arkkitehtuurin ansiosta.

Tietokantarajapintana toiminut iBatis mahdollisti alun perin DB2 tietokannalle tarkoitettujen kyselyn vaivattoman muuttamisen toimimaan MySQL-tietokannan kanssa. Suuria muutoksia ei oikeastaan edes tarvittu. Koska SQL-kyselyt oli eriytetty omaan XML-tiedostoonsa, on jopa mahdollista käyttää samaa tiedostoa useamman ohjelman tietokantarajapintana.

5.3 Jatkokehittäminen

Projektin jatkokehitystarpeita ovat virhekoodien, viestien ja muiden tulosteiden eriyttäminen omiin tiedostoihinsa niin, että Java-koodissa ei ole kovakoodattuna yhtään viestiä. Koska viestit ja koodit eivät toimeksiantajan puolelta olleet vielä täysin suunniteltuja ja koska koodit ovat erittäin sidoksissa olemassa olevan TQM-ohjelman toimintaan, jäi erillisten asetus ja viestitiedostojen yhdistäminen ohjelmaan jatkokehityksen alaisuuteen.

Kattavat testit ovat myös jatkokehitystarve. Käsin testaaminen tallennetuilla HTTP-kyselyillä muodostuu nopeasti liian työlääksi, eikä mahdollista nopeaa toiminnallisuuden varmistamista. Automaattiset ja kattavat testit lisäävät ohjelman luotettavuutta ja kehitettävyyttä. Nykytilassa käytettyä RESTClient-ohjelmaa voisi käyttää automatisoitujen testien luomisessa hyväksi. Ohjelmasta on olemassa komentoriviversio, jota voi komentaa esimerkiksi erillisellä testiohjelmalla.

Riippuen ympäristön syöttövirheiden todennäköisyydestä ja kriittisyydestä voi olla tarpeen kehittää vastaanotettujen XML-muotoisten viestien varmennusta. Spring-kehikko varmistaa, että XML on oikeanmuotoista, mutta ei kykene varmistamaan, että esim. syötetyt tiedot ovat oikein.

Lähteet

Eclipse Foundation 2014. About the Eclipse Foundation. Luettavissa: http://www.eclipse.org/org. Luettu: 29.4.2014.

Ecma International 2013. Standard ECMA-404 The JSON Data Interchange Format. Ecma International. Geneve. Luettavissa: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. Luettu: 22.4.2014.

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

iBatis. iBATIS Data Mapper Developer Guide. Luettavissa: http://ibatisnet.sourceforge.net/DevGuide.html. Luettu 6.5.2014.

MySQL 2014. MySQL Editions. MySQL. Luettavissa: http://www.mysql.com/products. Luettu: 6.5.2014.

Oracle 2014. JavaServer Pages Overview. Oracle. Luettavissa: http://www.oracle.com/technetwork/java/overview-138580.html. Luettu: 6.5.2014.

Ort E., Mehta B. 2003. Java Architecture for XML Binding (JAXB). Oracle. Luettavissa: http://www.oracle.com/technetwork/articles/javase/index-140168.html. Luettu: 25.4.2014.

RESTClient 2014. WizTools RESTClient projektisivu. Luettavissa: https://github.com/wiztools/rest-client. Luettu: 6.5.2014.

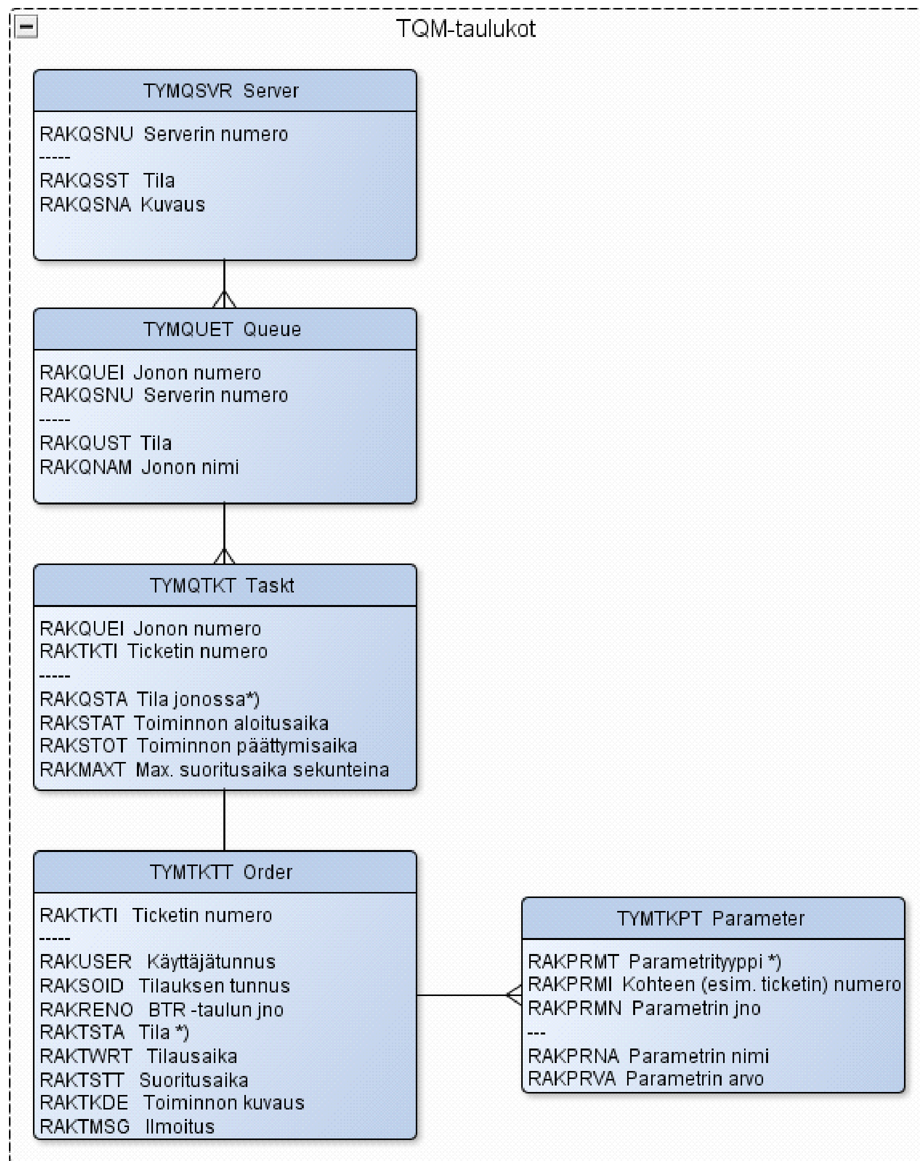
Spring Documentation 2014. Spring Framework Reference Documentation. Overview of Spring Framework. Pivotal. Luettavissa: http://docs.spring.io/spring/docs/4.1.0.BUILD-SNAPSHOT/spring-framework-reference/htmlsingle. Luettu: 5.5.2014.

W3C 1999. Introduction to HTML 4. What is HTML? W3C. Luettavissa: http://www.w3.org/TR/html401/ intro/intro.html. Luettu: 25.4.2014.

W3C 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C. Luettavissa: http://www.w3.org/TR/REC-xml. Luettu: 22.4.2014.

Liitteet

Liite 1. TQM-Controllerin tietokannan kuvaus



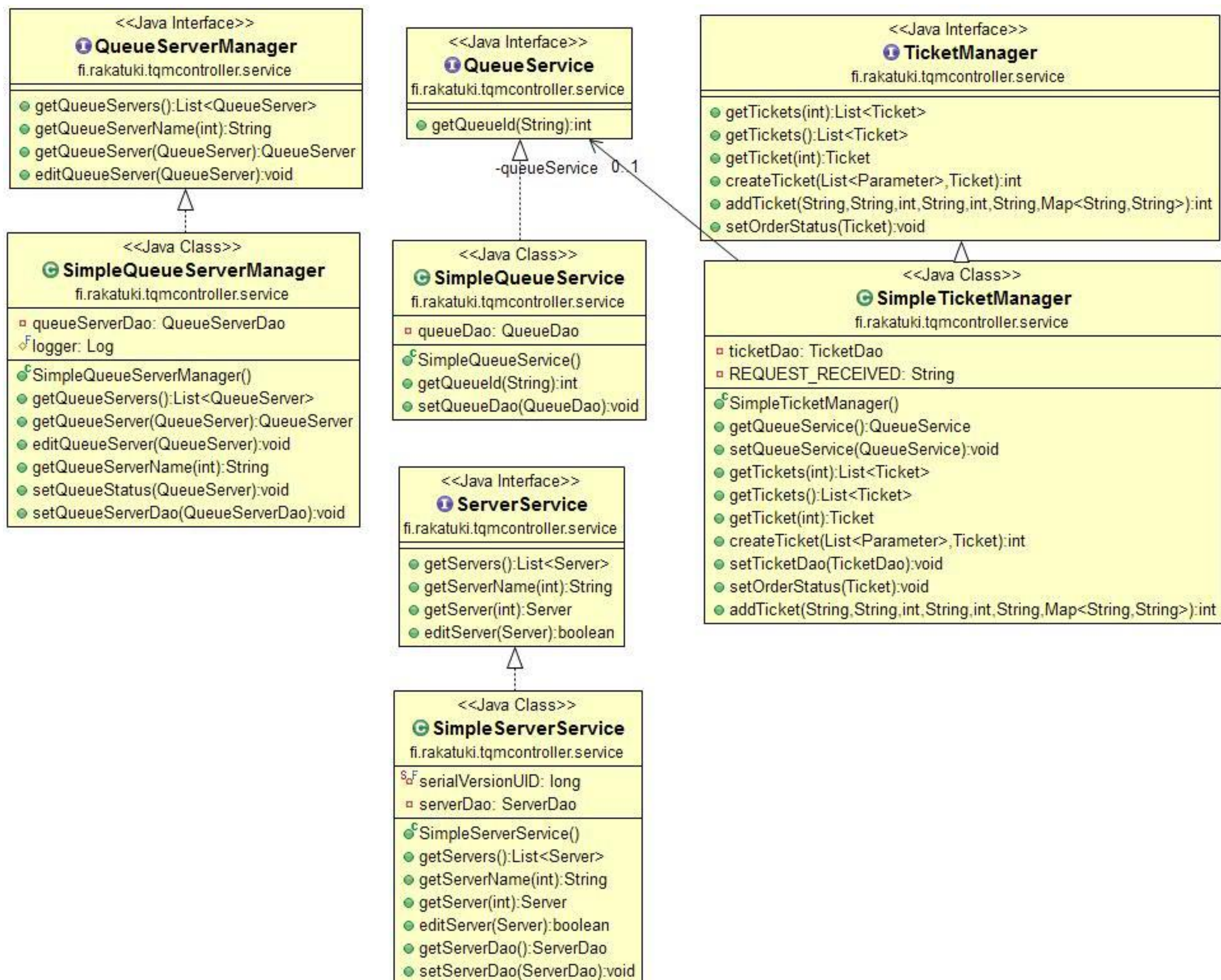
Liite 2. MySQL-tietokannan luomislauseet

```
CREATE TABLE TYMQTKT (
                              RAKQUEI INTEGER NOT NULL,
                              RAKTKTI INTEGER NOT NULL,
                              RAKQSTA INTEGER DEFAULT 0 NOT NULL,
                              RAKSTAT TIMESTAMP DEFAULT CURRENT TIMESTAMP NOT NULL,
                              RAKSTOT TIMESTAMP DEFAULT '0001-01-01-00.00.00.000000' NOT NULL,
                              RAKMAXT INTEGER DEFAULT 0 NOT NULL,
                              RAKQPID INTEGER DEFAULT 0 NOT NULL
               );

CREATE UNIQUE INDEX TYMQTKT_PK ON TYMQTKT (RAKQUEI ASC, RAKTKTI ASC);

ALTER TABLE TYMQTKT ADD CONSTRAINT TYMQTKT_PK PRIMARY KEY (RAKQUEI, RAKTKTI);


CREATE TABLE TYMQSVR (
                              RAKQSNU INTEGER NOT NULL,
                              RAKQSST INTEGER,
                              RAKQSNA VARCHAR(400) DEFAULT '' NOT NULL
               );

CREATE UNIQUE INDEX TYMQSVR_PK ON TYMQSVR (RAKQSNU ASC);

ALTER TABLE TYMQSVR ADD CONSTRAINT TYMQSVR_PK PRIMARY KEY (RAKQSNU);


CREATE TABLE TYMQUET (
                              RAKQUEI INTEGER NOT NULL,
                              RAKQSNU INTEGER,
                              RAKQUST INTEGER,
                              RAKQNAM CHAR(32)
               );

CREATE UNIQUE INDEX TYMQUET_PK ON TYMQUET (RAKQUEI ASC);

ALTER TABLE TYMQUET ADD CONSTRAINT TYMQUET_PK PRIMARY KEY (RAKQUEI);


CREATE TABLE TYMTKPT (
                              RAKPRMT INTEGER NOT NULL,
                              RAKPRMI INTEGER NOT NULL,
                              RAKPRMN INTEGER NOT NULL,
                              RAKPRNA CHAR(64) NOT NULL,
                              RAKPRVA VARCHAR(400) NOT NULL
               );

CREATE UNIQUE INDEX TYMTKPT_PK ON TYMTKPT (RAKPRMT ASC, RAKPRMI ASC, RAKPRMN ASC);
```

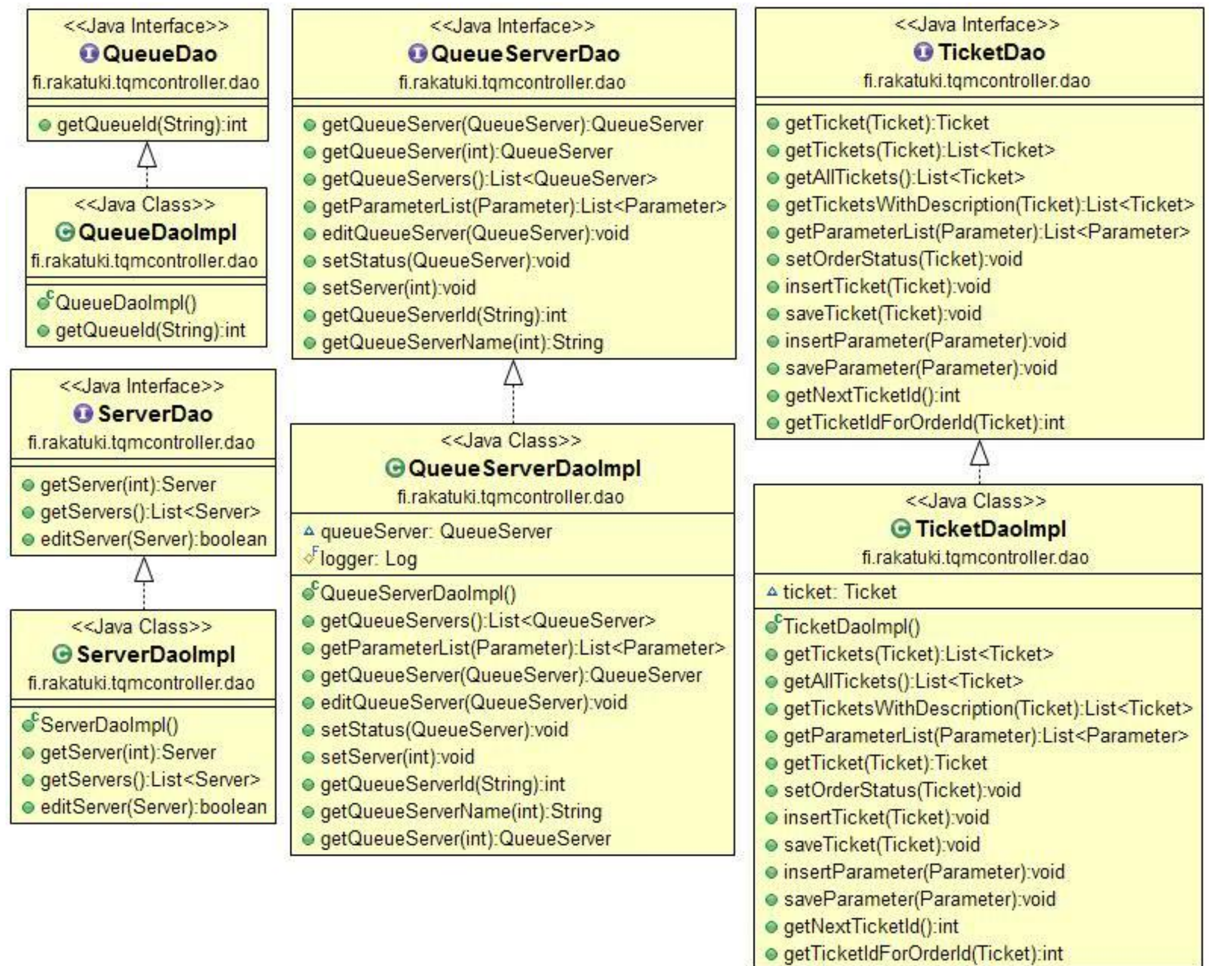
```
ALTER TABLE TYMTKPT ADD CONSTRAINT TYMTKPT_PK PRIMARY KEY (RAKPRMT, RAKPRMI, RAKPRMN);

--<ScriptOptions statementTerminator=";"/>

CREATE TABLE TYMTKTT (
                              RAKTKTI INTEGER NOT NULL,
                              RAKUSER CHAR(8) NOT NULL,
                              RAKSOID CHAR(16) NOT NULL,
                              RAKRENO INTEGER NOT NULL,
                              RAKTSTA INTEGER DEFAULT 0 NOT NULL,
                              RAKTWRT TIMESTAMP DEFAULT CURRENT TIMESTAMP NOT NULL,
                              RAKTSTT TIMESTAMP DEFAULT CURRENT TIMESTAMP NOT NULL,
                              RAKTMSG VARCHAR(400) DEFAULT '' NOT NULL,
                              RAKTKLO VARCHAR(200) DEFAULT '' NOT NULL,
                              RAKTKDE VARCHAR(200) DEFAULT '' NOT NULL,
                              RAKMAXT INTEGER DEFAULT 1800 NOT NULL
               );

CREATE UNIQUE INDEX SQL100406132952430 ON TYMTKTT (RAKTKTI ASC);

ALTER TABLE TYMTKTT ADD CONSTRAINT SQL100406132952430 PRIMARY KEY (RAKTKTI);
```

Liite 3. Web-tason luokat

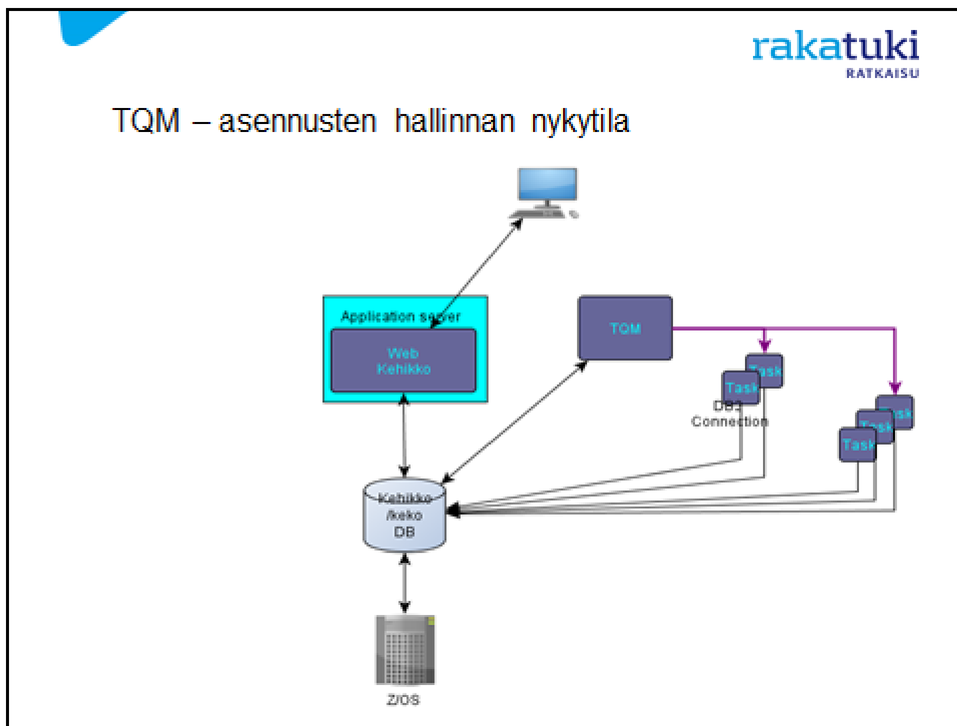
Liite 4. Service-tason luokat



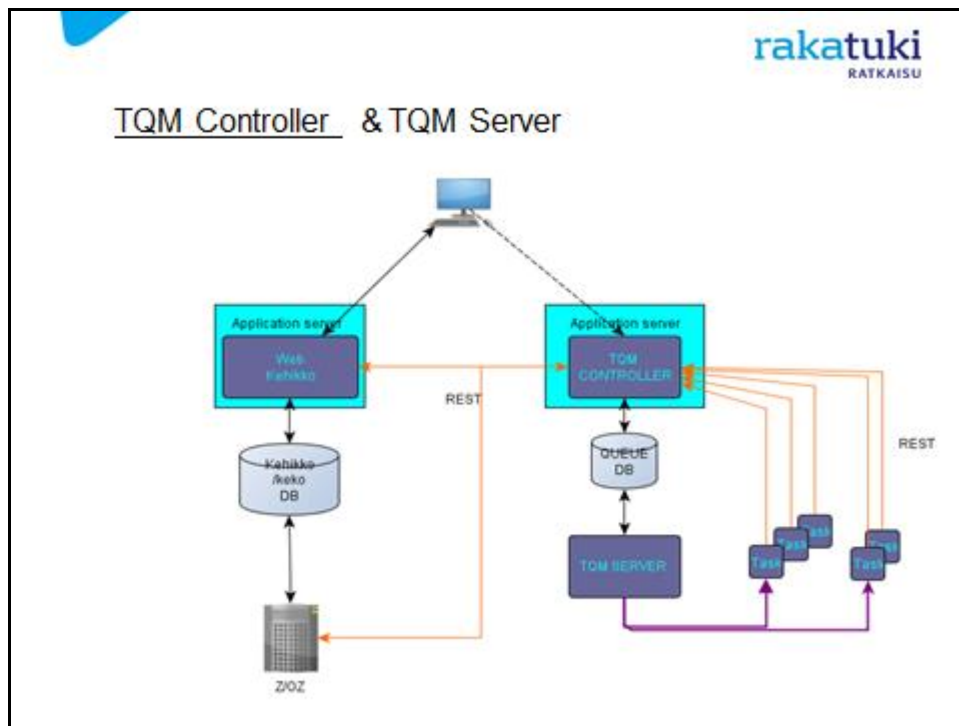
Liite 5. Dao-tason luokat



Liite 6. TQM-ohjelman ja Raka-Kehikon yhteistoiminta projektin alussa



Liite 7. TQM-ohjelman ja Raka-Kehikon yhteistoiminta projektin tavoitetilassa



Liite 8. TQM-Controllerin arkkitehtuuri