Kehityssuunnitelma – TIKO05

Team TIKO05 – Kai Kirjavainen, Kenneth Roehr ja Mika Kumanto

Kehityssuunnitelma

17.05.2011

Sisällysluettelo

1 Arkkitehtuuri – pom.xml purettuna

Kattavan kuvauksen sovelluksen luokista saa teknisestä määrittelystä (Hus_tekninen_maarittely.docx) ja muutenkin toiminnallisuudesta. Tämän apuna voidaan käyttää sovelluksesta luotavaa JavaDoc-dokumentaatiota, joka generoidaan Mavenilla komennolla:

$ mvn javadoc:javadoc

Tämä komento generoi sovelluksen target/site/apidocs kansioon JavaDoc-dokumentaation, joka voidaan siirtää web-palvelimelle. Muitakin Mavenin komentoja voidaan toki myös generoimaan tietoa (site, sonar:sonar, jne.). Sovelluksen arkkitehtuuri on melko monimutkainen ja tämän lisäksi ehkä hieman avittaa sonar tai site –komennot, joka kirjoittavat www-sivulle käytettävät jar-tiedostot. Kaikki kirjastot määritellään Mavenin pom.xml –tiedostossa ja listaan oleellisimpia kirjastoja ja toiminnallisuuksia sieltä:

- groupId: fi.hus, artifactId: HUSku ja paketointi war. Tässä ehkä pitäisi miettiä onko tuo groupId ihan kaikkien sääntöjen mukaisesti nimetty, koska paketit sijaitsevat fi.hus.costsharing paketin alaisuudessa
- Repositories: SpringSource EBR (Alettiin käyttää tätä, koska pitäisi olla testattua ja muuta siistiä. Vähän ongelmia ilmeni vanhojen kirjastojen kanssa, joten siirryttiin käyttämään JBoss repoa) JBoss repository ja Mavenin keskusrepot
- Spring framework 3.0.5, Spring Integration 2.0.3
- Sonarille määritelty erikseen hrep -urli, mutta vakiona ei sitä tarvitse. Käytettiin kehityspalvelimella nginx-palvelinta frontendinä ja tietyt osoitteet reititettiin toisille palvelimille.
- Spring-library, joka sisältää kaikki oleelliset Spring kirjastot. Ehkä jotain ylimääräisiä kirjastoja tätä projektia varten (esim. portlet).
- Apache Commons dbcp, joka on tietokannan importtia varten
- Hibernate 3.6.3, Hibernate EntityManager ja JPA 2.0 api –kirjastot OR/M:ää varten
- Hibernate validator 4.1.0 ja javax.validation 1 valdointia varten
- JUnit 4.8.1
- MySQL 5.1.6 ei käytössä, mutta kirjastot valmiina. Tarvitsee vain muiden asetusten vaihtoa.
- Castor XML 1.3.1 XML-parserointia varten. Tämä on otettu suoran pääreposta, koska Spring EBR Castorit olivat rikkinäisiä. Tai aiheuttivat virheellisiä tilanteita ja niitä ei ollut päivitetty useampaa vuoteen
- H2 1.2.147 Java RDBMS. Ajettiin sovelluskäytössä muistissa ja tiedostossa. Testipalvelimella kanta oli H2:lla, joka pyöri erillisenä instanssina.
- Servlet 2.5, JSP 2.1
- Spring Integration ja Spring Integration File –kirjastot
- 1.6 Java

- mvn:jetty run komenolla saadaan Mavenista käyntii instanssi Jetty-palvelimesta. Käytetty versio 7.2.2.v20101205 ja ajettiin portissa 18080. Tämä siis on edelleen sen takia, koska testipalvelimelle oli säädetty käyttämään nginx-palvelinta frontendinä
- mvn:tomcat run komennolla saadaan Mavenista käyntii instanssi Tomcat-palvelimesta, joka toimii vakioasetuksilla portissa 8080. Testattu versio jokin 6.x
- Testausta varten JUnit-plugin, jota voidaan käyttää mvn test ja raportointia varten määritelty findbugs. Eli tämä findbugs-raportti saadaan generoitua mvn site komennolla, joka luo kyllä kaikkea muuta mahdollista

2 Työkalut

Käytetyt ohjelmointiympäristöt: Eclipse 3.4.x (m2e), Eclipse 3.6.x (m2e), IntelliJ IDEA 10.2, 10.3
Projektihallintajärjestelmä: Maven 2.x ja 3.x
Tietokanta: H2
Webpalvelin: Jetty 6.x, 7.x ja Tomcat 6.x testattu toimivan. Ei pitäisi toimia vakiona Tomcat 5.5 – palvelimella
Selaimet: Chrome, Firefox ja IE
UML-piirrostyökalu: DIA
Versiohallinta: SVN ja Sventon sen ohella
Raportointi: Sonar
Dokumentoinnin versiohallinta: Google Docs

Versiohallinnassa on laitettu Eclipsen asetuksia, mutta ei IntelliJ IDEA. Tosin näitä ohjelmointiympäristön sidonnaisia asetuksia ei pitäisi käyttää, koska kaikki ”äly” tulee Mavenistä.

3 Ongelmat

Sovelluksessa on lukuisa määrä ongelmia, joihin ei ole otettu kantaan ja tehty hieman hölmösti. Muutenkin käytetyt metodit ja käytännöt eivät varmasti ole kaikkein parhaimpia, mutta tehty mikä tehty. Tähän on listattu jo havaituista ongelmista ja ratkaisusta, jotka pitäisi mielestämme ensimmäisenä korjata.

- Erän maksu (Payment) –luokassa tallennetaan HashMappiin %-osuus ja laskurivi. Tämän avulla saadaan ohjelman tietoon, että mistä laskuista ja laskuriveistä maksu muodostoo. Tässä vain on mahdollisesti eräänä ongelmana se, että HashMapin %-osuus ei ole tarpeeksi tarkka. Se on vain kahdella desimaalilla siis esim 60 % = 0,60. Ohjelmassa itestää pystyy laittaa tarkempia prosenttijakoja
- Muutenkin olisi syytä tarkistaa kaikki tallennetut desimaaliluvut, että niitä ei tallenneta erityisen liikaa tai vähän kantaa. Tämä pitemmällä aikavälillä huono juttu kannalle, jos kantaan tallennetaan vain tyhjiä desimaaleja älytön määrä
- Tiedon validointi toimii, mutta ei anna minkäänlaisia virheilmoituksia ja saman tiedon syöttö räjäyttää sovelluksen ja antaa sekavan virheilmoituksen. (insert, duplicate primary key, liitokset jne.)
- Eräajoa ei ole kunnolla testattu. En tiedä onko nykyinen cron-määritelmä oikein ja liiketoiminnan kannalta ei ole välttämäti oikein. Lyhyempinä sykleinä toimi oikein, mutta viimeisintä määritelmää ei ehditty testaamaaan
- Virhetilanteille ei ole kunnollista käsittelyä ja logitusta pitäisi selvästi parantaa, jos ja kun virheitä syntyy
- Tiedostopolleri (InvoicePoller) ei ole kunnolla koestettu ja mietitty, miten virhetilanteissa tulee tapahtumaan
- Muutenkin pitempää testausta ei ole suoritettu ja käyttäjätestaus myös. Eli mitään tieto, onko asiakas tyytyväinen valikkorakenteisiin tai vastaaviin.
- Tuottajakoodi on kirjoitettu staattisesti CostSharing -luokkaan. Tätä ei siis pysty jälkikäteen mitenkään muokkaamaan.
- Ohjelma ei tue maksuhyvityksiä, koska silloin kpl-määrä pitäisi muuttua -1. Nyt se on aina 1.
- Ulkoasu ei ole yhtenäinen jokaisella sivulla
- Ulkoasu on tietyillä sivuilla melkoisen sekava. Varsinkin sivuilla, missä muutetaan %-yksiköitä. Muutenkin ei ole käyttöliittymä täydellisen looginen ja tietenkin puuttuu asiakkaan mielipide
- JPA:n annotaatiot pitäisi tutkia ja ajatuksella miettiä onko ne yhteydet järkevästi tehty. Muutenkaan minkäänlaista optimointia ei ole suoritettu, joten suorituskyky on yksi kysymysmerkki. Ei pakolla kovin oleellinen tässä sovelluksessa, koska ainakin tällä tiedolla laskutoimintaa merkittävästi. Oleellisempaa olisi tiedon validius ja virheenhallinta.
- XML-parseria olisi syytä siistiä. Sisältää karuja ratkaisuja, mutta osittainen syy on myös erittäin huonosti toteutettu XML-tiedosto. EDI-sanomat suoraan XML-tiedostoksi ja käytetään koko XML-

määritelmään hyvin väärin. Kaikki tieto on esimerkiksi attribuuttien sisällä. Muutenkin melko kummallinen, mutta sille ei tässä vaiheessa voi tehdä mitään.

- Ei tukea monikielisyydelle. Kaikki sivuston teksti on kirjoitettu suoraan JSP-sivuihin.
- Geneerinen DAO/Service-luokka ei ole välttämäti kannattava tapa. Tätä olisi syytä tutkia enemmän ja miettiä aiheuttaako ongelmia pittemmän päälle.
- JSP-sivut nimetty suomeksi, mutta muuten ohjelmointikieli on englanniksi
- Web-logiikkaa ei testata ollenkaa ja muutenkin testaus jäi hieman uupuu, vaikka sitä yritettii alusssa toteuttaa. Tällä hetkellä vain yli runsas puolet ohjelmakoodista testataan automaattisissa testeissä (63,5 %). Muutenkin niiden rakennetta olisi syytä korjata ja nimeämiskäytäntöjä. InvoicePollerTest-luokka on myös rikki
- Ei sisäänkirjautumista
- Sonar-raportin mukaan seuraavia mahdollisia ongelmia: CostSharing-luokka on liian monimutkainen, merkkijonen kopioita, käyttämättömiä attribuutteja ja kaikkea muuta vastaavaa pientä. FindBug-raportissa ei ole ainakaan viimeisempien tulosten mukaan virheitä
    - Sonarin käyttö on kehityksessä varsin hyödyllistä ihan muutenkin, koska voidaan seurata miten koodi on kehittinyt
- Onko Integer-tyyppisten ID-arvojen käyttäminen väärin? Skipfish ainakin valistaa näistä, että mahdollistaa integer overflow vector -varoituksia. Toki tämä sovellus ei tule julkiseen verkkoon, joten virhesyötettä tuskin mahdottomasti tulee, mutta ihan lähtökohtaisesti tämä voi olla väärä tapa toteuttaa asioita.
- Maksusta ei voi poistaa tai lisätä yksittäistä riviä, vaan täytyy käydä koko laskun sisältö läpi.
- Jos ruvetaan muokkaamaan laskua ja tallennetaan, virheellisesti infokoodit kirjautuu tyhjä-arvolla. Johtuu lähinnä ylemmästä rajoitteesta
- Kaksitasoiset kustannuspaikat eivät ole aivan oikein tehty tai ei vastaa annettua XLS-tiedostoa. Sovelluksen laskentaan ei vaikuta annettu asiakaskoodi.

Ohjelmma on siis melkoisesti pieniä virheitä tai tilanteita, mistä saa aiheutua ongelmia. Tämä vain on lyhyt listaus. Todennäköisesti lista ongelmista on pidempi, jos sovellusta alettaisii kunnolla testaamaan ja rakennetta mietittäisiin uusiksi.

4 Jatkokehitys

Teitenkin tulevat suunnitelmat jatkokehitykselle ei ole tämän ryhmän asia, mutta tähän on listattu asioita, jotka ovat luonnollisia jatkokehityskohteita. Taas pitkä lista asioista, jotka arvioitu vaativuuden mukaan ja paljonko se tuottaa todennäköisesti muutoksia. Yksi tähti tarkoittaa muutosta, joka on helpoiten toteutttavissa ja sitä enemmän tähtiä, sitä enemmän tarvitsee työtä.

- **Sisäänkirjautuminen** on välttämätön toteuttaa, vaikka sovellus tulee toimiin vain sisäverkossa. Alkuvaiheessa se voi olla .htpasswd, mutta Spring Security -rajapinta on helppo integroida nykyiseen sovellukseen. Toki enemmän työtä saattaa aiheuttaa, jos sovelluksessa on tarvetta olla eritasoisia käyttäjiä ja integrointi yrityksen LDAP-hakemistoon. **Arvio:** * tai **
- **Virheentarkistus** on kyllä jo olemassa taustalla, mutta pitäisi myös näkyä käyttäjälle, mistä virhe syntyy. Myös virheentarkistusta pitäisi parantaa. **Arvio:** *
- **Käyttöliittymään AJAX-toiminnallisuutta,** joka helpottaisi ja parantaisi sovelluksen toiminallisuutta. Tätä toki kannattaa miettä, että millä tavalla ruvetaan toteuttamaan ja mikäli runsaasti kokemusta jo kirjastoista ei varmastikkaan kovin vaikea toimenpide. Joko itse kirjoittaa jollakin JS-kirjastolla esimerkiksi jQuery ja tämän lisäksi käyttä Jackson JSON-prosessointia. Toinen vaihtoehto olisi siirtyä käyttämään jotain rajapintaa, joka huolehtii tästä kaikesta. Esimerkiksi Wicket, Vaadin, GWT tai mitä niitä nyt onkaan. Tietty nuo kaikki ovat hieman erilaisia projekteja ihan toiminallisuudeltaan, mutta nykyiset JSP-sivut ovat melkoisen hirveitä. **Arvio:** *** (** jos kokemusta jo ajaxista. Niin tuskin aiheuttaa suurempia ongelmia.)
- **Sovelluksen toiminallisuuksien hajauttaminen palveluihin**. Mikäli sovellusta tullaan käyttämään tulevaisuudesta mistään toisesta sovelluksesta olisi syytä kustannusjakoa pystytä tarjomaan palveluna. Järkevintä olisi luultavasti toteuttaa REST-metodeja käyttäen, jota Spring MVC tukee hyvin ja yksinkertaista toteuttaa. **Arvio:** ***
- **Sähköpostihuomautukset** olisi varsin mainio idea käyttäjälle ilmoittaa, kun sovellukseen tulee laskuja, virheellisiä laskurivejä. Spring Framework taas sisältää tuen, jolla nämä tehdä suhteellisen pienellä vaivalla. Tietty vähän pohdintaa aiheuttaa, että mihinkä kaikkiin sähköposteihin tulee ilmoituksia. Eli joko jokaisen käyttäjätiliin määritellään, että vastaanottaako hän huomautuksia vai onko sovelluksessa vain lista sähköposteista, mihin lähetetään huomautus. **Arvio:** *
- **Lokitusta** pitäisi parantaa eli enemmän virhetilanteista, josta otetaan virheitä talteen ja millä Log4j-tyypillä ne kirjoitetaan. Kaikki lokitus syntyy info tai fatal/error-tasolla tällä hetkellä. **Arvio:** *