Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Ville Kautonen

Java web -sovelluksen tasonnosto

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

21.4.2020

Tekijä Otsikko Sivumäärä Aika	Ville Kautonen Java web -sovelluksen tasonnosto 29 sivua 21.4.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	tieto- ja viestintätekniikka
Ammatillinen pääaine	ohjelmistotuotanto
Ohjaajat	yliopettaja Auvo Häkkinen projektivastaava Jenni Björklund-Seppälä

Insinöörityön tavoitteena oli tutustua Java web -sovelluksen tasonnostoon ja sen toteutukseen. Tasonnosto on prosessi, jossa ohjelmisto päivitetään ajan tasalle. Työn tilaajana oli kansainvälinen yritysstrategia-, digitalisaatio-, ulkoistus- ja teknologiakonsultointia tarjoava yritys.

Insinöörityössä tarkasteltiin Java Platform Enterprise Edition -ohjelmistokehitysalustaa käyttävää web-sovellusta, jota isännöitiin JBoss Enterprise Application Platform -palvelimella. Alustavasti tutustuttiin sovelluksen ja palvelimen muodostamaan järjestelmäkokonaisuuden rakenteeseen. Järjestelmän rakenteesta saatiin käsitys siitä, mitkä järjestelmän osat vaativat päivitystä.

Projektissa tutustuttiin tasonnoston syihin, haasteisiin ja mahdollisuuksiin. Tasonnosto tehtiin, jotta järjestelmä pysyisi ajan tasalla jatkuvasti muuttuvalla tieto- ja viestintätekniikan alalla. Tasonnoston toteutustavoiksi tutkittiin järjestelmän JBoss-palvelimen ja ICEfaces-käyttöliittymäkirjaston päivittämistä.

Tasonnoston toteutustavaksi valittiin JBoss-palvelimen päivitys. Syynä tähän oli se, että ICEfaces-käyttöliittymäkirjaston päivitys olisi ollut liian raskas ja aikaa vievä toteuttaa. Päivitys toteutettiin seuraamalla Red Hat -yrityksen palvelimen päivitysohjeita sekä hyödyntämällä kyseisen yrityksen tarjoamia dokumentaatioita. JBoss-palvelimen päivityksellä saatiin aikaan vakaampi järjestelmä, jonka palvelin on tuettu vielä vuoteen 2023 asti. Päivitetyllä palvelimella luotiin pohja muiden järjestelmän osien päivitykselle ja uusien ominaisuuksien lisäämiselle.

Avainsanat	Java, JBoss, web-sovellus, ICEfaces, tasonnosto

Author Title Number of Pages Date	Ville Kautonen Bringing Java Web Application Up to Date 29 pages 21 April 2020
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Software Engineering
Instructors	Auvo Häkkinen, Principal Lecturer Jenni Björklund-Seppälä, Project Manager

The purpose of this thesis was to study the updating process of a Java web application and how it should be implemented. The updating process constitutes of bringing an application up to date. The commissioner of the study was a multinational company offering business, digitalization, outsourcing and technology consulting.

The application used here was a Java Platform Enterprise Edition based web application hosted on a JBoss Enterprise Application Platform server. The project began with an initial study of the system shedding light on which parts of the system needed updating.

The thesis also delves into the reasons, challenges and opportunities that come from the updating process. The update was done so that the system could keep up with the ever-changing software requirements of the modern age. It was decided that two parts of the system required updating the JBoss server and ICEfaces user interface library.

In the end it was decided that only the JBoss server would be updated as updating the ICEfaces library would have been too resource intensive and time consuming. The update was carried out by following a JBoss server migration guide and documentation provided by Red Hat Inc. The update made way for a more stabilized system supported all the way to the year 2023. Updating the server also laid groundwork for future updates of other parts of the system as well as implementing new features.

Keywords	Java, JBoss, Web Application, ICEfaces, update

Sisällys

Lyhenteet

1 Johdanto 1

2 Päivitettävän web-sovelluksen arkkitehtuuri 2

2.1 Java Platform Enterprise Edition 4
2.2 JBoss Enterprise Application Platform 5
2.3 IBM Database 2 6
2.4 Hibernate 6
2.5 XHTML, JSF ja ICEfaces käyttöliittymätason perustana 7

3 Tasonnoston tarkoitus 8

3.1 Tasonnoston syyt 8
3.2 Tasonnosto insinöörityön web-sovelluksessa 9

4 Tasonnoston toteutustapoja 10

4.1 ICEfaces-päivitys ja JBoss-päivitys 10
4.2 Pelkkä JBoss-päivitys 16

5 Valittu toteutus ja sen testaus 21

6 Järjestelmän tila tasonnoston jälkeen ja tulevaisuuden mahdollisuudet 23

7 Yhteenveto 25

Lähteet 28

Lyhenteet

IBM Db 2	IBM Database 2. IBM:n tuottama relaatiotietokanta.
Java EE	Java Platform Enterprise Edition. Ohjelmistokehitysalusta, joka laajentaa Java Platform Standard Editionin ominaisuuksia siten, että se tukee suurempien ja palvelinpohjaisten sovellusten kehitystä uusilla kirjastoilla.
Java SE	Java Platform Standard Edition. Pääasiallinen Javan ohjelmistokehitysalusta, joka tarjoaa yleiset Java sovelluskehitykseen tarvittavat kirjastot ja ohjelmointirajapinnat.
Jboss EAP	Jboss Enterprise Application Platform. Red Hat -yrityksen tarjoama yritystason palvelinohjelmisto.
JPA	Java Persistence API. Ohjelmointirajapinta, joka kuvailee relaatiotietokantojen hallintaa.
JSF	Java Server Faces. Java-spesifikaatio, joka helpottaa käyttöliittymien rakentamista web-sovelluksille tarjoamalla valmiskomponentteja.
ORM	Object Relational Mapping. Käsite, jolla tarkoitetaan tietokannan tiedon mallintamista olio-ohjelmoinnin mukaisina rakenteina.
RIA	Rich Internet Application. Web-sovellus, joka toimii kuten työpöytäsovellus.
XHTML	eXtensible Hypertext Markup Language. HTML:stä kehitetty rakennekieli, joka täyttää XML-kielen standardin.

1 Johdanto

Insinöörityön tavoitteena oli tutustua JBoss Enterprise Application Platform -palvelinta hyödyntävän Java web -sovelluksen tasonnostoon. Tasonnostolla tarkoitetaan järjestelmän tai sovelluksen ohjelmiston päivitystä ja ajan tasalle tuomista.

Tasonnostoprosessi voidaan toteuttaa monella tavalla, joka tekee sen toteuttamisesta hankalaa. Tästä syystä insinöörityössä tutustuttiin tasonnoston toteutustapoihin ja niiden hyötyihin sekä haittoihin. Näistä toteutustavoista valittiin toimivin ratkaisu asiakkaan tarpeisiin.

Työn tilaajana oli kansainvälisesti toimiva yritysstrategia-, digitalisaatio-, ulkoistus- ja teknologiakonsultointia tarjoava yritys. Kyseinen yritys hyötyi työstä siten, että tehty tasonnosto auttoi pitämään web-sovelluksen ajan tasalla jatkuvasti kehittyvällä ja muuttuvalla teknologian alalla. Ajan tasalla pysyminen on olennaista, jotta järjestelmän tulevaisuuden ylläpito ei kärsi vanhentuneesta ohjelmistosta, jolle ei ole enää tukea. Insinöörityö tarjosi myös syvän katsauksen tasonnoston vaihtoehtoihin, joka auttoi tekemään toimivimman ratkaisun asiakkaan sen hetkisen tilanteen perusteella.

Aluksi työssä käytiin läpi järjestelmän hyödyntämät teknologiat sekä miten ne sopivat yhteen ja muodostavat järjestelmän rakenteen. Alussa tutustuttiin siis web-sovelluksen taustalla olevaan palvelinteknologiaan, kirjastoihin, koodikantaan sekä tietokantaan.

Järjestelmään tutustumisen jälkeen pohdittiin, mitä tasonnoston yhteydessä tulisi tehdä ja miksi. Tässä vaiheessa selvitettiin, mitä hyötyjä tasonnostosta on sekä miksi tasonnoston tekeminen on isossa projektissa hankalaa.

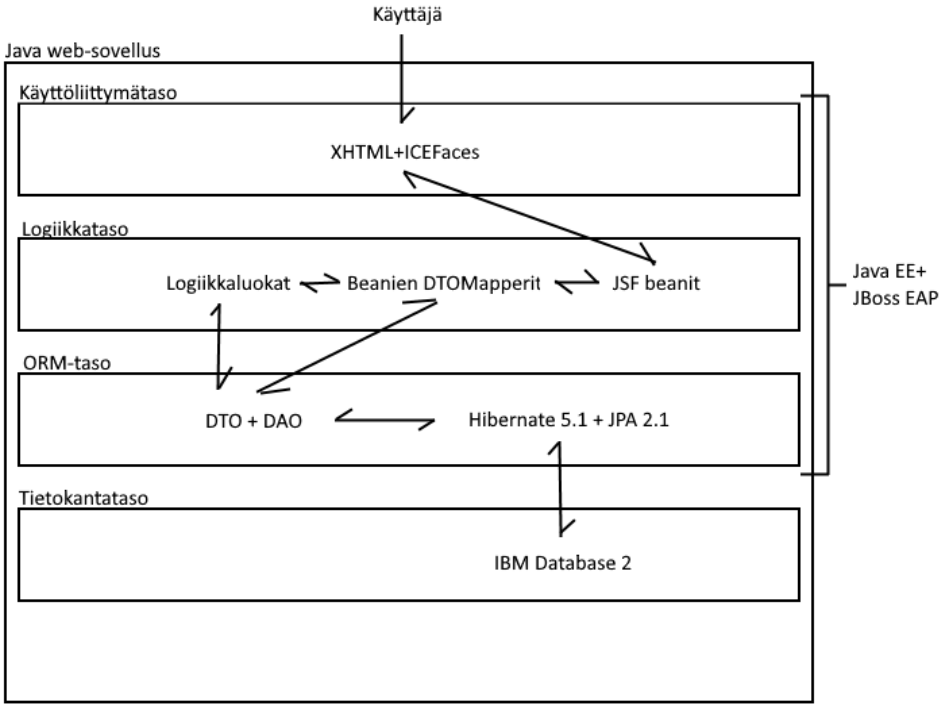
Kun oli selvitetty, miksi tasonnosto tulisi tehdä, selvitettiin tapoja toteuttaa tasonnosto. Tasonnoston toteutustapojen tutkiminen aloitettiin selvittämällä, mitä järjestelmän osia, kuten kirjastoja ja palvelinta, tulisi päivittää ja missä oli tehtävä kompromisseja esimerkiksi yhteensopimattomuuden tai ajallisten puutteiden vuoksi. Toteutustapoihin tutustumisen yhteydessä tarkasteltiin myös, mitä muutoksia järjestelmän koodikantaan oli tehtävä, jotta web-sovellus toimisi vielä päivitysten jälkeenkin.

Lopuksi työn tuloksista pohdittiin tehdyn tasonnoston hyötyjä ja tulevaisuuden mahdollisuuksia. Tulevaisuuden kannalta käytiin läpi myös tehdyn tasonnoston haittapuolet. Tämän jälkeen pohdittiin myös, mitä uutta tietoa työ tarjosi kokonaisuutena ja mitä työstä opittiin.

2 Päivitettävän web-sovelluksen arkkitehtuuri

Tässä luvussa tutustutaan insinöörityön web-sovelluksen järjestelmän tärkeimpiin osiin ja teknologioihin sekä siihen, miten ne sopivat yhteen ja muodostavat järjestelmäkokonaisuuden. Web-sovelluksessa oli muutamia pienempiäkin osia, joihin ei tarkemmin tässä työssä tutustuta, sillä ne eivät ole olennaisia tasonnoston kannalta.

Yksinkertaistettuna järjestelmän arkkitehtuuri oli kuvan 1 mukainen. Se voidaan jakaa neljään eri tasoon kuvan mukaisesti: käyttöliittymätasoon, logiikkatasoon, ORM-tasoon ja tietokantatasoon.



Kuva 1. Järjestelmän yksinkertaistettu arkkitehtuurimalli.

Käyttöliittymätaso koostuu teknologioista, jotka muodostavat web-sovelluksen käyttöliittymän eli XHTML-rakennekielestä ja ICEfaces-kirjastosta. XHTML (eXtensible Hypertext Markup Language) luo web-sovelluksen sivujen rakenteen, johon upotetaan ICEfacesin tarjoamia valmiskomponentteja.

Logiikkatasolla suoritetaan web-sovelluksen ohjelmistoprosessit eli keskustelu käyttöliittymätason kanssa siten, että käyttöliittymän toiminnalliset elementit kuten painikkeet sekä kenttien arvojen vaihtumiset päivittävät JSF-beanien muuttujia ja käynnistävät toiminnallisuuksia logiikkatasolla.

JSF-beanit, joiden muuttujia käyttöliittymätaso päivittää, ovat kuten tavalliset Java Bean luokat. Ne toteuttavat Serializable-rajapinnan vaatimukset ja siten beanit pystytään muuttamaan tavuvirraksi. Tämän lisäksi kyseiset beanit on rekisteröity JSF:n hallitsemiksi, joko luokan sisäisillä annotaatioilla tai XML-tiedoston konfiguroinnilla, joka tekee niistä JSF-beaneja (1).

Käyttöliittymätason käynnistämät logiikkatason toiminnallisuudet päivittävät käyttöliittymätasolla kenttiä ja tarvittaessa keskustelevat tietokantatason kanssa. Tietokantatason keskustelu tapahtuu beanien DTOMappereilla. Ne kutsuvat ORM-tasolla sijaitsevia DAO-luokkia keskustelemaan tietokannan kanssa.

ORM (Object-Relational Mapping) -taso on järjestelmätaso, jonka tarkoitus on tehdä tietokannan oliomallinnus sekä keskustelu itse tietokannan kanssa. Se sisältää Data Transfer Object (DTO) -luokat. Ne ovat siis tietokannan tauluista generoituja luokkia, joiden muuttujat vastaavat tietokannan taulujen kenttiä. Tämän lisäksi ORM-tasolla on myös Data Access Object (DAO) -luokat. DAO-luokat sisältävät valmiita kyselyjä ja toiminnallisuuksia, joilla voidaan hakea ja muokata tietokannan tauluja. DTO- ja DAO-luokkien toiminta perustuu ohjelmistokehys Hibernateen, joka suorittaa DAO-luokkien kyselyt ja päivittämällä DTO-luokkien arvoja.

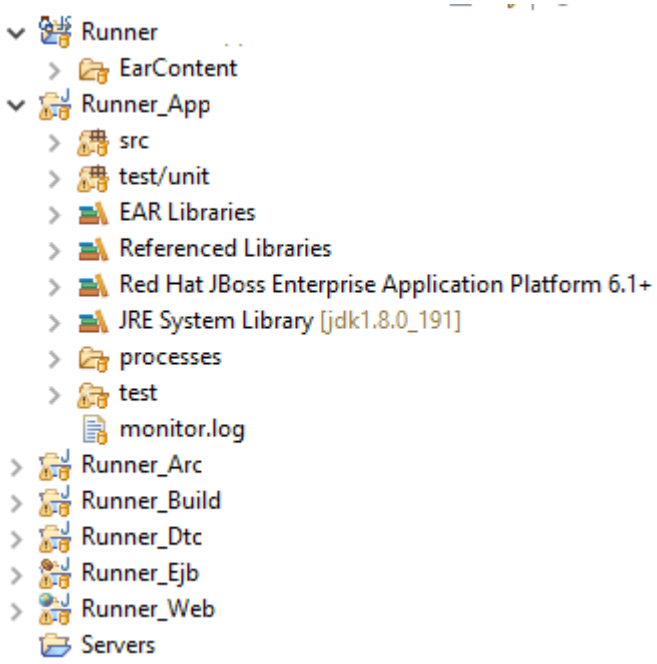
Tietokantataso koostuu IBM Database 2 -relaatiotietokannasta, joka sisältää web-sovelluksen tiedot tauluissa. Tämä taso eroaa muista tasoista siten, että se ei suoranaisesti ole kiinni web-sovelluksessa, vaikka onkin olennainen osa sitä, mikä tarkoittaa sitä, että se isännöidään toisella palvelimella ja siihen otetaan yhteys web-sovelluksesta Hibernatella.

Rakenteellisesti järjestelmä koostui kuvan 1 mukaisesti Java Platform Enterprise Edition -sovelluskehitysalustasta, johon liitetään IBM:n Database 2 -tietokanta Javan Hibernate-ohjelmistokehyksen avulla. Web-sovellus isännöidään JBoss Java Enterprise Application Platform -palvelimella. Näiden teknologioiden lisäksi järjestelmässä hyödynnetään IceFaces-kirjastoa sovelluksen käyttöliittymän luonnissa. Kaikkiin edellä mainittuihin teknologioihin ja siihen, miten ne liittyvät eri järjestelmän tasoihin tutustutaan tarkemmin seuraavissa alaluvuissa.

2.1 Java Platform Enterprise Edition

Insinöörityössä käytettävän web-sovelluksen pohjana toimi Java EE (Java Platform Enterprise Edition). Java EE on yritystason sovelluskehitysalusta, jota hyödynnetään usein palvelinpohjaisten Java-sovelluksien kehityksessä. Se laajentaa Java SE:tä (Java Platform Standard Edition) tarjoamalla hyödyllisiä sovelluskehitysrajapintoja sekä spesifikaatioita verkkopalveluille ja hajautetulle tietojenkäsittelylle. (2; 3.) Tämä tekee siitä huomattavasti toimivamman ratkaisun palvelinpohjaisten sovellusten kehittämiseen, ja se on myös syy siihen, miksi projektissa käytetään Java EE:tä eikä Java SE:tä.

Java EE muodosti projektin rakenteen ja pohjan, kuten kuvassa 1 näkyy. Siinä Java EE sekä JBoss-palvelin sisältävät koko käyttöliittymä-, logiikka- ja ORM-tason. Käytännössä tämä tarkoittaa siis sitä, että Java EE tarjosi kuvan 2 muotoisen rakenteen projektille, jossa projektin osat on jaettu Runnerin osioihin, ja JBoss-palvelin puolestaan suorittaa kyseiset osat.



Kuva 2. Web-sovelluksen rakenne.

Web-sovelluksen rakenne jakautuu siten, että jokainen Runner-kansio toteuttaa oman toiminnallisuutensa. Runner-kansiossa oleva EarContent sisältää projektin käyttämät kirjastot. Runner_App puolestaan sisältää kaikki loogiset prosessit ja toimii ylipäätänsä web-sovelluksen pääkansiona. Runner_Arc muodostaa projektin ohjelmistollisen arkkitehtuurin eli rajapinnat ja yläluokat, joita voidaan sitten helposti laajentaa. Runner_Buildiin rakentuu projektin koontiversio eli valmis ohjelma. Runner_Dto sisältää tietokannan hallinnan luokat. Runner_Ejb sisältää Ejb (Enterprise Java Beans) -luokat. Runner_Web sisältää kaikki verkkopuolen luokat ja XHTML-tiedostot. Näiden Runner-kansioiden sisältöön ei insinöörityössä tarkasti tutustuta, sillä niiden ymmärrys ei ole tärkeää tasonnoston kannalta.

2.2 JBoss Enterprise Application Platform

Projektissa käytetty palvelin oli Red Hat JBoss Enterprise Application Platform 6.4 (JBoss EAP 6.4). Kyseinen palvelin on Red Hat -ohjelmistoyrityksen kehittämä yritystason palvelinalusta, jolla voidaan isännöidä Java EE -pohjaisia sovelluksia. JBoss EAP 6.4 on yksi teknologioista, joka haluttiin insinöörityössä tasonnostossa päivittää versioon

7.1. Päivitys haluttiin tehdä, sillä JBoss EAP 6.4 on saavuttanut End of Life -tilan ja sitä ei enää tueta tai päivitetä.

JBoss-palvelimen tarkoitus web-sovelluksen osana on se, että se suorittaa web-sovelluksen Java-koodin, kuten kuvassa 1 näkyy, jossa JBoss EAP on sijoitettu oikealle sisältämään käyttöliittymä-, logiikka- ja ORM-tason. Palvelimen käynnistäminen käynnistää koko järjestelmän ja tuo käyttöliittymätason XHTML-sivut käyttäjän saavutettavaksi selaimella. Tämän lisäksi JBoss hallitsee kaikkia logiikkatason ohjelmaprosesseja siten, että se suorittaa kyseiset prosessit käyttäjän syötteen perusteella. Samalla se hallitsee ORM-tason yhteyttä tietokannan kanssa.

2.3 IBM Database 2

Web-sovelluksen taustalla toimi IBM Database 2 (IBM Db2) -tietokanta. Db2 on IBM:n alun perin vuonna 1983 julkaisema SQL-kieltä hyödyntävä relaatiotietokanta, joka toimii hyvin samantyyppisesti kuin muutkin relaatiotietokannat, kuten esimerkiksi Microsoftin SQL Server.

Db2:n tarkoitus osana web-sovellusta on selkeä. Se on tietokantajärjestelmä eli se pitää sisällään web-sovelluksen tietokantarakenteen sekä kaiken tiedon, jota web-sovelluksessa käytetään.

Tasonnoston kannalta tietokanta ei ollut kovinkaan olennainen. Tämä johtui siitä, että sitä ei haluttu päivittää ja ainut siihen yhteydessä oleva teknologia oli Hibernate, jota ei myöskään haluta päivittää.

2.4 Hibernate

Hibernate on Java-kirjasto, joka tarjoaa ohjelmistokehyksen tietokantayhteyden muodostamiseksi sekä oliomallinnukseen (4). Työkalu implementoi Java Persistence API (JPA) -ohjelmointirajapintaa, jonka Java EE tarjoaa ja siten muodostaa helposti yhteyden melkein mihin tahansa relaatiotietokantaan. Projektissa oli käytössä Hibernate 5.1, jota käytettiin muodostamaan yhteys sekä kommunikoimaan IBM Db 2 -tietokannan kanssa.

Arkkitehtuuritasolla Hibernate hallitsee ORM-tasoa ja suorittaa DAO-luokkien kyselyt tietokantaan. Hibernate kuvaa tietokannan vastaukset DTO-luokan olioiden muuttujiin sekä päinvastoin se voi päivittää tietokantaan DTO-luokan olioiden muuttujista saadut arvot. Pääasiassa se siis tekee tietokannan ja sovelluksen välisen keskustelun.

2.5 XHTML, JSF ja ICEfaces käyttöliittymätason perustana

Käyttöliittymätason sivujen pohjana ja rakenteena oli XHTML (eXtensible Hypertext Markup Language). XHTML on kuten tavallinen HTML (Hypertext Markup Language), mutta se täyttää XML-kielen vaatimukset. Tämä tarkoittaa sitä, että elementit on sisennettävä ja suljettava oikein sekä kaikki elementtien nimet, ja ominaisuudet on kirjoitettava pienillä kirjaimilla. Syy tiukoille vaatimuksille on W3Schoolsin mukaan se, että XHTML-koodin tulkitseminen vaatii vähemmän prosessointitehoa, sillä selainten ei tarvitse yrittää tulkita huonosti kirjoitettua HTML-koodia (5).

XHTML tukee JavaServer Faces (JSF) -ohjelmistokehystä, joka tarjoaa helppokäyttöisiä ja dynaamisia käyttöliittymäkomponentteja. Projektissa ei tosin käytetty suoraan JSF-ohjelmistokehystä vaan ICEfacesia, joka on JavaServer Facesia laajentava ohjelmistokehityspaketti.

ICEfaces laajentaa JSF:ää lisäämällä uusia komponentteja, jotka sisältävät sisäänrakennetun Ajax-tuen siten, että käyttöliittymän arvojen muuttaminen automaattisesti päivittää järjestelmäpuolen muuttujien arvoja sekä päinvastoin. ICEfacesillä tärkeimpänä ominaisuutena on se, että sillä voidaan toteuttaa helposti Rich Internet Applicationeja (RIA) web-sovelluksia, jotka käyttäytyvät kuten työpöytäsovellukset (6).

Projektissa ICEfacesiä käytettiin sovelluksen käyttöliittymän luonnissa siten, että järjestelmän XHTML-tiedostot koostuivat pääasiassa ICEfaces-elementeistä. Se oli olennainen tasonnoston kannalta, sillä sen versio on 1.8.2, joka on tuettu JBoss EAP 6.4 -palvelimessa, mutta ei myöhemmissä versioissa.

3 Tasonnoston tarkoitus

Tässä luvussa käsitellään tasonnostoa ja sitä, mikä oikeastaan on tasonnosto sekä miksi se ylipäätänsä tehdään. Aliluvuissa selvitetään ensin yleisellä tasolla syitä, miksi tasonnostoja tehdään, jonka jälkeen käydään läpi insinöörityön web-sovelluksen tasonnoston syitä ja päämääriä.

Järjestelmän tasonnosto käsitteenä tarkoittaa käytännössä prosessia, jossa järjestelmän osakomponentteja ja ohjelmistoja päivitetään sekä tuodaan ajan tasalle. Prosessi alkaa selvittämällä, onko tasonnostolle ylipäätänsä tarvetta, jonka jälkeen tulee pohtia, miten tasonnosto toteutetaan ja mitä kannattaa päivittää.

Toteutustavan valitsemisen jälkeen lähdetään toteuttamaan tasonnostoa, jonka toteuttamisessa voi kestää paljon tai vähän aikaa riippuen suunnittelusta ja valitusta toteutustavasta. Tämän takia on järkevää käyttää paljon aikaa itse suunnittelutyössä, jotta voi säästää aikaa toteutuksessa.

3.1 Tasonnoston syyt

Pääasiallinen syy, miksi tasonnostoja tehdään, on varautuminen järjestelmäkehityksen tulevaisuutta varten ja estämään legacy-järjestelmän muodostumista. Legacy-järjestelmällä tarkoitetaan järjestelmää, joka sisältää vanhentunutta teknologiaa, joka käytännössä vielä toimii, mutta usein on hyvin hankala ylläpitää ja laajentaa. Legacy-järjestelmän ylläpitämistä hankaloittaa vanhan dokumentaation puute ongelmanratkaisutilanteissa. Laajentamista puolestaan hankaloittaa teknologian yhteensopimattomuus uudempien teknologioiden kanssa. Vanhentuneille järjestelmille voi olla myös vaikea löytää uusia kehittäjiä ja ylläpitäjiä, sillä vanhoja teknologioita ei enää opeteta käyttämään.

Tasonnoston välttäminen kasvattaa teknistä velkaa jatkuvasti niin kauan kuin sitä vältetään, sillä vanhentuneiden ohjelmistojen päivittäminen hankaloituu aina uusien ohjelmistojen ja ohjelmistopäivitysten myötä. Syy siihen, miksi tasonnostoa usein vältetään, on se, että yritykset eivät halua käyttää budjettia ohjelmistoihin, jotka toimivat sillä hetkellä tai päivityksiin, joista ei ole suoraa hyötyä, vaikka ne estäisivät hankaluuksia

tulevaisuudessa (7). Usein tasonnostoprosessiin kuuluukin parhaimman ratkaisun etsiminen ja sen selostaminen asiakkaalle järkevänä päätöksenä.

Järjestelmän tasonnosto voidaan toteuttaa myös siitäkin syystä, että uusista ohjelmistoversioista löytyy haluttuja ominaisuuksia ja mahdollisuuksia, joita vanhat versiot eivät kykene tarjoamaan.

Tasonnosto on siis hyvinkin tärkeä prosessi, koska tietotekniikan ala on jatkuvassa muutoksessa ja uusia teknologioita ilmestyy jatkuvasti, mikä luo uusia mahdollisuuksia ja haasteita vanhoille järjestelmille. Järjestelmät, joita ei päivitetä kärsivät korkeammista ylläpitokuluista sekä aikamääreistä ja voivat loppujen lopuksi joutua hylätyiksi.

3.2 Tasonnosto insinöörityön web-sovelluksessa

Web-sovelluksen arkkitehtuuriluvussa käytiin läpi insinöörityön web-sovelluksen arkkitehtuuria ja järjestelmän osia, mutta ei erityisen tarkasti tutustuttu siihen, mille web-sovelluksen osille tasonnosto toteutettiin. Kyseisessä web-sovelluksessa oli kaksi osaa, jolle asiakas kaipasi tasonnostoa ja päivitystä. Nämä olivat alla olevan taulukon 1 mukaisesti web-sovelluksen palvelin JBoss EAP 6.4 sekä käyttöliittymän ohjelmistokehityspaketti ICEfaces 1.8.2.

Taulukko 1. Tasonnostossa tehtävät päivitykset.

Päivitettävä osa	Nykyinen versio	Uusi versio
JBoss EAP	6.4	7.1
ICEfaces	1.8.2	3

JBoss EAP haluttiin päivittää, sillä JBoss EAP 6.x -sarjan tuki loppui vuoden 2019 kesäkuussa. Tämä tarkoittaa sitä, että JBoss EAP 6.x ei saa enää ylläpitotukea tai korjauspäivityksiä ja jää Legacy-teknologiaksi. Tästä syystä palvelin haluttiin päivittää uuteen versioon 7.1, joka saa tukea vielä vuoteen 2023. (8.)

Käyttöliittymän ohjelmistokehityspaketti ICEfaces 1.8.2 on myös EOL eli End of Life -tilassa, jossa sitä ei enää päivitetä (9). Tosin tämä ei ollut tärkein syy päivitykselle vaan se, että JBoss EAP ei enää päivityksen jälkeen tarjoa tukea ICEfaces 1.8.2:lle, joten ne

eivät täysin sovi yhteen. Tämän takia ICEfaces haluttiin päivittää versioon 3, joka on JBoss EAP 7.1 tukema sekä modernimpi kuin ICEfaces 1.8.2 (10).

4 Tasonnoston toteutustapoja

Tasonnostosta tekee hankalaa se, että sille ei ole selvää toteutustapaa, sillä jokainen järjestelmä on erilainen. Tämä johtaa siihen, että järjestelmän ylläpitoryhmän on käytettävä työaikaa selvittääkseen, onko tasonnosto järkevä toteuttaa senhetkisessä tilanteessa, ja tämän jälkeen, miten se kannattaisi toteuttaa.

Tässä luvussa tutustutaan siihen, mitä toteutustapoja insinöörityön web-sovelluksen tasonnostolle oli. Tasonnoston toteutustavoista selvitettiin niiden hyödyt ja haitat sekä pohditaan, mitä kompromisseja oli tehtävä.

Pääasiallisina tavoitteina tasonnostolle oli siis JBoss EAP 6.4 -palvelimen päivitys versioon 7.1 ja ICEfacesin päivitys versiosta 1.8.2 versioon 3. Tosin molempien päivittäminen olisi hankalaa, sillä vaadittu työmäärä olisi huomattavasti suurempi kuin vain JBossin päivittäminen. Näistä toteutustavoista web-sovelluksen asiakas päätti, miten halutaan edetä ja sen käytännön toteutukseen tutustutaan seuraavassa luvussa.

4.1 ICEfaces-päivitys ja JBoss-päivitys

Ensimmäisenä toteutustapana tutustuttiin ICEfacesin ja JBossin päivittämiseen sekä siihen, miten se voitaisiin käytännössä toteuttaa. ICEfaces tuli ensin päivittää versioon 3, sillä JBoss EAP 7.1 ei tue ICEfacesin aikaisempaa versiota 1.8.2. Tämän jälkeen voitaisiin päivittää JBoss EAP 6.4 versioon 7.1.

ICEfacesin päivittäminen ei ole teoriassa kovinkaan monimutkainen prosessi, vaan se eteni melko yksinkertaisesti kuten alla olevassa kuvassa 3. Päivittäminen aloitettiin seuraamalla ICEsoft technologiesin tarjoamaa päivitysohjetta: ”Converting ICEfaces 1.8 Applications to ICEfaces 3 Tutorial”. Ohje kertoo olennaiset vaiheet, jotka on toteutettava, kun halutaan päivittää ICEfacesia hyödyntävä sovellus versiosta 1.8.x versioon 3. (11.)

Ohjeen seuraamisen jälkeen siirryttiin projektikohtaisiin vaiheisiin, jotka havaittiin tarpeellisiksi päivityksen yhteydessä.

```
Converting ICEfaces 1.8 Applications to ICEfaces 3 Tutorial vaiheet
1. Vanhojen kirjastojen poisto
2. Uusien kirjastojen lisäys
3. Konfiguraatiotiedostojen päivitys
3.1. faces-config.xml päivitys
3.2. web.xml päivitys
4. XHTML-sivujen merkkauksen päivitys

Projektikohtaiset ylimääräiset vaiheet
5. RenderManager API:n korvaus ICEpush PushRenderillä
6. D2DViewHandlerin poisto
7. InputFile toiminnallisuuden korvaus FileEntryllä
8. Web-sovelluksen käyttöliittymän ID virheiden korjaus
9. Kuvien ja muiden web-resurssien tiedostosijaintien ja -polkujen päivitys
```

Kuva 3. ICEfaces-päivitysvaiheet.

Päivitys alkoi vanhojen kirjastojen ja viittauksien poistamisella eli kaikkien ICEfaces 1.8.2 ja JSF 1.2 -tiedostojen poistamisella projektista. JSF 1.2 -viittaukset poistettiin, sillä ICEFaces 3 käyttää JSF 2 -kirjastoja. Tämä tarkoitti insinöörityön web-sovelluksen tapauksessa sitä, että poistettiin aikaisemmin esitellyn Runner- sekä Runner_Web-kansioiden sisältä kaikki ICEfaces- sekä JSF-tiedostot.

Kun kaikki aikaisempien versioiden viittaukset olivat poistettu, edettiin seuraavaan vaiheeseen, jossa ladattiin ja asennettiin kaikki ICEfaces 3- ja JSF 2 -kirjastot. Lataaminen tapahtui ohjeen mukaisesti ICEfacesin sivustoilta. Lataamisen jälkeen ladatut ICEfaces 3 ja JSF 2 .jar -tiedostot sijoitettiin Runner_Web-kansion sisälle kirjastoille varattuun tilaan.

Kirjastojen päivitykset voivat olla hankalia kuten tässä tapauksessa, sillä projekti sisälsi niin monia eri kirjastoja, että oli vaikeaa selvittää, mitkä kirjastot tulisi poistaa vanhoja kirjastoja poistaessa. Tämän lisäksi oli otettava huomioon, että uudet kirjastot sijaitsivat oikeissa paikoissa ja niihin viitattiin oikein, jotta kaikki järjestelmän osat löysivät uudet kirjastot oikein. Prosessin toteuttaminen ei siis ollut niin suoraviivaista kuin ohje antaisi olettaa, varsinkaan kun prosessi toteutettiin isolle projektille (noin 80000 tiedostoa tai kooltaan noin 2gb), kuten tässä insinöörityössä.

Kun kirjastot oli päivitetty, päivitettiin niiden konfiguraatiotiedostot. Tämä aloitettiin ”faces-config.xml” -tiedoston päivittämisellä. Tämä tiedosto on Java Server Facesin eli JSF:n konfiguraatiotiedosto. Muutokset olivat melko minimaalisia ja koostuivat pääasiassa XML-tiedoston ”headerin” päivittämisestä versiosta 1.2 versioon 2 sekä ”d2dviewhandlerin” poistamisesta. ICEfaces 3 tekee ”d2dviewhandlerin” toiminnan automaattisesti. Loppujen lopuksi vaihdettiin vielä ”managed-bean-scope”-ominaisuuden arvoksi ”view” arvosta ”request”. Tämä päivitti ”beanin” elinaikatoiminnallisuuden siten, että se kestää koko JSF-näytön elinajan verran eikä vain yhden http-pyynnön verran. Muutos oli tehtävä, sillä JSF 2 tukee ”ViewScopea” eikä ”RequestScopea”.

Seuraava vaihe oli ”web.xml”-tiedoston päivitys. ”web.xml” on Java EE -sovelluksen xml-konfiguraatiotiedosto. Tiedoston parametrit kertovat palvelimelle kaiken, mitä sen tulee tietää web-sovelluksesta. Pääasiassa ”web.xml” määrittelee Java Servlettejä, jotka ovat sovelluksen osia, joilla voidaan laajentaa palvelimen toimintaa.

Muokkaus alkoi poistamalla ”web.xml”-tiedostosta kaikki vanhat servletit sekä palvelimen kuuntelijat. Tämän jälkeen muokattiin Faces Servlettiä. Sen tarkoitus on yhdistää palvelin ja JSF-toteutus toisiinsa. Syynä tähän oli se, että ICEfaces 1.8 ja 3:n välillä sen toiminta on hieman muuttunut. Tämän takia se vaati uusia parametreja, välittämään tiedon JSF:stä sekä ICEfacesta. Lopuksi ”web.xml”-tiedostosta poistettiin turhia kontekstiparametreja, joita ei enää tarvittu ja muokattiin Faces Servlettien eli Facelettien päivitysparametri epävirallisesta viralliseen.

Kun kaikki konfiguraatiotiedostot olivat päivitetty, siirryttiin ohjeen neljänteen ja viimeiseen vaiheeseen eli XHTML-sivujen merkkauksen muokkaukseen. Merkkauksen muokkaus oli tehtävä, koska JSF-kirjoitusstandardit olivat muuttuneet versiosta 1.2 versioon 2 siirryttäessä.

Sivujen merkkauksen muokkaus alkoi vaihtamalla kaikkien sivujen ”.iface”-pääte muotoon ”.jsf”. Tämä tehtiin ”web.xml”-tiedostossa, josta löytyvät viittaukset kaikkiin sovelluksen sivuihin. Kyseisiä viittauksia oli noin 300 kappaletta, mutta se ei ollut liian suuri ongelma. Vaihto voitiin toteuttaa yksinkertaisella etsintä- ja korvausoperaatiolla, joka löytyy helposti monista ohjelmointiympäristöistä sekä tekstieditoreista.

Päätteen muokkauksien jälkeen siirryttiin yksittäisten sivujen merkkausten muokkaukseen. Merkkauksen muokkauksessa oli päivitettävä sivujen header lisäämällä sinne uusia viittauksia, jolla saatiin JSF- ja ICEfaces-komponenttien tiedot sivujen käyttöön. Tämän lisäksi oli päivitettävä sivujen body-elementit viittamaan näihin uusiin komponenttiviittauksiin.

Viittauksien päivitys oli teoriassa yksinkertaista, sillä se toteutettiin jokaisella sivulla samalla lailla. Käytännössä tämä oli hieman isompi prosessi, koska jokainen sivu on kuitenkin muokattava erikseen ja sivut ovat erilaisia. Tämän takia sivut oli käytävä yksitellen läpi ja tehtävä tarvittavat päivitykset niihin. Prosessi ei olisi ollut teknisesti haastava, mutta hyvin paljon aikaa vievä, sillä sivuja on kuitenkin noin 300. Koska ICEfaces päivityksen toteutuksesta ei ollut varmuutta, toteutettiin viittauksien päivitys vain yhdelle sivulle alustavasti, jotta päivitys pystyttiin testaamaan.

Viittausten päivityksen jälkeen suoritettiin ohjeen viimeinen vaihe, jossa vaihdettiin pois ice:outputDeclaration-elementti. Kyseisen elementin tarkoitus on ilmoittaa dokumentin metatietoja, kuten dokumentin tyyppi. Se on vanhentunut ICEfaces 3:ssa ja se piti vaihtaa yleiseen DOCTYPE-elementtiin.

Vaikka ICEfacesin päivitysohje olikin suoritettu loppuun, löytyi projektista vielä ongelmia, joihin ohje ei ottanut kantaa. Nämä ongelmat huomattiin tasonnoston pienikokoisessa testauksessa, jossa alustavasti testattiin ICEfacesin päivittämistä tasonnoston toteutustapana. Kyseisiä ongelmia oli monia erilaisia, jotka ilmenivät käytetyn Eclipse-ohjelmointiympäristön löytäminä virheinä sekä palvelimen käynnistyksen yhteydessä. Tämän lisäksi tutustuttiin ICEfacesin versioiden väliseen yhteensopivuusdokumentaatioon ”ICEfaces 1.x Compatibility Features of ICEfaces 3” (12).

Ensimmäisenä ongelmana huomattiin, mitä ominaisuuksia ICEfaces 3 ei sisällä, joita ICEfaces 1.8.2 sisälsi. Tämä tapahtui jo ennen palvelimen käynnistämistä ja havaittiin Eclipsen varoituksista. Koska Eclipse on dynaaminen ohjelmointiympäristö, niin se huomaa jo kehityksen aikana, mikäli järjestelmän koodissa on viittauksia tai riippuvuuksia kirjastoihin, joita ei projektista löydy.

Eclipsen varoituksien mukaan projektista ei enää löytynyt luokkia RenderManager, D2DViewHandler sekä InputFile. RenderManager on luokka, jonka avulla voidaan

kutsua käyttöliittymän renderöintipalveluita palvelinpuolelta (13). D2DViewHandler puolestaan on ICEfacesin ratkaisu JSF:n ViewHandler-abstraktiluokan toteutukseksi. ViewHandler on siis JSF:n tarjoama abstraktiluokka, jonka avulla voidaan toteuttaa omia ratkaisuja käyttöliittymän renderöintiin sekä päivittämiseen (14). Molemmat puutteet liittyivät siihen, miten palvelin käsittelee ja päivittää käyttöliittymää. Viimeisenä InputFile-luokka liittyi ICEfacesin tarjoamaan InputFile-elementtiin. Kyseisellä elementillä voidaan siirtää käyttäjän tietokoneelta tiedostoja palvelimelle.

Selvitystyön jälkeen havaittiin, että RenderManager API:a ei enää tuettu ICEfacesin version 1.8.2 jälkeen. Mikäli sen toiminnallisuutta haluttiin toistaa, oli käytettävä ICEpush-kirjaston tarjoamaa PushRendereriä (15). Tämän toiminnan päivitykseen ei vielä tarkemmin tutustuttu, koska ei ollut varmuutta siitä, haluaako asiakas päivittää ICEfacesin, ja tarkoituksena oli vain selvittää, kuinka raskas prosessi ICEfacesin päivitys on.

D2DViewHandlerin puuttumiseen tutustuttiin RenderManager-ongelman jälkeen. Tähän löytyi vastaus aikaisemmin mainitusta ”ICEfaces 1.x Compatibility Features of ICEfaces 3” -dokumentaatiosta. Siinä mainittiin, että ICEfacesin versiossa 1.8.x oli pakollista määritellä oma ViewHandler-sovelluksille, joissa tahdotaan hyödyntää Facelettejä. Tämän takia sovellukseen oli lisätty D2DViewHandler, joka toteutti ViewHandlerin toiminnan. Dokumentaatio mainitsi tämän jälkeen, että tätä ei enää tarvita ja siihen tehdyt viittaukset tulisi poistaa sovelluksesta, sillä sitä ei enää ole. D2DViewHandler yksinkertaisesti vain poistettiin sovelluksesta ja sen poistosta ei alustavasti huomattu olevan mitään haittaa.

InputFile-luokan katoamisen selvittäminen oli olennaista sovelluksen toiminnallisuuden kannalta, sillä sovellukseen pitää pystyä lisäämään käyttäjän toimesta tiedostoja. Tähänkin ongelmaan löytyi ratkaisu ”ICEfaces 1.x Compatibility Features of ICEfaces 3” -dokumentaatiosta. Ongelmana syynä oli se, että InputFile-elementti on korvattu FileEntry-elementillä ICEfaces 3:ssa. Tosin saman lailla kuten muissakin uusissa luokissa FileEntry ei toimi täysin samalla tavalla kuin InputFile ja se vaatii koodin muuttamista, joka voi olla hankalaa ja aikaa vievää.

Inputfilen vaihtaminen vaatii 16 sivun muokkaamisen käyttöliittymä- ja logiikkatasolla. Ongelmana tässä oli se, että jokainen sivu toimii eri tavoin ja siksi vaatii erilaisen ratkaisun. Tämäkin jätettiin teoreettiseksi korjaukseksi vielä tässä vaiheessa, sillä ei ole varmuutta, haluaako asiakas päivittää ICEfacesin.

Eclipsen löytämien virheiden korjaamisen jälkeen tarkasteltiin itse sovelluksen ja palvelimen käynnistystä ja mikäli siinä nousee ongelmia. Palvelimen käynnistäminen kaatui oletetusti lukuisiin virheisiin, jotka kirjattiin ylös kuvan 4 mukaisesti.

```
Havaittuja ongelmia palvelimen käynnistyksessä ja järjestelmän käytössä:
- Kaatuu BaseBean metodiin NPE, jokin IceFaces 3:ssa tai JSF 2 lisää elementtejä DOM:iin, joilla ei ole ID:tä.
-> Muokatttava checkComponentId metodia BaseBeanissa huomioimaan mahdolliset Null elementti ID:t.
- Duplicate id:eitä. Icefaces 3 ei salli duplicate elementti id:eitä.
-> Selvitettävä näytöiltä, että niissä ei ole duplicate id:eitä. (Tähän olisi hyvä keksiä jokin tapa selvittää millä näytöillä on duplicateja, muuten työläs)
- Main-templatessa kaksi "environment" id outputtextia -> nimetään "environment_name" ja "environment_version"
- Tiedostopoluissa vikaa, jonka takia kuvat eivät löydy, koska JSF 2 käsittelee tiedostoja ja polkuja erilailla kuin aikaisemmin.
  Mahdollisesti korjattavissa siirtämällä kaikki webcontent kansion resurssit (css, xmlhttp yms.) yhden "resources" kansion alle webcontent kansioon.
  Tämä ainakin korjasi sen, että CSS löytää kuva tiedostot, mutta yksittäiset näytöt vaativat "resources" lisäyksen tiedostopolkuun (testattu S096).
- zip filen avaus ongelma -> SEVERE virhe palvelimen käynnistyessä, mutta ei nähtävästi vaikuta toimintaan.
(Vaikuttaisi liittyvän jotenkin IceFaces 3 ja JSF 2 väliseen yhteyteen. Mahdollisesti jokin version mismatch)
```

Kuva 4. Palvelimen käynnistyksen virheiden havaintoloki.

Ensimmäinen ongelma nousi käyttöliittymän BaseBeanin eli käyttöliittymän perussivun luonnissa. Tässä ongelmana oli se, että BaseBeanin metodi checkComponentId ei ottanut huomioon käyttöliittymän elementtejä, joilla ei ole ID:tä, sillä näitä ei ollut ICEfaces 1.8.2:ssa. Tämä johtui siitä, että ICEfaces 3 tai JSF 2 luo omia elementtejä, joilla ei ole ID:tä. Ratkaisuna tähän laadittiin yksinkertainen korjaus, joka tarkisti elementtien ID:n puuttumisen.

ID-korjauksesta edettiin uuden ongelman diagnosointiin ja korjaukseen, jossa ICEfaces 3 kaatui siihen, että jotkut sivuista sisälsivät elementtejä, joilla oli identtiset ID:t. Nämä olivat vielä sallittuja (jos niitä ei renderöity samaan aikaan) ICEfaces 1.8.2:ssa, vaikkakin kyseenalaisia toteutuksia. Alustavasti tämä korjattiin vain aloitussivulta, mutta se olisi korjattava jokaiselta sivulta, jos sivujen ei haluta kaatuvan.

Identifikaatio-ongelmien jälkeen palvelin saatiin käynnistettyä, mutta jo ensimmäisellä sivulla havaittiin uusi ongelma. Sovellus ei enää näyttänyt minkäänlaisia kuvia. Tämän ongelman tutkittiin johtuvan siitä, että JSF 2 ei käsittele enää tiedostopolkuja samalla lailla, vaan suhteellinen tiedostopolku alkaa eri paikasta kuin JSF 1.2:ssa. Tämä johtaa siihen, että sovellus ei enää löydä kuvia annetuista tiedostopoluista. Tähän ongelmaan ei löytynyt yksinkertaista ratkaisua. Alustavasti ratkaisuksi suunniteltiin, että kuvat siirrettäisiin uuden ”resources” kansion alle ja lisättäisiin tiedostopolkuihin ”resources” alkuun. Tätäkään ratkaisu ei toteutettu loppuun asti, sillä ICEfaces-päivityksestä ei ole varmuutta.

Viimeisenä havaintona huomattiin käynnistyksen yhteydessä virhelokitus. Se varoitti jonkin zip-tiedoston avauksesta. Kyseisen ongelman ei kuitenkaan havaittu vaikuttavan

palvelimen käynnistymiseen tai sovelluksen toimintaan. Virhe vaikutti liittyvän ICEfaces 3 ja JSF 2:n väliseen yhteyteen. Tähän ei löytynyt ratkaisua.

Lopuksi ICEfacesin päivityksestä luotiin alustava tekstidokumentti. Dokumentissa käytiin läpi päivityksen yhteydessä esiintyneet virheet sekä niihin löytyneet ratkaisut ja miten itse ICEfaces-päivitys tapahtuu. ICEfacesin ja JBossin päivittäminen tasonnoston toteutustapana on siis hyvinkin mahdollinen, mutta todella raskas ja haastava toteuttaa. Tässäkin alaluvussa tutustuttiin vasta ICEfacesin päivitykseen eikä mainittu mitään JBossin päivittämisen toteutuksesta, sillä siihen asti ei päästy. Tämä johtui siitä, että asiakas päätti, että toteutus on liian aikaa vievä saatuun hyötyyn nähden. Tästä syystä tutustuttiin seuraavassa alaluvussa selostettuun toteutustapaan eli pelkän JBossin päivittämiseen.

4.2 Pelkkä JBoss-päivitys

Vaikka ideaali tapaus olisi, että päivitettäisiin molemmat ICEfaces sekä JBoss, voidaan suunnitella toinenkin toteutustapa. Tässä toteutustavassa päivitettäisiin vain JBoss ja kierrettäisiin ICEfacesin päivitys luomalla omatekoisia ratkaisuja siitä nouseviin ongelmiin. Ongelmana tässä on se, että omatekoiset ratkaisut eivät ole standardin mukaisia ja niitä on vaikea ylläpitää, mikäli niitä ei dokumentoida hyvin. Toteutustapa etenisi alla olevan kuvan 5 mukaisesti.

```
JBoss palvelimen päivitysvaiheet
1. JBoss Server Migration Tool -työkalun suoritus
2. JBoss EAP 7.1 palvelimen lisäys Eclipseen
3. JMX alijärjestelmän lisäys
4. JSF 1.2 ja Persistence API 2.1 lisäys uuden palvelimen moduuleihin
5. Asetetaan Eclipse kokoamaan palvelimen projekti yhteen arkistoon.
6. Lisätään riippuvuuksia JBossin kokoamiskonfiguraatioihin.
7. Päivitetään Eclipsen JBoss version kirjastot
```

Kuva 5. JBoss-palvelimen päivitysvaiheet.

Tässä toteutustavassa tasonnostoprosessi aloitettiin JBossin päivityksellä eli päivittämällä JBoss EAP 6.4 versioon 7.1. Palvelimen päivittämisessä hyödynnettiin Red Hatin tarjoamaa päivitysohjetta ”Migrating from Previous Versions of JBoss EAP to JBoss EAP 7.x” ja päivitystyökalua ”JBoss Server Migration Tool” (16, 17). Ohjeen pohjalta saatiin luotua kuvan 5 mukaiset etenemisvaiheet.

JBoss Server Migration Tool on Red Hat -yrityksen toteuttama Java-työkalu. Työkalun avulla voidaan JBoss-palvelin päivittää uuteen versioon siten, että palvelimelle tehdyt konfiguroinnit säilyvät. Työkalulla on kaksi toimintamuotoa: interaktiivinen ja ei-interaktiivinen. Interaktiivisessa muodossa avautuu työkalun käytön yhteydessä komentorivi. Interaktiivinen muoto antaa käyttäjälle vaihtoehtoja miten eri päivitysvaiheissa toimitaan, kuten mitkä konfiguraatiot halutaan siirtää ja mitkä puolestaan halutaan jättää. Ei-interaktiivinen tapa on päinvastainen ja suorittaa päivityksen oletusasetuksilla. JBoss Server Migration Tool -työkalun avulla palvelimen päivitys on helppo aloittaa. Se tekee suurimman osan työstä päivittämällä olennaiset palvelinasetukset uuden version mukaisiksi. Tosin palvelin vaatii tämän jälkeen vielä hienosäätöä, jotta se saadaan täysin toimimaan.

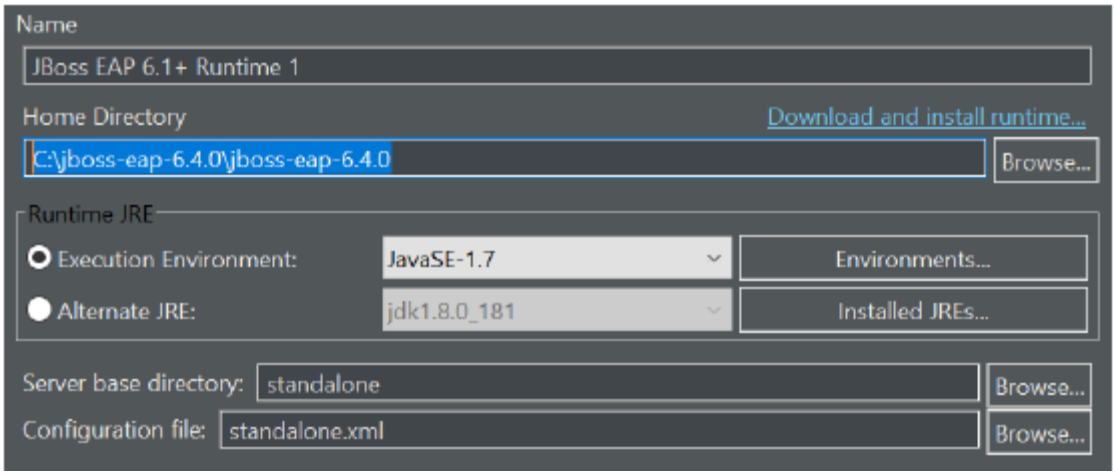
JBoss Server Migration Tool -työkalun käynnistys aloitetaan Windowsilla suorittamalla komentorivillä server-migration.bat-tiedosto. Tiedostolle annetaan kaksi käynnistysargumenttia ”source” ja ”target”. Nämä argumentit ovat vastaavasti päivitettävän palvelimen tiedostopolku ja kohdepalvelimen tiedostopolku. Projektin tapauksessa päivitettävä palvelin oli siis tällä hetkellä käytettävissä oleva palvelin JBoss EAP 6.4 ja kohdepalvelin oli uusi JBoss EAP 7.1-palvelin, jota ei ollut vielä konfiguroitu. Työkalu käynnistettiin interaktiivisessa muodossa, jotta voitiin käydä läpi kaikki päivityksen yhteydessä tapahtuvat muutokset. Tämän jälkeen työkalu aloitti päivitysprosessin ja esitti käyttäjälle kysymyksiä, joihin käyttäjän tuli vastata kyllä tai ei. Pääasiassa kysymykset pyysivät vain lupaa siirtää vanhoja konfiguraatioasetuksia uudelle palvelimelle, joten kaikkiin nouseviin kysymyksiin vastattiin kyllä. Kun päivitys oli valmis, saatiin alla olevan kuvan 6 mukainen tulosloki, jossa kerrotaan eri päivitysvaiheiden tulokset.

```
08:04:57,434 INFO  [org.jboss.migration.core.logger] (main)
--------------------------------------------------------------------------------------------------------------
 Task Summary
--------------------------------------------------------------------------------------------------------------

 server ..................................................................................... SUCCESS
  standalone ................................................................................ SUCCESS
   config-files ............................................................................. SUCCESS
    config-file(source=/Users/emmartins/wildfly/dist/jboss-eap-6.4/standalone/configuration/standalone.xml) .. SUCCESS
     subsystems-xml-config ................................................................... SUCCESS
      remove-extension(module=org.jboss.as.threads) .......................................... SUCCESS
      remove-subsystem(namespace=urn:jboss:domain:threads:1.1) ............................... SUCCESS
     subsystems-management-resources ......................................................... SUCCESS
      migrate-subsystem(name=web) ............................................................ SUCCESS
      update-subsystem(name=infinispan) ...................................................... SUCCESS
      update-subsystem(name=ee) .............................................................. SUCCESS
      update-subsystem(name=ejb3) ............................................................ SUCCESS
      update-subsystem(name=remoting) ........................................................ SUCCESS
      add-subsystem(name=batch-jberet) ....................................................... SUCCESS
      add-subsystem(name=bean-validation) .................................................... SUCCESS
      add-subsystem(name=singleton) .......................................................... SUCCESS
      add-subsystem(name=request-controller) ................................................. SUCCESS
      add-subsystem(name=security-manager) ................................................... SUCCESS
      update-subsystem(name=undertow) ........................................................ SUCCESS
      update-subsystem(name=messaging-activemq) .............................................. SUCCESS
     security-realms ......................................................................... SUCCESS
      security-realm(name=ApplicationRealm) .................................................. SUCCESS
      security-realm(name=ManagementRealm) ................................................... SUCCESS
     management-interfaces ................................................................... SUCCESS
      enable-http-upgrade-support ............................................................ SUCCESS
     socket-bindings ......................................................................... SUCCESS
      update-management-https ................................................................ SUCCESS

--------------------------------------------------------------------------------------------------------------
 Migration Result: SUCCESS
--------------------------------------------------------------------------------------------------------------
```

Kuva 6. JBoss Server Migration Tool -tulosloki.

Seuraavana vaiheena oli uuden palvelimen liittäminen web-sovellukseen. Vaikka JBoss Server Migration Tool ilmoitti migraation onnistuneen, tuli palvelimeen tehdä vielä manuaalista konfigurointia, jotta se saatiin toimimaan oikein web-sovelluksen kanssa. Tämä johtui siitä, että sovelluksen aikaisempi palvelin oli jo itsessään hieman muokattu oletusasetuksista sekä siitä, että JBoss EAP 7.1 ei tue ICEfaces 1.8.2:ta tai JSF 1.2:ta.

Palvelimen konfigurointi aloitettiin lisäämällä Eclipseen uusi palvelin kuvan 7 mukaisesti. Tosin JBoss EAP 6.4:n sijaan valittiin JBoss EAP 7.1. Tämän lisäksi oli huomioitava, että JBoss EAP 7.1 tarvitsee ajonaikaiseksi ympäristökseen Java Development Kitin eikä Java Platform Standard Editionin, vaikka JBoss EAP 6.4 toimii Java Platform Standard Editionilla. Paneelista oli siis vaihdettava Runtime JRE -valinta Execution Environmentista Alternate JRE:hen.



Kuva 7. Eclipse-ohjelmointiympäristön palvelimen lisäyspaneeli.

Uuden palvelimen konfiguroinnin ja asennuksen jälkeen yritettiin palvelimen käynnistystä. Palvelin ei kuitenkaan käynnistynyt, sillä siitä puuttui JMX-alijärjestelmä. Alijärjestelmät ovat Java EE:n toiminnallisia osia, jotka tarjoavat monipuolisia toimintoja, kuten lokitus- ja verkko-ominaisuuksia. JMX-alijärjestelmä puolestaan tarjoaa mahdollisuuden rekisteröidä verkkopalveluita sovellukseen (18).

Ongelmana oli se, että JMX-alijärjestelmä vaadittiin palvelimen osaksi, jotta alijärjestelmä SAR toimii. Kyseinen alijärjestelmä sallii SAR-arkistojen käyttöön ottamisen, ja se vaatii, että järjestelmästä löytyy JMX-alijärjestelmä. JMX- ja SAR-alijärjestelmät löytyvät JBoss EAP 7.1 -palvelimesta oletuksena, mutta JBoss EAP 6.4 -konfiguraatioiden siirtäminen JBoss EAP 7.1 -palvelimeen päällekirjoitti osan JBoss 7.1 -asetuksista ja poisti JMX-järjestelmän palvelimesta.

JMX-alijärjestelmän lisäys tapahtui muokkaamalla palvelimen standalone.xml-tiedostoa, joka on JBoss-palvelimen konfigurointitiedosto. Standalone.xml sisältää tiedot kaikista palvelimen alijärjestelmistä sekä sisälletyistä Java-moduuleista. Korjaus toteutettiin hakemalla alkuperäisen JBoss 7.1-palvelimen standalone.xml-tiedostosta JMX-alijärjestelmän asetus päivitetyn palvelimen standalone.xml-tiedostoon.

Kun alijärjestelmät olivat toiminnassa, saatiin palvelimelta viesti, että palvelin ei löydä moduuleja JSF 1.2 ja Persistence API 2.1. Moduulit ovat siis Java-kirjastokokoelmia, joita JBoss hyödyntää palvelimen kokoamisessa. Nämä eroavat sovelluksen sisäisistä kirjastoista siten, että ne ovat palvelimeen sidonnaisia, eivätkä itse sovellukseen. Tässä

nousi esille yksi ongelmista, jotka aiheutuvat palvelimen päivittämisestä silloin, kun se vielä sisältää teknologioita, joita se ei tue. Ongelmana oli se, että JBoss EAP 7.1 ei enää tue JSF 1.2:ta tai Persistence API 2.1:tä. Tästä syystä se ei sisältänyt näitä moduuleja automaattisesti vaan ne oli lisättävä manuaalisesti. Tosin ideaalissa tilanteessa sovelluksen toiminta ei ylipäätänsä olisi ollenkaan palvelinsidonnaista. Tämä ongelma ei ollut korjattavissa tämän päivityksen yhteydessä.

Ratkaisu moduulien puuttumiseen oli yksinkertainen. Palvelimeen oli lisättävä puuttuvat moduulit aikaisemman version kansioista. Etsittiin puuttuvat moduulit eli JSF 1.2 sekä Persistence API 2.1 vanhasta JBoss 6.4 -palvelimen kansioista ja kopioitiin ne vastaaviin kohtiin JBoss 7.1 -palvelimen kansioissa.

Vaikka ratkaisun olisi pitänyt korjata kaikki palvelimen moduuliongelmat, tuli palvelimelta virheviesti käynnistyksen yhteydessä. Virheviestin mukaan palvelin ei löytänyt itse projektin palvelimen päämoduulia eli Runner.ear-moduulia, joka oli itse websovellus ja sen kirjastot. Tämä oli vaikea ongelma diagnosoida tai korjata, sillä se voi johtua monesta syystä.

Päämoduulin ongelmaan yritettiin etsiä ratkaisuja Internetistä tuloksetta. Lopuksi se saatiin korjattua valitsemalla Eclipsen palvelimen kokoamisasetuksista vaihtoehto ”Deploy projects as compressed archives”. Kyseinen vaihtoehto tekee niin, että palvelin saa websovelluksen käyttöönsä yhtenä kompressoituna .jar-arkistona. Tämän jälkeen palvelin kykeni löytämään päämoduulin käynnistyksen yhteydessä.

Selvää syytä sille, miksi projektin kompressoidun muodon välitys palvelimelle korjasi ongelman, ei löytynyt. Ongelmasta teki vielä hämmentävämmän se, että JBoss 6.4 ei tätä vaihtoehtoa vaatinut vaan kykeni löytämään projektin ei kompressoidussa muodossa. Hypoteeseina tosin olivat ne, että JBoss EAP 7.1 käsittelee tiedostopolkuja eri tavoin kuin JBoss EAP 6.4 tai että vika on Eclipse-ohjelmointiympäristössä ja siinä, miten se kokoaa projektin ja välittää sen palvelimelle.

Kun päämoduulin ongelmat oli korjattu, tuli vielä päivittää JBossin jboss-deployment-structure.xml-tiedosto. Kyseisessä tiedostossa määritellään, mistä moduuleista palvelin on riippuvainen sekä mitä versiota nämä moduulit käyttävät. Tiedosto päivitettiin siten, että palvelin oli riippuvainen Persistence Apin versiosta 2.1 ja JSF:n versiosta 1.2, jotta

JBoss ei käyttäisi uudempaa versiota näistä, jotka eivät toimisi ICEfaces 1.8.2:n ja Hibernate 4:n kanssa.

Viimeisenä osana JBoss-päivitystä tuli vielä päivittää Eclipsen projektiin, minkä JBoss version kirjastot ovat projektille käytettävissä. Tämä vaihdettiin versiosta 6.4 versioon 7.1 JBoss-version mukaisesti, jonka jälkeen Eclipse löysi JBoss 7.1 -kirjastot eikä se enää nostanut virheitä puuttuvista kirjastoista.

Kun viimeiset konfiguraatiot oli tehty, JBoss EAP 7.1 -palvelin saatiin käynnistettyä virheittä ensimmäistä kertaa. Palvelimen käynnistyttyä käytiin vielä tarkistamassa selaimella, että se on käyttäjän saavutettavissa ja toimii oikein. Tämä ei tosin taannut täyttä varmuutta päivityksen onnistumisesta vaan palvelimelle ja web-sovellukselle oli tehtävä kattavaa testaamista, mikäli tasonnosto toteutettaisiin näin.

5 Valittu toteutus ja sen testaus

Tasonnoston toteutustapoihin tutustumisen ja alustavan selvittelyn tekemisen jälkeen esitettiin toteutustavat projektin asiakkaalle, joka päätti, miten tasonnoston kanssa edetään. Tässä luvussa käydään läpi, mitä valitun toteutustavan kanssa tehtiin ja miten tasonnosto loppujen lopuksi toteutettiin.

Edellisessä luvussa tutustuttiin kahteen tasonnoston toteutustapaan, jotka olivat ICEfacesin sekä JBossin päivitys tai vain JBoss-palvelimen päivitys. Kun nämä toteutustavat esitettiin asiakkaalle, päädyttiin siihen lopputulokseen, että päivitetään vain JBoss-palvelin. Syynä tähän oli se, että ICEfacesin sekä JBossin päivitys olisi ollut liian aikaa vievää toteuttaa sekä testata.

Tasonnoston toteutustavaksi valittiin vain JBossin päivitys. Tämä toteutustapa toteutettiin alustavasti aikaisemman tutkimuksen perusteella tehdyn ohjeen mukaisesti. Vaikka ohjeella saatiinkin web-sovellus käynnistymään uudella JBoss EAP 7.1 -palvelimella, kyseinen ohje ei ollut täysin kattava, jonka takia oli testattava, miten tasonnosto vaikutti web-sovelluksen toimintaan kokonaisuudessa.

Järjestelmän toimivuus oli tärkeä varmistaa, sillä mitä myöhemmin virheellinen toiminta huomataan, sitä kalliimmaksi se käy. Syynä tähän on se, että järjestelmän muokkaus vaatii paljon enemmän toimia sen ollessa jo tuotannossa kuin vasta testausvaiheessa sekä sen lisäksi loppukäyttäjien käyttökokemus ja tehokkuus kärsivät, jos järjestelmä toimii virheellisesti tuotannossa.

Tasonnosto on vielä testauksessa tämän insinöörityön kirjoituksen aikana, joten ei vielä täysin tiedetä kaikkia mahdollisia ongelmia, joita tasonnostosta nousee. Voidaan kuitenkin käydä läpi, mitä toimenpiteitä on tehty jo tasonnoston testauksen eteen sekä tutustua teoreettisella tasolla siihen, mitä tulee vielä tehdä järjestelmän kattavassa testaamisessa.

Alustavasti testausta tehtiin toteutustavan tutkimuksen yhteydessä. Tämä tehtiin pääasiassa vain tarkastelemalla järjestelmän tilaa JBoss-päivityksen yhteydessä ja huomioimalla tässä vaiheessa nousevat virheet sekä pintapuolisin katsomalla, että järjestelmä ylipäätänsä toimii. Tosin tämän kaltainen testaus ei ole riittävää, kun kyseessä on iso järjestelmä, jonka moneen osaan palvelimen päivitys vaikuttaa.

JBossin päivittämisen jälkeen pitää palvelin ja järjestelmän osat, joihin se vaikuttaa testata kauttaaltaan, jotta saadaan varmuus tasonnoston onnistumisesta. Tämä testaus on vielä kesken kirjoituksen aikana, joten ei voida täysin sanoa, mitä vaiheita testauksessa tulee olemaan. Tästä syystä käydään läpi teoreettisia vaiheita, joita järjestelmälle on ainakin tehtävä testauksen yhteydessä.

Päivittämisen jälkeinen testaus voidaan aloittaa suorittamalla järjestelmän automatisoidut testit eli regressiotestit. Regressiotestaamisessa suoritetaan järjestelmään tehdyt testit, joilla tarkastetaan, että aikaisemmin toimineet toiminnallisuudet eivät ole hajonneet uuden ominaisuuden tai tässä tapauksessa palvelimen myötä. Palvelimen päivitys voi mahdollisesti hajottaa esimerkiksi ICEfaces-komponentteja, sillä se ei virallisesti tue niitä. Tästä syystä on siis hyvä suorittaa regressiotestaus, joka melko kattavasti tarkistaa, että järjestelmän koodikanta toimii vielä kuten sen kuuluukin.

Regressiotestaamisen lisäksi voitaisiin suorittaa yksikkötason testejä automatisoidusti, mutta niistä tuskin on hyötyä. Tämä johtuu siitä, että ne testaavat pääasiassa vain

eristetyissä ympäristöissä, että ohjelmistokoodin logiikka on toimiva. Tämä ei siis huomioi kokonaiskuvaa, joten se ei kata palvelimen vaihtumista.

Automatisoitujen testien ajamisen jälkeen voidaan siirtyä manuaaliseen testaukseen, johon kuluu eniten aikaa ja resursseja. Tosin siitä saadut tulokset ovat tärkeämpiä kuin automaatiotestauksesta. Manuaalinen testaus aloitetaan suunnittelemalla testisuunnitelma, joka kattaa mahdollisesti hajonneet tai muuten muuttuneet tilanteet. Pääasiassa siis tarkastetaan, että palvelin näyttää ja päivittää kaikki sivut oikein sekä osaa kommunikoida palvelinpuolen ohjelmistokoodin sekä käyttäjäpuolen välillä. Tämän testauksen laajuutta on vaikea tietää ennen testisuunnitelman tekoa, joka on vielä tekeillä kirjoituksen aikana.

Kun muut testit on suoritettu, suoritetaan vielä hyväksymistestaus. Hyväksymistestauksessa järjestelmän asiakas suorittaa omat testinsä, joiden perusteella järjestelmän hyväksytään toimivan ja se voidaan siirtää tuotantoon.

Vaikka testaamista jatkettaisiin hyvinkin kauan, on jossain vaiheessa päätettävä, että järjestelmä on tarpeeksi vakaa ja toimiva. Testaamisvaiheen pituus riippuu aikamääreistä ja budjetista.

Testaamisen lopettaminen ei tarkoita, että järjestelmä olisi täysin valmis, vaan järjestelmän toimintaa valvotaan käyttäjien nostamien havaintojen perusteella. Havainnot tulkitaan ja selvitetään, onko kyseessä virheellinen toiminta vaiko käyttäjän virhe. Tämän jälkeen voidaan mahdollisesti nostaa virhetiketti kehittäjien huomioon, jotka selvittävät virheen. Ideaalissa tilanteessa näitä virheitä on vain muutama eivätkä ne ole vakavuudeltaan kriittisiä.

6 Järjestelmän tila tasonnoston jälkeen ja tulevaisuuden mahdollisuudet

Vaikka tasonnosto onkin vielä testauksen osalta kesken kirjoituksen aikana, voidaan käydä läpi, mitä tasonnostolla saavutettiin ja missä tilassa järjestelmä on tasonnoston jälkeen. Tässä luvussa käydään siis läpi järjestelmän tila tasonnoston jälkeen ja pohditaan, miten se vaikuttaa järjestelmän tulevaisuuteen. Järjestelmän tulevaisuudesta

tutkitaan, mitä haasteita tehty tasonnosto aiheuttaa sekä mitä hyötyjä tehdystä tasonnostosta saatiin.

Tasonnoston jälkeen järjestelmän tila on muuttunut siten, että sen JBoss EAP -palvelin ei enää ole vanhentuneessa versiossa 6.4 vaan uudessa 7.1-versiossa. Tämä oli tasonnoston ainoa päivitys, jonka projektin asiakas koki tarpeelliseksi tällä hetkellä. Tosin kyseinen tasonnosto ei jää viimeiseksi, sillä järjestelmässä on vielä muutamia osia, jotka ovat vanhentuneita sekä vaarassa jäädä legacy-teknologiaksi. Näiden päivitykseen palataan tulevaisuudessa asiakkaan toiveesta.

Tasonnostossa päädyttiin päivittämään vain JBoss-palvelin, mutta se oli tärkeä päivitys. Tällä päivityksellä taataan järjestelmän tulevaisuuden ylläpito ja kehitys. Uusi JBoss-versio on modernimpi ja sisältää tuen uudemmille järjestelmän osille sekä vanhojen järjestelmän osien uusille versioille. Palvelimen päivitys siis helpottaa muiden järjestelmän osien päivitystä. Tämän lisäksi uusi versio on vielä tuettu vuoteen 2023 asti, joka auttaa löytämään dokumentaatiota ongelmanratkaisutilanteissa. Myös uudet päivitykset voivat korjata palvelimesta löytyviä virheitä.

Päivittämisen lisäksi tasonnoston yhteydessä tutustuttiin ICEfaces-kirjaston päivitykseen, jota ei päädytty toteuttamaan. Tämän eteen tehty työ ei kuitenkaan ollut turhaa, sillä sitä voidaan käyttää tulevaisuudessa pohjana alustamaan ICEfaces-kirjaston tasonnostoa.

Vaikka tasonnosto toteutettiinkin onnistuneesti, jää järjestelmään vielä ongelmakohtia, joita se ei ratkaissut. Suurimpana ongelmana on vanhentunut ICEfaces-kirjasto, jota uusi JBoss-versio ei virallisesti tue. Tällä hetkellä palvelin vaikuttaisi toimivan vanhan kirjaston kanssa, mutta palvelimen testaus on vielä kesken, joten siitä ei ole täyttä varmuutta. Mikäli testauksessa havaitaan ongelmia uuden palvelimen ja kirjaston välillä, on joko päivitettävä kirjasto uuteen versioon tai keksittävä jokin ongelman kiertävä ratkaisu. Kumpikaan ratkaisu ei ole täydellinen, sillä kirjaston päivitys on aikaisemmin todettu hyvinkin aikaa ja resursseja vieväksi prosessiksi. Ongelman kiertävä ratkaisu puolestaan voi aiheuttaa ongelmia tulevaisuudessa, jos sitä ei dokumentoida hyvin eikä se ole hyvien ohjelmointikäytäntöjen mukaista.

Tasonnostosta nousseet haasteet eivät loppujen lopuksi ole niin suuria kuin siitä saatu hyöty. Vaikka tasonnoston eteen tehty työ voi vaikuttaa hieman turhalta, sillä se ei korjaa mitään tai tarjoa uusia ominaisuuksia tai toiminnallisuuksia, on se silti tärkeä tehdä ennen kuin myöhemmin. Tasonnoston avulla järjestelmä pysyy ajan tasalla jatkuvasti kehittyvällä teknologian alalla. Tämän lisäksi se tarjoaa hyvän pohjan järjestelmän tulevaisuuden ylläpidolle sekä mahdollisille tuleville tasonnostoille.

7 Yhteenveto

Tässä luvussa käydään läpi, mitä insinöörityöllä tavoiteltiin ja miten onnistuneesti näihin tavoitteisiin päästiin. Tämän lisäksi tutkitaan työn etenemistä yleisellä tasolla eli käydään läpi työssä tehtyä prosessia sekä mitä olisi voitu tehdä toisin tai paremmin. Lopuksi pohditaan introspektiivisesti, mitä työstä opittiin.

Insinöörityön tavoitteena oli tutustua Java web -sovelluksen tasonnostoon ja sen toteuttamiseen. Tämä jaettiin pienempiin vaiheisiin, joista tasonnostoprosessi muodostuu. Ensimmäisenä vaiheena oli web-sovelluksen järjestelmään tutustuminen ja siitä tasonnoston tarpeessa olevien järjestelmäosien löytäminen. Seuraavassa vaiheessa pohdittiin tasonnostoprosessia sekä siitä saatavia hyötyjä. Kyseisessä vaiheessa käytiin läpi myös syitä, miksi tasonnostoa usein vältetään niin pitkään kuin mahdollista, vaikka se olisi hyvä toteuttaa mahdollisimman ajoissa.

Alustavien pohdintavaiheiden jälkeen tutkittiin tasonnoston toteutustapoja. Toteutustavoista selvitettiin asiakkaantarpeisiin toimivin ratkaisu, jota lähdettiin toteuttamaan. Toteutusvaiheen jälkeen tutkittiin vielä tehdyn tasonnoston testausta sekä vaikutuksia. Viimeisenä tavoitevaiheena oli selvittää, mihin tilaan järjestelmä päätyi tasonnoston jälkeen ja mitä mahdollisuuksia tämä tarjoaa tulevaisuudessa.

Ensimmäinen vaihe eli järjestelmän ymmärtäminen sekä tasonnoston tarpeen tunnistaminen osakomponenteista sujui onnistuneesti, vaikka projekti olikin iso. Web-sovelluksen järjestelmästä saatiin kokonaiskuva muodostettua ja siitä löydettiin kaksi osaa, joille tasonnosto voitaisiin toteuttaa. Nämä osat olivat sovelluksen palvelin ja ICEfaces-kirjasto.

Kun tasonnoston tarve oli havaittu, siirryttiin seuraavaan vaiheeseen eli tasonnostoprosessin ymmärtämiseen. Tässä vaiheessa selvitettiin, mitä tasonnostoprosessissa tulisi tehdä ja tunnistettiin tärkeitä syitä, miksi tasonnosto tulisi toteuttaa. Vaihe suoritettiin ilman ongelmia.

Teoreettisten vaiheiden jälkeen tutkittiin tasonnoston toteutustapoja. Toteutustavoista pyrittiin selvittämään, miten tasonnosto voidaan käytännössä toteuttaa projektin sovellukselle sekä mikä näistä toteutustavoista sopii parhaiten asiakkaalle. Tämä toteutettiin suurimmalta osalta ongelmitta ja löydettiin asiakkaan tarpeisiin toimiva ratkaisu. Ratkaisuksi valittiin toteutustavoista pelkkä JBoss-palvelimen päivitys. Tosin tässä olisi voitu tutustua hieman tarkemmin ICEfaces-kirjaston päivittämiseen, mutta asiakas totesi, että se on liian raskas toteuttaa.

Tasonnoston toteuttaminen oli työn seuraava vaihe. Tämä toteutettiin aikaisemmin selvitetyn ohjeen mukaan pelkällä JBoss-palvelimen päivityksellä. Tasonnosto saatiin toteutettua melko yksinkertaisesti luodun ohjeen perusteella. Tästä vaiheesta jäi pieneksi pettymykseksi tasonnoston testaus, joka on vielä tekeillä työn kirjoituksen aikana. Testaukseen olisi voitu perehtyä paremmin ja selvittää tarkemmin, mihin järjestelmän osiin palvelimen päivitys vaikuttaa.

Työn lopussa pohdittiin vielä järjestelmän tilaa tasonnoston jälkeen sekä miten vaikuttaa järjestelmän tulevaisuuteen. Tämä ei vaatinut muuta kuin pohdintaa, joten se onnistui helposti. Tosin järjestelmän tulevaisuudesta voidaan tehdä vain arvioita, sillä tasonnoston testaus on vielä vaiheessa. Testauksen vähäisyys vaikuttaa siten myös tähän.

Insinöörityön tavoitteen kaikki vaiheet suoritettiin onnistuneesti. Aluksi selvitettiin tasonnostoprosessin kulku ja taustatiedot. Näiden perusteella suoritettiin Java-websovellukselle onnistunut tasonnosto. Tavoitteisiin päästiin onnistuneesti.

Työn tekeminen opetti tasonnostoprosessin vaiheet ja mitä missäkin vaiheessa tehdään. Konkreettisen tiedon lisäksi jäi insinöörityöstä parempi käsitys siitä, miten työelämän ympäristöissä ohjelmistotuotantoprosessi ei ole niin yksinkertaista ja sääntöjen mukaista kuin koulussa. Usein on etsittävä ongelmiin ratkaisuja, jotka eivät ole tehokkaimpia vaan yksinkertaisempia. Tämä johtuu siitä, että tehokas ratkaisu ei aina ole nopein toteuttaa ja loppujen lopuksi asiakas haluaa toimivan ohjelmiston nopeasti. Tehokkuuteen voidaan

etsiä ratkaisuja myöhemmin, mikäli siitä nousee ongelmia. Tosin ei pidä aina mennä sieltä, mistä aita on matalin, vaan insinöörin on tutkittava monia ratkaisuja ja valittava toimivin ratkaisu ohjelmiston ja asiakkaan perusteella.

Lähteet

1 JSF - Managed Beans. 2020. Verkkoaineisto. Tutorials Point. <https://www.tutorialspoint.com/jsf/jsf_managed_beans.htm>. Luettu 16.3.2020.

2 Rouse, Margaret. 2017. Java Platform, Enterprise Edition (Java EE). <https://www.theserverside.com/definition/J2EE-Java-2-Platform-Enterprise-Edition>. Luettu 16.1.2020.

3 Differences between Java EE and Java SE. 2012. Verkkoaineisto. Oracle Corporation. <https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html>. Luettu 15.1.2020.

4 Hibernate ORM. Verkkoaineisto. <https://hibernate.org/orm/>. Luettu 27.1.2020

5 HTML and XHTML. Verkkoaineisto. Refsnes Data. <https://www.w3schools.com/html/html_xhtml.asp.> Luettu 24.1.2020.

6 ICEfaces Overview. Verkkoaineisto. Ice Soft Technologies Inc. <http://www.icesoft.org/java/projects/ICEfaces/overview.jsf>. Luettu 24.1.2020.

7 Niemistö, Tero. 2016. Tekninen velka - yrityksesi suurin digitalisaation jarru? <https://blog.kauppalehti.fi/digiarjessa/tekninen-velka-yrityksesi-suurin-digitalisaation-jarru>. Luettu 13.2.2020.

8 JBoss Enterprise Application Platform 6 (EAP 6) End of Maintenance FAQ. 2020. Verkkoaineisto. Red Hat, Inc. <https://access.redhat.com/articles/3751821>. Luettu 19.2.2020.

9 Ice Soft eNewsLetter. 2015. Verkkoaineisto. Ice Soft Technologies Inc. <https://us10.campaign-archive.com/?u=c5a015915e9b8e8485228b74f&id=125fb4b888>. Luettu 19.2.2020.

10 ICEfaces Supported Platforms. 2020. Verkkoaineisto. Ice Soft Technologies Inc. <http://www.icesoft.org/java/projects/ICEfaces/supported-platforms.jsf>. Luettu 19.2.2020.

11 Converting ICEfaces 1.8 Applications to ICEfaces 3 Tutorial. 2017. Verkkoaineisto. Ice Soft Technologies Inc. <https://www.icesoft.org/wiki/display/ICE/Converting+ICEfaces+1.8+Applications+to+ICEfaces+3>. Luettu 20.2.2020.

12 ICEfaces 1.x Compatibility Features of ICEfaces 3. 2017. Verkkoaineisto. Ice Soft Technologies Inc. <https://www.icesoft.org/wiki/display/ICE/ICEfaces+1.x+Compatibility+Features+of+ICEfaces+3>. Luettu 2.3.2020.

13 Class RenderManager. 2009. Verkkoaineisto. Ice Soft Technologies Inc. <http://res.icesoft.org/docs/v1_8_2/javadocs/icefaces/api/com/ice-soft/faces/async/render/RenderManager.html>. Luettu 2.3.2020.

14 Class ViewHandler. Verkkoaineisto. Oracle Corporation. <https://docs.ora-cle.com/cd/E17802_01/j2ee/javaee/javaserverfaces/2.0/docs/api/javax/faces/ap-plication/ViewHandler.html>. Luettu 2.3.2020.

15 Ajax Push - APIs. 2014. Verkkoaineisto. Ice Soft Technologies Inc. <https://www.icesoft.org/wiki/display/ICE/Ajax+Push+-+APIs>. Luettu 2.3.2020.

16 JBoss Server Migration Tool User Guide. 2017. Verkkoaineisto. Red Hat, Inc. <https://docs.jboss.org/author/display/CMTOOL/JBoss+Server+Migra-tion+Tool+User+Guide>. Luettu 4.3.2020.

17 Migrating from Previous Versions of JBoss EAP to JBoss EAP 7.x. 2017. Verkko-aineisto. Red Hat, Inc. <https://docs.jboss.org/author/disp-lay/CMTOOL/JBoss+EAP+6.4+to+7.0#JBossEAP6.4to7.0-MigratingfromPre-viousVersionsofJBossEAPtoJBossEAP7.x>. Luettu 4.3.2020.

18 JMX subsystem configuration. 2015. Verkkoaineisto. Red Hat, Inc. <https://docs.jboss.org/author/display/AS71/JMX+subsystem+configuration>. Lu-ettu 9.3.2020.