jamk.fi

Vaihtoehtoisten teknologioiden vertailu käyttöliittymän kehittämisessä Liferay-ympäristössä

Jani Oksanen

Opinnäytetyö
Syyskuu 2018
Luonnontieteiden ala
Tradenomi (AMK), Tietojenkäsittelyn tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Oksanen, Jani	Opinnäytetyö, AMK	Syyskuu 2018
	Sivumäärä 40	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Vaihtoehtoisten teknologioiden vertailu käyttöliittymän kehittämiseen Liferay-ympäristössä

Tutkinto-ohjelma

Tietojenkäsittelyn tutkinto-ohjelma

Työn ohjaaja(t)

Tommi Tuikka

Toimeksiantaja(t)

Kansaneläkelaitos ICT-palvelujen tulosyksikkö

Tiivistelmä

Kelan ICT-palveluissa on kehitetty useita sähköisiä palveluita, joiden pääsääntöisenä ohjelmistokehyksenä on toiminut JavaServer Faces. Nämä palvelut on pakattu .war-paketeiksi. Uusimman Liferay 7-version myötä tämä paketointitapa on merkattu vanhentuneeksi, mikä tarkoittaa suoran tuen loppumista Liferayn puolesta. Tästä syystä on suotavaa vaihtaa sovellusten paketointi .jar-paketointiin eli toisin sanoen OSGi-moduleihin.

Tutkimuksen päätarkoituksena on vertailla React- ja Angular-ohjelmistokehyksiä JavaServer Faces teknologiaan käyttöliittymän kehittämisessä, sekä soveltuvuuteen Kelan tarpeisiin. Tutkimuksessa pyritään myös selvittämään sopivaa korvaajaa JavaServer Facesille.

Tutkimus toteutettiin kvalitatiivisena tutkimuksena, sillä tarkoituksena oli vertailla ohjelmistokehyksiä käyttöliittymän kehittämiseen Liferay- ja muissa web-sovelluspalvelinympäristöissä.

Vertailukohtana tutkimukselle toimii vanha JSF-sovellus, sekä tutkimuksen vertailukohdat määritettiin haastattelemalla Kelan kehittäjiä, joilla on usean vuoden kokemus Liferaysta, sekä muista ohjelmistokehyksistä. Kehysten vertailu toteutettiin pisteytyksellä nollasta viiteen ja tulos kirjattiin yhteenvedossa raportin muotoon.

Tutkimuksen tuloksena luotiin kaksi sovellusta käyttäen Angular - ja React-ohjelmistokehyksiä, sekä Spring boot Restful API-palvelu tietokantoja varten. Kehysten havaittiin olevan suhteellisen vakaita ja ne saavat päivityksiä tasaisella aikataululla. Kehysten soveltuvuutta Kelan tarpeisiin ei kuitenkaan voida suositella, sillä rakenteellisia vaatimuksia ei pystytty luomaan Liferayn NPM bundlerin ongelmien vuoksi.

Avainsanat (asiasanat)

Liferay, React, Angular, JSF, ohjelmistokehys, Javascript, Typescript, OSGi moduuli, Spring boot restful API

Muut tiedot (salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Oksanen, Jani	Bachelor's thesis	September 2018
		Language of publication: Finnish
	Number of pages 40	Permission for web publication: x

Title of publication

Comparison of alternative frameworks for user interface development in Liferay environment

Degree programme

Business Information Systems

Supervisor(s)

Tuikka, Tommi

Assigned by

The Social Insurance Institution of Finland

Abstract

Kela(The Social Insurance Institution of Finland in English) has developed multiple electronic services over years. The main framework for those services has been JavaServer Faces(JSF) packed to .war extension. With the newest Liferay 7 version, this package type has been marked as deprecated, which means the end of direct support from the Liferay Inc. Because of this, Liferay does not recommend the use of this kind of packaging type and the software packages should be converted into .jar package type otherwise called as OSGi modules.

The main goal of this research was to compare React and Angular frameworks for user interface development keeping in mind Kela's requirements. The conclusion for comparison is to recommend new framework to replace JavaServer faces.

The research was implemented as qualitative research because the main goal was to compare and find a suitable framework for development in Liferay portal and other web application servers. The baseline for comparison was defined by interviewing Kela's developers who had been working many years with Liferay and other frameworks as well as older JSF portlet developed by Kela.

The framework comparison was implemented with points from 0 to maximum of 5 and the results were summarized in the conclusion part.

Two portlets were created during the research, one with React and one with Angular and one Spring boot restful API for database connection management. Because of problems with Liferay npm bundler, based on the research, the use of these frameworks for Kela cannot be recommended.

Keywords/tags (subjects)

Liferay, React, Angular, JSF, framework, Javascript, Typescript, OSGi module, Spring boot Restful API

Miscellaneous (Confidential information)

Sisältö

Kuviot

Käsitteet

Kela	Kansaneläkelaitos tai Kansaneläkelaitoksen ICT-palveluiden tulosyksikkö. Kehittää sähköisiä palveluita sisäiseen ja ulkoiseen käyttöön.
Oracle	Oracle-Corporation. Javan kehittäjä ja ylläpitäjä sekä JSF-ohjelmistokehyksen ylläpitäjä/kehittäjä.
Ohjelmistokehys	Rakenne, jolla sovelluksen käyttöliittymä luodaan. Suomennos englannin kielen sanasta Framework.
Ecmascript 6	Javascript standardi, joka määrittää pohjan Javascript ohjelmoinnille. Ecmascript 6 selainten tuki on vaiheessa ja vaatii kompiloinnin, eli kääntämisen.
Java	Oraclen kehittämä ohjelmointikieli, jolla kehitetään sovellutuksia.
JavaEE	Oraclen kehittämä ohjelmointikieli, joka perustuu Javaan ja lisää siihen web-tekniikoita.
Javascript	Ohjelmointikieli, jolla kehitetään web-sovellutuksia.
Typescript	Microsoftin kehittämä ohjelmointikieli, jonka syntaksi muistuttaa Javascriptiä.
State	Sovelluksen sisäinen muisti, johon tallennetaan tietoja esimerkiksi formin välityksellä. State tyhjennetään session päätyttyä.
Javascript moduuli	Javascript moduuli tiivistää koodin käytännölliseksi osaksi, joka vie ulos omia arvojaan.
AMD loader	Sanoista Asynchronous Module Definition. Loader nimensä mukaisesti lataa sivustolla vain tarvittavat modulit ja niiden sisältämät koodit.
JSF (JavaServer Faces)	Oraclen ohjelmistokehys käyttöliittymän kehittämiseen JavaEE-sovellutuksille.

React	Facebookin kehittämä ja ylläpitämä Javascript-pohjainen ohjelmistokehys.
Angular	Googlen kehittämä ja ylläpitämä Typescript-pohjainen ohjelmistokehys.
Liferay	Ohjelmistoportaali, jolle voidaan kehittää internet ja intranet sivustoja.
PrimeFaces	JSF-kirjasto käyttöliittymän kehittämiseen JavaEE-sovellutuksille.
BackEnd	Sovellutuksen taustakerros, joka keskustelee tietokantatojen ja datankäsittelijöiden kanssa.
FrontEnd	Sovellutuksen päällyskerros, joka luo käyttäjälle käytettävän käyttöliittymän sekä lähettää ja pyytää backendiin tietoja.
IDE	Ohjelmisto, jolla kehitetään sovellutuksia. Yleisimpiä esimerkkejä Eclipse, Netbeans ja Intellij. Tulee sanoista Integrated Development Environment.
DOM	Englannin kielen sanoista Document Object Model. Selaimen luoma malli HTML rakenteesta, jota käytetään Javascript ohjelmoinnissa hyödyksi.
Tietotaulu	Englannin kielen sanasta datatable. HTML taulurakenne, jossa on sarakkeita ja rivejä.
Kalenteri	Kalenteri, jota käytetään sovelluksissa esimerkiksi jonkin alkupäivämäärän esittämiseen. Sisältää datepicker toiminnon, eli mahdollisuuden valita päivämäärän.
Haitarirakenne	Tietorakenne, jossa on elementtejä elementin sisässä, ja niitä voidaan availla tiettyä elementtiä painamalla. Englanninkielinen termi accordion.

Portlet	Web-pohjainen komponentti, joka mahdollistaa integraation sovelluksen ja portaalin välillä. Liferay-ympäristössä sivustoille asetetaan portlet komponentteja, jolla sivuston sisältö luodaan.
Object Relational Mapping	Toiselta nimeltään ORM. Ohjelmointitekniikka, jossa olio-ohjelmoitua koodia yhdistetään relaatiotietokantaan. Tietokannasta löytyvästä datasta luodaan entiteetti, joka asetetaan Java olioon.
Rest	Lyhenne sanoista Representional State Transfer. Arkkitehtuurinen tyyli jaetulle hypermedialle.

1 Johdanto

1.1 Opinnäytetyön tavoite

Opinnäytetyön tavoitteena on vertailla uusia ohjelmistokehyksiä käyttöliittymän kehittämiseen, sekä selvittää mahdollisia korvaajia JSF-teknologialle käyttöliittymän kehityksessä, ja ehdottaa niitä toimeksiantajalle. Tutkimukseen asetetut tutkimuskysymykset saatiin toimeksiantajan puolesta, sekä ohjelmistokehyksille määritettiin samalla rakenteelliset minimivaatimukset.

Tutkimuksen vertailukohdaksi saatiin JavaServer Faces-ohjelmistokehyksellä luotu palautelomake, josta tutkimuksen aikana luodaan uusittu versio käyttäen nykyaikaisia ohjelmistokehyksiä Reactia ja Angularia. Angular- ja React-sovellusten lisäksi tutkimusta varten tehdään Spring boot REST API-palvelu, jonka tehtävänä on hoitaa tietokannan käsittely.

1.2 Toimeksiantajan esittely

Opinnäytetyön aihe saatiin toimeksiantona Kansaneläkelaitoksen ICT-palveluiden tulosyksiköstä. Kelan ICT-palvelut on perustettu vuonna 1968 nimellä Kelan tietotekniikkaosasto, ja pääpaikka sijaitsi Helsingissä. Kelan ICT-palvelut ovat Kansaneläkelaitoksen alainen organisaatio, jonka vastuulla on kehittää sähköisiä palveluita sisäiseen, sekä ulkoiseen käyttöön. Kelan ICT-palvelut toimivat pääsääntöisesti Helsingissä, Turussa, sekä Jyväskylässä, ja henkilöstöä on yhteensä noin 600.

Kuvio 1. Kansaneläkelaitoksen organisaatiokaavio.

1.3 Opinnäytetyön tietoperusta sekä rakenne

Tämän opinnäytetyön tietoperustana käytetään pääsääntöisesti sovelluspalveluiden tuottajien omia dokumentaatioita, sähköisiä artikkeleita, sekä vertailukohtien määrittämiseen haastatteluita toimeksiantajan puolelta.

Opinnäytetyön johdannossa kuvataan opinnäytetyön tavoitteet, kuvataan toimeksiantaja, sekä kerrotaan tietoperusta ja rakenne. Tutkimusasetelmassa kuvataan tutkimuksen lähtökohta, sekä rajaus, kerrotaan tutkimusmenetelmä, sekä tutkimuskysymykset.

Tutkimuksen viitekehystä, eli käytännön osiota ennen, käydään läpi tutkimuksessa käytettävät teknologiat yleisellä tasollaan. Käytännönosassa vertaillaan ohjelmistokehystä vertailukohtien valossa, sekä selvitetään sopivaa kehystä toimeksiantajalle. Opinnäytetyön lopussa kootaan käytännön tutkimuksessa saadut tulokset raporttimuotoon yhteenvedossa. Pohdinnassa käydään läpi tutkimuksen aikana ilmenneet ongelmat, sekä mahdolliset jatkotutkimusaiheet.

2 Tutkimusasetelma

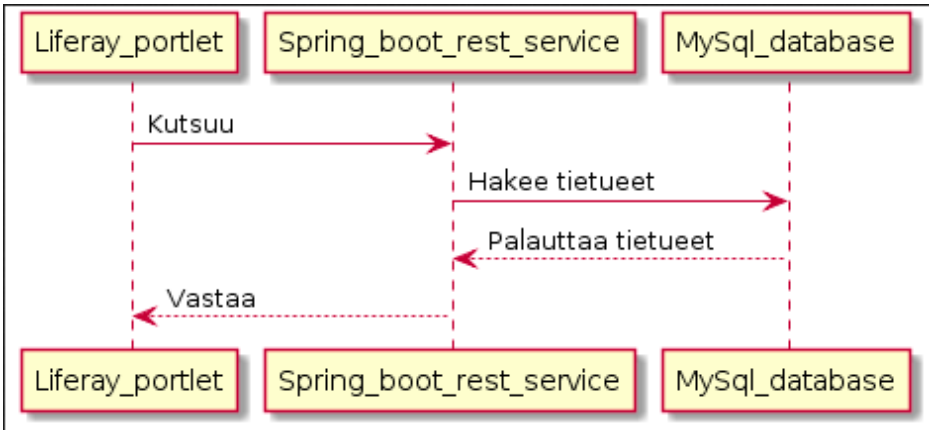
2.1 Tutkimuksen lähtökohta ja rajaus

Kelassa on tuotettu web-palveluita jo usean vuoden ajan, ja suurin osa niistä on kehitetty käyttäen JavaEE-ohjelmointikieltä, sekä JavaServer Faces-ohjelmistokehystä.

Tutkimuksen aikana kehitetään kaksi erillistä sovellusta, joiden ohjelmistokehyksinä käytetään Reactia ja Angularia, sekä RESTFUL-palvelu, jolla sovellusten dataa päivitettiin tietokantaan. Tutkimuksesta saatavia tuloksia verrataan toimeksiantajalta saatuun JavaEE-sovellukseen, jonka käyttöliittymä on kehitetty käyttäen JSF-ohjelmistokehystä, sekä Primefaces-komponentteja. Vertailukohdaksi saadun sovelluksen opinnäytetyön tekijä on aiemmin syksyllä 2017 kääntänyt vanhasta Liferay 6.2-ympäristöstä Liferay DXP-ympäristöön.

Opinnäytetyö rajataan Liferay-ympäristöön, jonka kanssa työn tekijä on työskennellyt syksyn 2017 ja kevään 2018 projektissa, jossa kehitetään uutta sivustoa käyttäen Liferayn DXP-versiota. Tutkimuksessa luodaan palautelomakesovellus käyttäen React- ja

Angular 2-ohjelmistokehyksiä, ja niissä käsiteltävät tiedot tallennetaan mysql-tietokantaan käyttäen itse tehtyä Spring boot restful-palvelua.



Kuvio 2. Luotavien sovellusten elinkaavio.

Ennen tätä tutkimusta Kelan ICT-palveluissa ei ole kehitetty Liferay-portletteja, joiden käyttöliittymän tekemiseen olisi käytetty Angularia tai Reactia.

2.2 Tutkimusmenetelmä

Tutkimuksen tavoitteena on vertailla Angular- ja React-ohjelmistokehyksiä käyttöliittymän kehittämiseen Liferay-ympäristössä vanhentuvan teknologian tilalle. Tutkimusta lähdetään tekemään laadullisena(kvalitatiivisena) tutkimuksena, koska vertailukohta tulee toimeksiantajan puolesta, ja johon valittuja ohjelmistokehyksiä verrataan. Käytännössä tutkimuksen vaatimukset saadaan toimeksiantajalta, mitä ominaisuuksia ohjelmistokehykseltä pitää löytyä, kuin myös tutkimuskysymysten määräämät tarpeet.

Aineistonkeruumenetelminä käytetään sähköisiä lähteitä: teknisiä dokumentaatioita ja artikkeleita. Tutkimuksesta saatavia tuloksia verrataan keskenään pisteytyksellä, mutta lopullinen tulos kirjataan raportin muotoon. Vertailukohtina käytetään esimerkiksi vaadittujen ominaisuuksien toteutuksen helppoutta, ja näistä kuvaavan dokumentaation löytymistä, tai esimerkiksi ylläpidettävyyttä.

2.3 Tutkimuskysymykset

Tutkimuskysymyksiä asetettiin kaksi kappaletta, jotka toimivat niin sanottuina pääkysymyksinä.

Kuinka vaivaton käyttöliittymän tekeminen valitulla teknologialla on? Tämä kysymys jaetaan useampaan alikysymykseen; kuinka vaivaton ohjelmistokehys on ottaa käyttöön? Pystyykö ohjelmistokehys vastaamaan toimeksiantajan vaatimuksiin? Kehittämisen nopeus valitulla ohjelmistokehyksellä, sekä ohjelmistokehyksestä löytyvä dokumentaatio.

Kuinka ylläpidettävä valittu ohjelmistokehys on? Tämä jaetaan myös useampaan alikysymykseen; kuinka pysyvä ohjelmistokehys on ja millainen historia siltä löytyy? Kuinka ohjelmistokehys saa versiopäivityksiä, ja kuinka suuret muutokset niissä on?

3 Tutkimuksessa käytettävät teknologiat

Tässä luvussa kuvataan lyhyesti tutkimuksessa käytettävät teknologiat.

3.1 Apache Maven ja Blade CLI

Sovelluskehittämisen yhteydessä sovellukset täytyy rakentaa, jotta ne saadaan asennettua sovelluspalvelimelle. Rakentamisella viitataan tapaan, jolla projekti konvertoidaan itsenäiseen muotoon, jotta sitä voidaan ajaa joko sovelluspalvelimella tai itsenäisesti tietokoneella. Yksi rakentamisen päätehtävistä on kääntää kirjoitettu koodi sellaiseen muotoon, että tietokone osaa käyttää/ajaa sitä. (Definition - what does build mean? N.d). Yleisimpiä sovelluksen rakentamiseen käytettäviä työkaluja ovat Apache Maven, sekä Gradle.

Maven on sovelluksen rakentamisenhallintaan tarkoitettu työväline, ja sen tehtävänä on rakentaa ja hallita Java-pohjaisia projekteja, sekä tarjota niille riippuvuuksien hallinnan. Maven pohjaisesta projektista löytyy pom.xml tiedosto, jossa määritetään sovelluksen versio ja käytettävät riippuvuudet. Riippuvuuksilla tarkoitetaan Java-kirjastoa, johon on paketoitu rajapintoja, luokkia ja metodeita uudelleen käytettäväksi. Riippuvuuksille voidaan määrittää scope, jolla rajoitetaan riippuvuuden transitiivisuutta. Oletusarvona riippuvuuksilla on compile. Tällöin riippuvuus on saatavilla projektin kaikissa classpatheissa(määrite, mistä Java virtual machine etsii tarvittavia luokkia). Provided scope on kuin compile, mutta tällä oletaan, että Java Development Kit tai kontti tarjoaa kyseisen riippuvuuden sovelluksen ajon ajaksi. Runtime scope tarkoittaa, että riippuvuus ei ole tarpeellinen sovelluksen kompiloinnin aikana vaan

sitä tarvitaan sovellusta käytettäessä. Test määrittää riippuvuuden tarpeelliseksi ainoastaan testitapauksiin. System scope on samanlainen kuin provided scope, mutta tässä tapauksessa tarvittava paketti täytyy itse lisätä projektiin (Introduction to the Dependency Management N.d). Halutut riippuvuudet ladataan nexus repositorioista, joita voidaan määrittää sovelluskehittimen Maven asetuksiin tai vastaavasti komentolinjan Maven konfiguraatiotiedostoihin.

Maven työkalu toimii niin unix kuin Windows-pohjaisilla käyttöjärjestelmillä ja vaatii toimiakseen Javan. Mavenin asentaminen on yksinkertaista, sillä ainoat vaatimukset ovat sopivan versiopaketin lataaminen apachen sivuilta, sen purkaminen omalle koneelle, ja siitä saatavan bin kansion lisääminen käytäjän PATH muuttujaan (Installing Apache Maven N.d). Maven työkalua käytetään joko PATH muuttujaan lisätyn mvn komennon avulla komentoriviltä, tai sovelluskehittimessä integroituna.

Blade CLI on Liferayn tuottama työkalu, jolla voidaan helposti rakentaa uusia Liferay moduleja. Blade CLI on ilmainen Gradle-pohjainen komentolinjalta ajettava rakennustyökalu ja sitä käytetään Liferay 7.0 modulien rakentamiseen (Blade CLI N.d). Työkalulla voidaan luoda niin Gradle, kuin Maven pohjaisia projekteja

Bladen asentaminen käy helposti lataamalla Liferay sivulta asennuspaketin, joka automaattisesti asentaa ohjelmiston (Installing Blade CLI N.d). Tässä opinnäytetyössä React ja Angular projektit on luotu käyttäen Blade CLI työkalua.

3.2 Liferay ja Apache tomcat

Liferay portaali on Liferay Inc yhtiön kehittämä ja ylläpitämä sisällönhallintajärjestelmä, jolla voidaan rakentaa internet ja intranet sivuja. Portaalissa sovellukset ovat useimmiten portletteja, joita voidaan lisätä sivulle drag-and-drop tyyppisestä tai upotettuna custom teeman kautta.

Liferay portaalin lähdekoodi on kirjoitettu Javalla, ja siihen pystytään lisäämään omia ns. OSGi fragmentteja, joilla lähdekoodin toteutusta pystytään muokkaamaan haluttuun suuntaan. Vaikka portaali on kirjoitettu Javalla, voidaan sille kuitenkin kehittää web applikaatioita myös esimerkiksi Reactilla tai Angularilla (Introduction to Liferay Development 2017). Sovelluspalvelimena Liferay käyttää joko Apachen Tomcattia tai

Redhatin Wildflyta. Tässä tutkimuksessa sovelluspalvelimena käytettiin Apache tomcattia.

Apache Tomcat on Apache Software Foundationin kehittämä ja ylläpitämä avoimenlähdekoodin sovelluspalvelin, joka implementoi Java Servlet-, JavaServer Pages-, Java Expression language- ja Java webSocket-teknologiat (Apache Tomcat N.d).

Liferay portaalista on saatavilla kaksi versiota, joista ensimmäinen on DXP, joka on maksullinen ja suunnattu yrityskäyttöön. DXP-versio tarjoaa Liferayn puolesta kaupallisen tuen tuotteelleen. Toinen versio tunnetaan nimellä CE, tämä on ilmainen, kaikkien ladattavissa oleva portaaliratkaisu, joka tarjoaa karsitun version maksullisesta DXP versiosta. Tässä tutkimuksessa käytetään Liferayn CE versiota.

Liferay CE:n asentaminen on helppoa, sillä se vaatii ainoastaan Liferayn sivulta ladattavan Liferay-Tomcat bundlen. Tämä bundle puretaan haluttuun kansioon omalle koneelle, jolloin portaali on käyttövalmis. Asentamisen jälkeen portaali käynnistetään komentolinjalta ja ensimmäisen käynnistyskerran yhteydessä määritetään käytettävä tietokantaratkaisu. Liferayn mukana tulee kehittämiseen tarkoitettu tietokanta Hypersonic, jota käytämme tässä tutkimuksessa. Asetusten tekemisen jälkeen portaali käynnistetään uudelleen, jolloin Liferay on valmis kehitystä varten (Installing Liferay portal on Tomcat 8 N.d).

3.3 Restful API

Sovelluksissa käytettävää dataa säilytetään tietokannoissa, ja niiden käsittelemiseen tarvitaan joko sovellukseen sisäänrakennettu tietokantayhteys, tai erillinen palvelu, joka hoitaa tarvittavat tapahtumat sovelluksen ja tietokannan välillä. REST palvelu on rajapinta, joka toimii järjestelmien välillä käyttäen http-protokollaa datan hakemiseen (BBVAOPEN4U 2016). Tärkeimpiä transaktioita datan käsittelemiseen ovat GET, jolla haetaan ja luetaan dataa, POST, jonka tehtävänä on luoda uutta tietoa tietokantaan, PUT, jota käytetään datan muokkaamiseen, PATCH, jota käytetään silloin, kun halutaan muokata osittain tiettyä entiteettiä, sekä DELETE, joka nimensä mukaisesti poistaa tietueen (HTTP Methods N.d).

HTTP - metodi	CRUD	Onnistunut kutsu	Epäonnistunut kutsu	Käyttöesimerkki
POST	Create	201	204(No content)	HTTP POST www.domain.com/users
GET	Read	200	404(Not Found) /400 (Bad Request)	HTTP GET www.domain.com/users?12
PUT	Update/Replace	201(Created), 200(Modified), 204(No Content)	404(Not Found)	HTTP PUT www.domain.com/users/12
PATCH	Partial Update/Modify	200(OK), 204(No Content)	404(Not Found)	HTPP PATCH www.domain.com/users/12
DELETE	Delete	200(OK), 202(Accepted),204(No Content)	404(Not Found)	HTTP DELETE www.domain.com/users/12

Kuvio 3. HTTP-kutsut ja niiden palauttamat koodit.

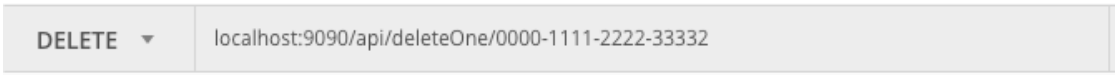
RESTillä on kuusi omaa arkkitehtuurista rajausta, joiden pitää täyttyä, mikäli rajapintaa halutaan viitata restful tyyppiseksi. Näitä rajauksia ovat yhtenäinen rajapinta, Asiakas-Palvelin, tilattomuus, välimuistiin laitettava, kerrostettu järjestelmä, sekä pyynnöstä ajettava koodi.

Yhtenäisessä rajapinnassa palvelulle täytyy määritellä URI, jota pitkin rajapinta avataan käyttäjille. Asiakas-Palvelin rajauksella tarkoitetaan mallia, jossa käyttäjälle näkyvän sovelluksen ja palvelin sovellusten on pystyttävä kehittymään omaan tahtiin ilman riippuvuutta toisiinsa. Tilattomuudella tarkoitetaan sitä, että jokaisen http kutsun pitää sisältää kaikki tarvittava tieto, mitä kutsuun vaaditaan, jolloin se ei voi käyttää palvelimen contextiin tallennettuja tietoja.

Välimuistiin tallennettavalla tiedolla tarkoitetaan mahdollisuutta tallentaa asioita selaimen tai sovelluksen välimuistiin. Tämä kasvattaa sovelluksen suorituskykyä ja skaalautuvuutta. Kerrostettu järjestelmä mahdollistaa RESTFUL apin julkaisemisen toisella palvelimella, tiedon pitämisen toisella, sekä kutsun autentikoinnin toisella.

Pyynnöstä ajettava koodi helpottaa ja keventää sovellusta, jolloin kaikkea ei tarvitse ladata ennen kuin niitä tarvitaan (RESTFUL API N.d).

Olioiden manipulointi tapahtuu urlin välityksellä, johon esimerkkinä voidaan antaa halutun entiteetin pääavain, jollloin REST API poistaa tietokannasta halutun rivin.



Kuvio 4. REST-kutsu, jolla poistetaan tietokannasta tietue, jonka pääavain on 0000-1111-2222-33332.

3.4 Fetch API

Fetch API tarjoaa rajapinnan, jonka avulla noudetaan resursseja internetistä. Rajapinta tarjoaa määritykset Request- ja Response-olioille sekä muille asioille, jotka liittyvät verkkopyyntöihin. Määritysten ansiosta resursseja voidaan käyttää milloin vain tulevaisuudessa (Fetch API, N.d).

Tässä tutkimuksessa Fetch APIa käytetään Spring Boot rest-palvelun yhteydessä. Sillä annetaan tiedon tallennuksen yhteydessä POST kutsu Spring boot restful APIssa määrittettyyn osoitteeseen, jolloin rest palvelun controller-luokat aktivoituvat. POST-kutsun yhteydessä lähetetetään Javascript-oliosta muodostettu json-merkkijono, joka muutetaan rest-palvelun toimesta Java-olioksi, eli entiteeksi ja se tallennetaan tietokantaan.

```
fetch('http://localhost:9090/api/addOne',{
    method:'POST',
    body:JSON.stringify(viesti),
    headers:{
        'Content-Type': 'application/json'
    }
}).then(res=>res.json())
.catch(error=>console.log("error: "+error))
.then(response => console.log("Response: "+response));
```

Kuvio 5. Tiedon tallentaminen tietokantaan käyttäen Fetch APIn POST-metodia.

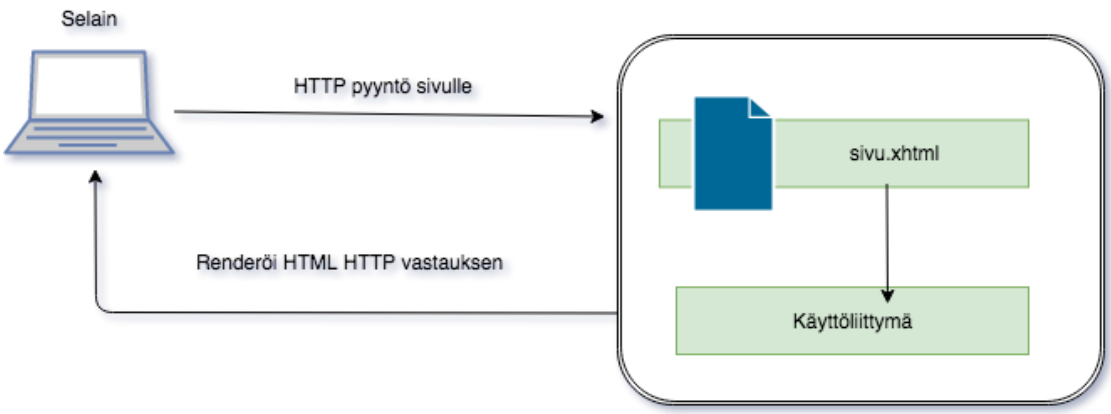
Fetch APIa käytetään myös tiedon hakemiseen tietokannasta. Spring Boot rest-palvelussa on määritettynä polku, jota käytettäessä haetaan kaikki tietueet tietokannasta. Fetch APIlla tämä onnistuu GET-metodilla, joka tekee pyynnön polkuun. Sieltä saatava vastaus muutetaan json-muotoon ja se asetetaan muuttujaan. Tätä muuttujaa käytetään tietoperustana tietopöydille(datatable).

```
let data;

fetch('http://localhost:9090/api/all',{
    method:'GET',
    headers:{
        'Content-Type': 'application/json'
    }
}).then(res =>res.json())
.catch(error => console.log("error: "+error))
.then(response =>

    this.setState({
        testData:response
    })
);
```

Kuvio 6. Fetch APIn GET-metodi,jolla pyydetään tiedot spring boot-palvelusta ja paluuarvo asetetaan Reactin state muuttujaan.

3.5 JavaServer Faces

JavaServer Faces on Oraclen luoma ja kehittämä palvelinpuolen komponenttikäyttöliittymä Java-pohjaisille sovelluksille. JavaServer Facesilla luotu käyttöliittymä pysyy palvelimen päässä, johon otetaan http yhteys selaimesta. Palvelimella sijaitseva sivu luo käyttöliittymän, joka palautetaan ja renderöidään HTML-vastauksena selaimeen (JavaServer Faces Technology User Interface 2010).



Kuvio 7. JavaServer Faces teknologian toimintaperiaate.

JavaServer Faces on Javan webkäyttöliittymästandardi, sekä ohjelmistokehys, joka toteuttaa Model-View-Controller-mallia. Tämä tekee sovelluksista paljon ylläpidettävimpiä, sillä käyttöliittymän luova koodi on eriytettynä toiminnallisuudesta (JSF Architecture N.d).

Tätä tutkimusta varten saatu sovellus käyttää JavaServer Facesin lisäksi PrimeFaces-kirjastoa, joka laajentaa ennestään käytettävissä olevia rakenteita.

3.6 React

React on Facebookin tuottama ohjelmistokehys, jolla renderöidään HTML-rakenteeseen sisältöä tai uusia elementtejä käyttäen Javascriptin Ecmascript 6 standardia, sekä JSX syntaksilisäystä. Ennen renderöintiä ReactDOM vertailee uusia elementtejä ja niiden lapsia jo olemassa oleviin, jolloin saadaan päivitetyttä ainoastaan muutoksen saaneet osat. JSX muistuttaa HTML templaattikieltä, mutta siihen on yhdistettynä myös Javascriptiä. JSX tuottaa React komponentteja, jotka renderöidään HTML rakenteeseen (Introducing JSX N.d).

React-komponentit ovat normaaleita Ecmascript-luokkia, jotka perivät React.Component-yliluokan. Tämänlaisten komponenttiluokkien sisään luodaan funktioita, joita kutsumalla saadaan toteutettua sovellutuksen toiminnallisuutta. Pakollisia funktioita React.Component tyyppisille luokille on ainoastaan render, joka nimensä mukaisesti palauttaa näkymään jotain, eli ns. renderöi sivulle tietoa (React.Component N.d).

```
class Header extends React.Component{

    render(){
        return(
            <div className="Header">
                <h1 className="HeaderH1"> Anna palautetta ja kysy</h1>
            </div>
        );
    }
}
```

Kuvio 8. Yksinkertainen Ecmascript 6 luokka, joka perii React.Component yliluokan.

Näitä React-komponentteja luodaan kutsumalla luokan nimiä ikään kuin HTML-elementtiä. Esimerkkinä ylläolevan kuvan Header-luokkaa kutsutaan tagilla <Header />. Luokkien sisällöt renderöidään sivustolle käyttämällä Reactin ReactDOM.render metodia.

React-dom on paketti, joka tarjoaa DOM spesifejä metodeja, joita käytetään ylimmällä tasolla ikään kuin pakoreittinä ulos React modelista (ReactDOM N.d). React-

dom paketti tuodaan Ecmascript 6 tyylin mukaisesti käytettävään Javascript tiedostoon.

```
import ReactDOM from 'react-dom';
```

Kuvio 9. React-dom paketin tuominen ja käyttöönottaminen EC6-standardin mukaisesti.

3.6.1 Props ja state

React koodissa komponenttien välistä kommunikointia hoidetaan props-olioilla ja sitä käytetään avain-arvo periaatteella komponenttia kutsuttaessa. Propsit, eli propertiesit kulkevat isäntäkomponentista lapsikomponenttiin päin ja ne ovat muuttumattomia(immutable).

Tilallisuutta(state) käytetään silloin, kun komponentin halutaan päivittävän komponenttia tietojen muuttuessa. Tilallisessa komponentissa setState-kutsua käytettäessä päivitetään tieto state-muuttujaan, jonka seurauksena kyseinen dom-elementti renderöidään uudelleen.

Mikäli komponentille ei tarvita tilallisuutta(state), voidaan se luoda ecmascript 6 mukaisesti normaaliksi funktioksi, joka palauttaa JSX-rakenteen.

```
//Ei tilallisia funktioita -> ei tarvitse olla luokka tai periä mitään.
const Header = () => <div><h1 className="HeaderH1">Anna palautetta ja kysy</h1></div>;
```

Kuvio 10. Yksinkertainen Ecmascript 6 funktio, joka palauttaa JSX-rakenteen.

3.6.2 Riippuvuudet

Riippuvuuksien lisääminen Mavenilla rakennettuun React-projektiin on helppoa package.json-tiedoston ansiosta. Tähän tiedostoon kirjataan riippuvuuksien(dependecies) alle tarvittavat paketit, jotka build-komennon yhteydessä ladataan projektiin node-moduleina. Riippuvuudet otetaan käyttöön import-määreellä.

3.7 Angular

Angular on Googlen kehittämä sovellusalusta, sekä ohjelmistokehys, joka on tarkoitettu HTML- ja Typescript-pohjaisten sovellusten kehittämiseen. Angular

pohjaiset sovelluksen sisältävät juurimodulin nimeltä AppModule, joka tarjoaa bootsrap mekanismin sovelluksen käynnistämiselle. Tämänkaltaisilla moduleilla tuodaan ja määritetään tarvittavat riippuvuudet sovellukseen ja ne määritetään @ngModule notaatiolla metadatassa.

Moduleissa määritetään myös Angular komponenttit, jotka luovat sisältöä sovelluksen juuri templaattiin. Angular komponentit määritetään @component notaatiolla, jonka metadataksi voidaan asettaa halutessa mm. selector, joka valitsee hetken, milloin kyseistä komponenttia käytetään sekä templateUrl:n, joka osoittaa komponetti HTML tiedostoon(Architecture Overview 2018).

```
@Component({
selector:'app-form',
templateUrl:'/o/palaute-angular/js/app/form/form.component.html'

})
```

Kuvio 11. Angular-komponentti, jossa määritetään valittava elementti, sekä näytettävä templaatti.

3.7.1 Data binding

Datan liikkuminen komponentin ja templaatin välillä tapahtuu data-bindingilla, jossa komponentti hoitaa datan muodostumisen ja templaatin tehtävänä on esittää se. Data-bindingit jaetaan kolmeen kategoriaan datan liikkumisen perusteella; from source-to-view, from view-to-source, sekä view-to-source-to-view. Kaksi ensimmäistä kategoriaa ovat niin sanottuja one-way tyylisiä databindejä, joissa data kulkee yhteen suuntaan. Viimeiseksi mainittu näistä kolmesta on kahden suunnan eli Two way databindingia, jossa data virtaa samaan aikaan source ja viewin välillä. Sitä käytetään silloin, kun halutaan näyttää muuttujan arvo niin, että se päivittyy reaaliajassa sivulle sekä sourceen(Template syntax 2018).

3.7.2 Riippuvuudet

Riippuvuuksien lisääminen Angular sovellukseen tapahtuu package.jsoniin lisättävien määreiden avulla, jotka sitten julistetaan app.modulessa, sekä tuodaan haluttuihin komponentteihin import-määreen avulla(NPM packages 2018).

3.8 Sovelluksen elinkaarenhallinta

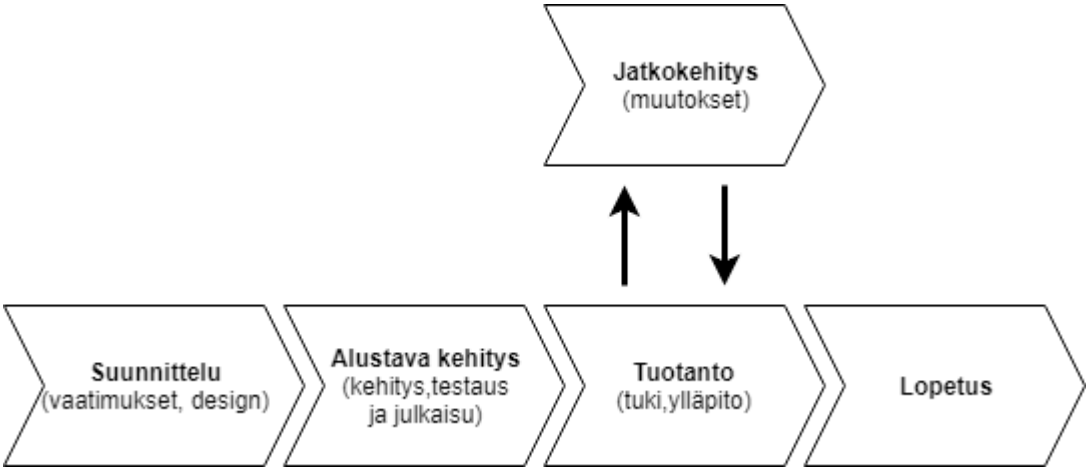
Kelassa kehitetään useita sovellutuksia useisiin tarpeisiin ja useimmiten niiden elinkaari on pitkä, joissakin tapauksissa jopa kymmeniä vuosia. Tästä syystä sovellusten elinkaari on mietittävä ja dokumentoitava tarkasti.

Sovelluksen elinkaarella tarkoitetaan jatkumoa projektin aloituksesta loppuun saakka, eli toisin sanoen sovelluksen elinkaari jaetaan päämääräisesti kahteen osioon, kehitykseen sekä sovelluksen ylläpitoon. Kehityksen aikana määritetään tarve uudelle sovellukselle asiakkaan toimesta ja samalla määritetään vaatimukset, mitä sovelluksen on tehtävä. Tästä eteenpäin sovelluksen tekemiseen määrätty ryhmä selvittää, onko sovellus mahdollista kehittää määritysten puitteissa, sekä onko projekti taloudellisesti, käytännöllisesti, sekä teknologisesti mahdollista toteuttaa.

Alkujärjestelyiden jälkeen aloitetaan sovelluksen suunnittelu, jossa otetaan määrityksen uudelleen käsiteltäväksi. Tässä vaiheessa käytetään myös hyväksi käyttäjien mielipiteitä ja näkökantoja, jotta sovelluksesta saataisiin määrityksiä vastaava. Kun sovellus on saatu suunniteltua, ruvetaan sitä ohjelmoimaan. Ohjelmoinnin kanssa tehdään testejä, jotta varmistutaan sovelluksen täyttävän määritykset. Yleinen ohje on, että vähintään 50% koko sovelluksesta olisi testattava.

Kehityksen lopussa valmis ja testattu sovellus viedään tuotantoon, jolloin se siirtyy ylläpitoon. Ylläpitoon kuuluu kaikkien tuotannossa huomattujen virheiden korjaaminen, sekä sovelluksen varmistaminen, että toimivana pysyminen (Software development life cycle N.d). Mikäli tuotannossa huomataan isoja virheitä, on niiden korjaaminen erittäin kallista ja aikaa vievää.

Kustannusten jakautuminen on isossa osassa elinkaarenhallintaa. Tutkimuksessaan Zarnekow ja Brenner totesivat kustannusten suurelta osion painoittuvan projektin ylläpito- ja tuotantoajalle. Tutkimuksessa havaittiin toistuvien ja ei toistuvien kustannusten jakaantuvan 4:1 suhteeseen projektissa, joka oli tuotannossa viisi vuotta. Tästä vedetyssä johtopäätöksessä suhdeluku vain kasvaa, mitä pidempään projekti tuotannossa on(Zarnekow, R. Brenner, W 2005).



Kuvio 12 Sovelluksen elinkaarenhallintakaavio alusta loppuun.

4 Tutkimus

4.1 Spring Boot Restful CRUD Service

Ennen React- ja Angular-sovellusten kehittämistä, luodaan Spring boot rest palvelurajapinta, jota käytetään sovellusten ja tietokannan väliseen kommunikointiin. Spring boot sovellus on ns. standalone-tyyppinen, eli projekti luo oman sovelluspalvelimensa, jonka päällä se pyörii. Sovelluspalvelin toimii portissa 9090 ja vaatii toimiakseen @SpringBootApplication-notaation, sekä staattisen main-metodin, jossa luodaan aina uusi Spring-applikaatio.

```
## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url = jdbc:mysql://localhost:3306/oppariDB?useSSL=false
spring.datasource.username = root
spring.datasource.password = root

spring.jpa.show-sql=true

## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto (create, create-drop, validate, update)
#spring.jpa.hibernate.ddl-auto = update

inspring.jpa.hibernate.naming.implicit-strategy=org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyJpaImpl
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

server.port=9090
```

Kuvio 13. Sovelluksen application.properties-tiedosto, jossa tietokantayhteys määritetään.

4.1.1 Entity

Spring-palvelu koostuu Entity-luokasta, joka kuvastaa tietokantataulussa olevaa tietueriviä. Tässä tietokantataulussa oleva data siirretään Java-olioon(POJO), jotta sitä voidaan käyttää tulevaisuudessa perinteisillä Javan olio-ohjelmointitavoilla. Entityluokka annotoidaan @Entity-notaatiolla, sekä vielä erikseen @Table-notaatiolla, jolla määrätään oliota vastaava taulu. Luokasta löytyy samat muuttujat kuin tietokantataulussa olevat sarakkeet (Mapping with JPA 2004).

```java
package fi.oppari.mysql.service.SpringService.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;



@Entity
@Table(name = "palauteUser")
public class PalauteUser {

    @Id
    private String userUUID;

    @Column(name = "firstname")
    private String firstName;

    @Column(name = "lastname")
    private String lastName;

    @Column(name = "email")
    private String email;

    public String getUserUUID() {
        return userUUID;
    }

    public void setUserUUID(String userUUID) {
        this.userUUID = userUUID;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

}
```

Kuvio 14. Tietokantataulua vastaava Javan Entity-luokka.

4.1.2 Repository

Spring-sovelluksessa tietokantayhteyttä käsitellään repository-rajapintojen välityksellä. Repository luodaan yhtä luokkaa vastaamaan ja se perii halutessaan JpaRepo-

sitory- tai CrudRepository-rajapintaa. Tässä tutkimuksessa luotu PalauteUserRepository perii Springin JpaRepositoryn ja sille asetetaan arvoiksi PalauteUser -luokka, sekä yksilöivän arvon tyypiksi String, sillä PalauteUser-luokan pääavain(@Id) on tyyppiä String. Luodussa rajapinnassa on valmiit metodit tallentamiseen, tiedon hakemiseen, sekä poistamiseen.

```
package fi.oppari.mysql.service.SpringService.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import fi.oppari.mysql.service.SpringService.model.PalauteUser;

@Repository
public interface PalauteUserRepository extends JpaRepository<PalauteUser, String>{

}
```

Kuvio 15. PalauteUseria varten luotu repository, joka perii JpaRepositoryn.

4.1.3 Controller

Palvelussa olevia metodeja käsitellään urlin avulla, ja tähän tarvitaan oma controlleri. Controller on Java-luokka, joka annotoidaan @RestController- sekä @RequestMapping-notaatioilla. Luokan tehtävänä on kuunnella rajapintaan tulevia kutsuja, ja ohjata sitä kautta kutsut oikeille metodeille.

Controller luokassa on viittaus käytettävään repositorioon, ja se on varustettu @Autowired-notaatiolla. Luokassa olevat metodit annotoidaan niiden käyttötarkoituksesta riippuen. Esimerkiksi tilanteessa, jossa halutaan hakea kaikki tietueet tietokannasta, voitaisiin käyttää annotaatiota @GetMapping(”/all”). Tämä tulee lyhenteestä @RequestMapping(value=”/all” method=RequestMethod.GET) ja se kuuntelee urlosoitetta /all sekä vaatii toimiakseen GET-tyylisen kutsun.

```java
package fi.oppari.mysql.service.SpringService.controller;

import java.util.List;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import fi.oppari.mysql.service.SpringService.exception.ResourceNotFoundException;
import fi.oppari.mysql.service.SpringService.model.PalauteUser;
import fi.oppari.mysql.service.SpringService.repository.PalauteUserRepository;

@RestController
@RequestMapping("/api")
public class PalauteUserController {

    @Autowired
    PalauteUserRepository palauteUserRepository;

    @GetMapping("/all")
    public List<PalauteUser> getAll() {
        System.out.println("Getting all");
        return palauteUserRepository.findAll();
    }

    @PostMapping("/addOne")
    public PalauteUser addUser(@Valid @RequestBody PalauteUser user) {
        return palauteUserRepository.save(user);
    }

    @DeleteMapping("/deleteOne/{uuid}")
    public ResponseEntity<?> deleteUser(@PathVariable(value = "uuid") String UUID) {

        PalauteUser user = palauteUserRepository.findById(UUID)
                .orElseThrow(() -> new ResourceNotFoundException("User", "userUUID", UUID));
        palauteUserRepository.delete(user);

        return ResponseEntity.ok().build();

    }

}
```

Kuvio 16. RestController-luokka, jossa toiminnallisuutena kaikkien tietojen hakeminen, sekä yksittäisen lisääminen ja poistaminen.

Tietokantana toimii lokaalilla palvelimella oleva mysql, johon on luotu yksinkertainen taulu palauteUser, johon on asetettu testidataksi muutama tietue.

4.2 Pisteytys

Tutkimukseen asetetuista vertailukohdista muodostetaan omat kappaleet. Näissä kappaleissa arvioidaan ohjelmistokehyskohtaisesti arvioitava vertailukohta. Vertailu-

kohtien arviointi toteutetaan pisteytyksellä 0-5, viiden ollessa paras arvosana. Tutkimuksen päätteeksi ohjelmistokehyksistä muodostetaan taulukko, johon sijoitetaan vertailukohtien arvosanat, sekä tulokset kirjoitetaan auki raportin muodossa.

4.3 Ohjelmistokehysten vertailukohdat tutkimuksessa

Tutkimukseen valittavien vertailukohtien määrittämiseksi tutkija haastatteli kahta toimeksiantajan riveissä työskentelevää kehittäjää. Kehittäjät ovat toimineen sovelluskehitystehtävissä Liferayn kanssa useiden vuosien ajan, joten näkemys vertailukohdista on vahva. Vertailukohdat määriteltiin tutkimuskysymyksien pohjalta kahteen kategoriaan; kehittämiseen ja ylläpitämiseen.

Ylläpitämisen näkökulmasta vertailukohdiksi valikoituivat seuraavat

- Pysyvyys
    - Selvittää suurimmat muutokset ja niiden vaikutus ohjelmistokehykseen, sekä kehittämiseen
- Ohjelmistokehyksen historia
    - Perehdytään ohjelmistokehyksen historiaan käytön kannalta

Kehittämisen näkökulmasta vertailukohdiksi valittiin seuraavat

- Pystyykö ohjelmistokehys vastaamaan toimeksiantajan tarpeisiin
    - Rakenteellisten vaatimusten täyttäminen (tietotaulu, kalenteri, haitarirakenne sekä suodatus)
- Kehittämisen nopeus sekä rakentuminen
    - Rakentumisella tarkoitetaan koodin selkokielisyyttä
- Ohjelmistokehyksestä löytyvä dokumentaatio
    - Selvitetään teknisen dokumentaation saatavuus

4.4 Pysyvyys

Pysyvyydellä tarkoitetaan yleensä ohjelmistokehyksen historiassa tapahtuneita muutoksia, sekä kehyksen yhteensopivuutta aiempiin versioihin.

4.4.1 React

Ensimmäinen julkinen Reactin versio julkaistiin heinäkuussa 2013 versionumerolla 0.3.0 ja tätä tutkimusta tehtäessä viimeisin versio on 16.3.0. Githubista löytyvästä

Reactin changelogista voidaan päätellä, että suurimpia muutoksia koko ohjelmistokehyksen elinkaaren aikana on ollut versiossa 0.14.0 React- ja ReactDOM-pakettien erittäminen omiin node-moduuleihin (React ChangeLog 2018). Uudistuotantoon tämä ei enää vaikuta, sillä tässä tutkimuksessa luoto React-pohjainen sovellus käyttää näitä erillisiä moduuleita, mutta vastaavat muutokset tulevaisuudessa voivat aiheuttaa isossa mittakaavassa suuriakin muutoksia.

Pysyvyyden näkökulmasta suurimmat muutokset tulevat major-päivitysten yhteydessä, ja nämä versio numeroidaan tasaluvuin. Tämä helpottaa sovelluksen ylläpitoa, sillä versiot on helppo tunnistaa. Reactin pysyvyydelle annetaan neljä pistettä.

4.4.2 Angular

Angular 2 ensimmäinen versio on nimensä mukaisesti 2.0.0 ja se julkaistiin 14.9.2016. Tutkimusta tehtäessä viimeisin versio Angularista on 6.0.0-rc.3. Versionumeroinnin alkaminen 2.0.0 juontaa taustansa Angularin aiempaan versioon, AngularJs ,joka on kokonaan erilainen ohjelmistokehys rakenteeltaan ja toimivuudeltaan.

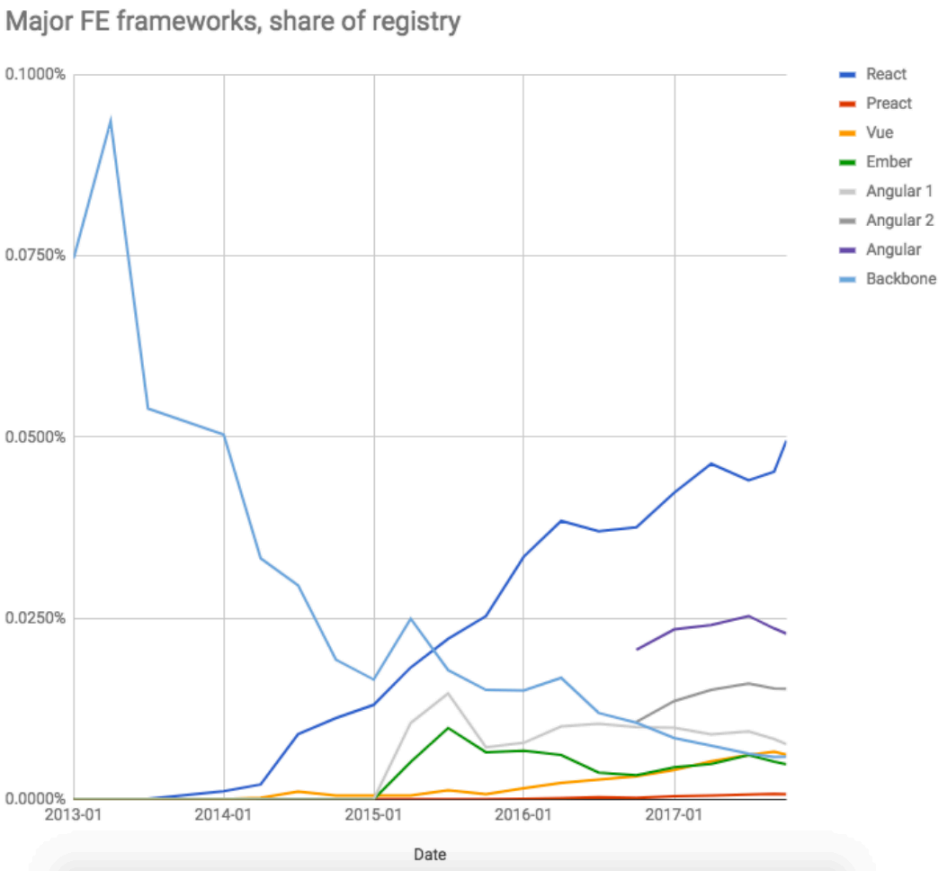
Angularin Githubista löytyvän changelogin perusteella versionumerointi noudattaa samaa yleistä mallia kuin React ja suurimmat muutokset tulevat major-päivityksissä. Versiossa 5.0.0 on määritetty rikkovaksi muutokseksi Typescript 2.4.x version käyttäminen (Angular ChangeLog 2018). Tämä on suhteellisen iso muutos, sillä ohjelmointikielien versioiden välillä voi olla suuriakin muutoksia. Pysyvyydestä Angular saa neljä pistettä.

4.5 Ohjelmistokehysten historia

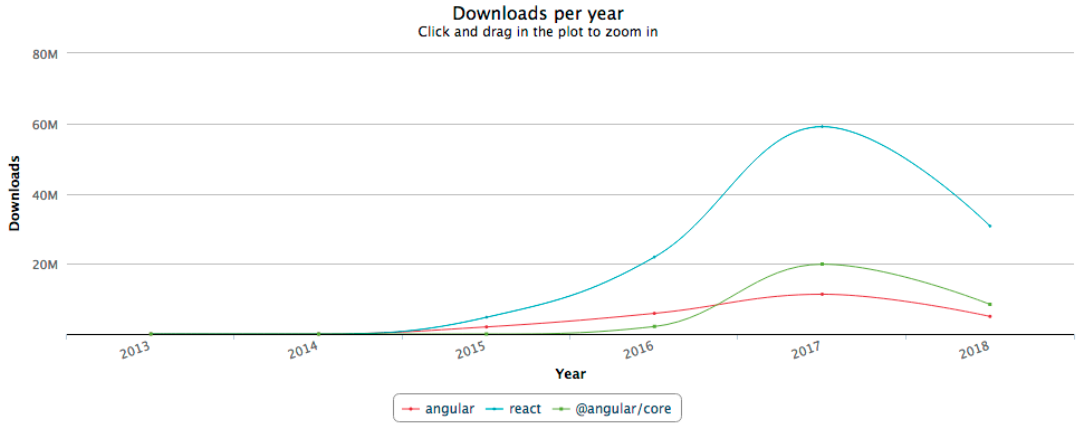
4.5.1 React

Reactin käyttö on julkaisunsa jälkeen kokenut jättimäisen kasvun. Osasyynä tähän räjähdysmäiseen suosioon on ohjelmistokehyksen luonteessa. Sen modulaarisuus ja sopivuus sovelluksen tarpeisiin on hyvä. Tästä syystä React ohjelmistokehys on synnyttänyt oman ekosysteeminsä, johon liittyvillä paketeilla voidaan ratkaista useita ongelmia (The State of Javascript Frameworks,2017 2018).

React hallitsee ohjelmistokehyksenä markkinoita niin käytön kuin lataustenkin suhteen Angulariin verrattuna. Käyttöhistoriasta Reactille viisi pistettä



Kuvio 17. Major FE frameworks, share of registry(The State of Javascript Frameworks,2017 2018).



Kuvio 18. Downloads per year (Paul Vorbach 2018).

4.5.2 Angular

Edellämainittujen käyrien perusteella Angular toiseksi käytetyin ohjelmistokehys web-applikaatioiden ja sivujen käyttöliittymän luomisessa, vaikka se julkaistiinkin

kolme vuotta Reactia myöhemmin vuonna 2016. Angular 2:n käyttäjämääriin vaikuttanee vanhemman AngularJs:n yhtäaikainen ylläpito ja tuki.

Käyttöhistorian ja kehyksen päivitystieheydestä Angularille viisi pistettä.

4.6 Toimeksiantajan rakenteelliset vaatimukset

Kelan asettamat minivaatimukset ohjelmistokehykselle ovat tietotaulu, kalenteri, ja haitarirakenne.

4.6.1 React

Reactin ydinkirjasto on suhteellisen suppea erikoiskomponenttien osalta, mutta se on helposti laajennettavissa package.json-tiedoston avulla, jonne lisätään koko projektissa käytettävät riippuvuudet. Githubissa on tarjolla laaja valikoima erilaisia komponentteja, joilla pystytään laajentamaan Reactin ydinkirjastoa.

Tutkimuksen vaatimuksena oleva tietotaulu toteutettiin Joaocarmon React-smart-data-table-komponentilla (Joaocarmo 2018). Tämä on erittäin helppokäyttöinen komponentti, joka ei vaadi suurta konfigurointia. Tietotaulu on lajiteltavissa otsikon perusteella laskevaan tai nousevaan järjestykseen.

Reactin kalenteri toteutettiin Wojtekmajn tarjoamalla React-calendar-komponentilla (Wojtekmja 2018). Kalenterissa on datepicker-mahdollisuus, jolla saadaan valittua haluttu päivämäärä. Komponentin käyttöönotto oli ongelmallinen, sillä jostain syystä css-luokkia ei onnistuttu lataamaan. Komponentin sai kuitenkin toimimaan ilman tyylittelyä.

Haitarirakenteen tässä tutkimuksessa toteutin Glenn Flanaganin tarjoamalla komponentilla React-collapsible(Flanagan 2018). Toteutus on yksinkertainen ja helppo ottaa käyttöön. Haitarirakenteen tyylittely onnistuu css-tiedostoon asetetuilla määreillä helposti ja nopeasti.

Laajan tarjonnan myötä Reactia saadaan laajennettua reilusti. Vaatimuksien täyttämisestä neljä ja puoli pistettä, sillä kalenteri -komponentin käyttöönottaminen ei ollut aivan ongelmaton.

4.6.2 Angular

Angularin kirjastoa on helppo laajentaa npm-riippuvuuksia lisäämällä package.json-tiedostoon. Nämä riippuvuudet pitää kuitenkin lisätä vielä erikseen app.module.ts tiedostoon sekä app.componen.ts tiedostoon.

Kuten Reactille, on myös Angularille useita kolmannen osapuolen tarjoamia kirjastoja rakenteellisten vaatimusten toteuttamiseen. Toisin kuin React, tarjoaa Angular oman Material-komponenttikirjastonsa, jolla Kelan määrittämät rakenteelliset vaatimukset olisi helppo toteuttaa. Tutkimuksen edetessä kuitenkin havaittiin ongelmia Liferayn tarjoaman npm bundlerin ja npm-moduulien kanssa. Npm bundler ei osannut kääntää oikein npm-moduuleita tarvittavaan OSGi-bundle muotoon. Tästä syystä koko projekti kaatui. Ongelmien johdosta Angularin osalta rakenteellisia vaatimuksia ei pystytty toteuttamaan.

Vaikka riippuvuuksien kanssa oli ongelmia, on Angularille kuitenkin tarjolla reilusti ratkaisuita vaatimusten täyttämiseen, jolloin toimivan npm bundlerin kanssa Angular ohjelmistokehyksellä olisi helppo luoda kalentereita, accordioita sekä tietotauluja.

Vaatimusten täyttämisestä Angularille yksi piste.

4.7 Kehittäminen

4.7.1 React

React-portletin luominen ja käyttöönottaminen on helppoa Blade cli-työkalulla. Blade luo oikean tiedostorakenteen projektille, sekä tarvittavat riippuvuudet ydinsovelluksen pyörittämiseen.

Käyttöliittymän luominen Reactilla on nopeaa ja suhteellisen vaivatonta perus rungon suhteen. Reactin ydinkirjastossa ei sisäänrakennettua validointia, joten input-fieldien validointi on toteuttettava joko omalla koodilla, tai npm-pakettina saatavilla komponenteilla.

Liferay-portaalina ei tuo juurikaan muutoksia kehittämiseen, joten samat säännöt pätevät suurilta osin kehittämistä. Riippuvuuksien lisääminen projektiin onnistuu

package.jsonin avulla, joskin joidenkin riippuvuuksien käyttöönottamisessa oli tutkimuksen aikana vaikeuksia. Projektin kääntäminen Mavenin avulla on helppoa ja vaivatonta.

Koodi on helposti luettavaa, sillä se kirjoitetaan Javascriptillä ja erillisiä tiedostoja luokille ei ainakaan tässä tutkimuksessa tarvittu. JSX hoitaa HTML rakenteen luomisen ja sen syntaksi onkin samanlainen kuin HTML5. Tämä on huomattavasti helpompaa lukea, kuin esimerkkisovelluksen JavaServer Faces-koodi.

```
const Aihe = (props) =>{
    return(
        <div className="AiheDiv">
            <p>Viestini koskee</p>
            <input id="kantaRadio" type="radio" checked={props.isKantaPalveluita} onChange={props.vaihdaKanta}/>
            <label htmlFor="kantaRadio">
                <span> Kanta-palveluita </span>
            </label>
            <input id="ongelmaRadio" type="radio" checked={!props.isKantaPalveluita} onChange={props.vaihdaKanta}/>
            <label htmlFor="ongelmaRadio">
                <span> yksityishenkilön teknistä ongelmaa</span>
            </label>
        </div>
    )
}
```

Kuvio 19. Esimerkki JSX syntaksista sekä Ecmascript 6 tilattomasta funktiosta.

Vaikka React on simppeli ja helppo ohjelmisto kehys, on se kuitenkin ydinkirjastoltaan suhteellisen suppea Kelan tarpeisiin. Kolmannen osapuolen komponentit parantavat tilannetta osaltaan, mutta niidenkin käyttäminen on jatkuvalla syötöllä työlästä ja ulkopuolisten riippuvuuksien käyttäminen hidastaa kehitystä. Reactille annetaan neljä pistettä kehittämisestä.

4.7.2 Angular

Angular-projektin luominen käy yhtä helposti Blade CLI työkalulla kuten Reactin vastaava. Blade luo Maven projektille tarvittavat tiedostot, sekä kansiorakenteen.

Käyttöliittymän kehittäminen Angularilla on jo hieman monimutkaisempaa verrattuna Reactiin. Syynä tähän on logiikan toteuttaminen useammassa eri tiedostossa (komponentit, templaatit ja moduulit). Alun ontumisen jälkeen sovelluslogiikka selveni, ja kehittäminen nopeutui selvästi. Projektin rakentaminen onnistuu Reactin portletin tavoin Mavenilla vaivatta.

Angular on rakenteeltaan hyvin erilainen kuin React, ja sitä sanotaankin kunnon ohjelmistokehykseksi siinä missä Reactia kehutaan kirjastoksi. Tämä osoittautui eduksi,

sillä Angularin two way data binding on helposti toteutettavissa, ja sen avulla olioiden tekeminen on helppoa ja yksinkertaista. Myöskin käyttäjän virheellisten syötteiden tarkastaminen on helppoa ja vaivatonta.

```
<p>Lahettaja *</p>
<input type="text" name="lahettaja" id="email" [(ngModel)]="model.lahettaja" #lahettaja="ngModel" required>
<div *ngIf="lahettaja.invalid && (lahettaja.dirty || lahettaja.touched)" class="alert alert-danger">
    <div *ngIf="lahettaja.errors.required">
        Lähettäjä puuttuu
    </div>
</div>
```

Kuvio 20. Esimerkki Angularin ngModel two way data bindingista, sekä validoinnista.

Syntaksiltaan Angularin koodi on perus HTML5 kieltä, lisättynä kuitenkin omilla kustomoidulla elementeillä. Pientä epävarmuutta ja sekaannusta kehittämiseen toi variaatiot, joilla esimerkiksi elementtiä painaessa tapahtuu jotain. Angularissa se voidaan tehdä ng-click tai (click) määreillä haluttuun elementtiin.

Alun jälkeen Angularilla kehittäminen on suhteellisen helppoa, joskin moduulien ja komponenttien kanssa säätäminen vie aikaansa. Kehittämisestä Angularille neljä pistettä.

4.8 Dokumentaatio

4.8.1 React

Reactin dokumentaatio on aloittelijaystävällinen ja siinä käydään läpi perusteet käyttöliittymän luomiseen, sekä vähän pidemmälle kehittyneille kehittäjille lisätietoa Reactin ominaisuuksista. Dokumentaation puolesta Reactin API on melkoisen suppea ja siinä käydäänkin läpi vain ylimmän tason rajapinnat (Docs N.d).

Liferayn puolesta React-portletille ei löydy myöskään hirveästi dokumentaatiota, joten monessa asiassa parhaimpana dokumentaationa toimikin Githubissa olevat React-portletit sekä Liferayn Slack ja forumit.

Internet on täynnä natiiviin Reactiin perustuvia ohjeita ja sen kehittäminen Liferay ympäristössä ei eroa juurikaan node palvelimella kehittämisestä.. Dokumentaatiosta kolme pistettä.

4.8.2 Angular

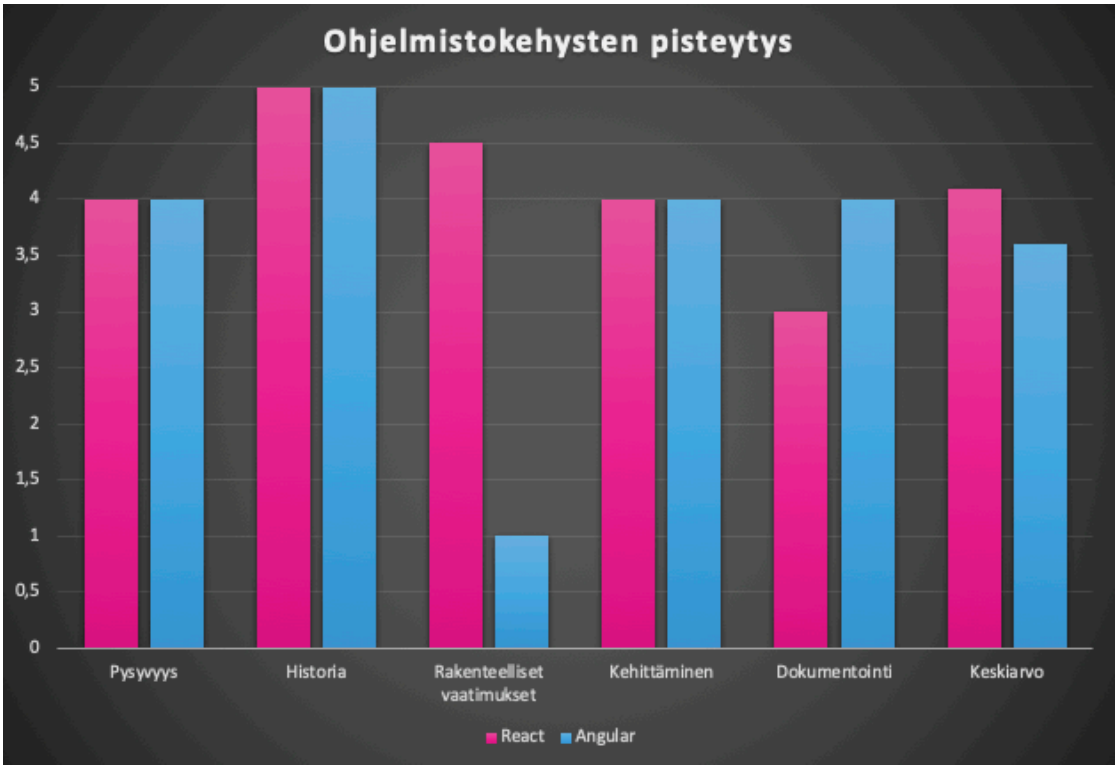
Angularin dokumentaatio vastaa jo enemmän ohjelmistokehyksen tai ohjelmistoalustan dokumentaatiota. Sieltä löytyy aloittelijalle hyvät pohjatiedot kehittämiseen, kuvauksia arkkitehtuurista, tietoa komponenteista ja moduleista sekä paljon muuta. Angularin rajapintadokumentaatiokin on huomattavasti kattavampi kuin Reactin vastaava ja se tarjoaa tietoa Angularin ratkaisuista (What is Angular? 2018).

Angularin kehittäminen ei juurikaan eroa Liferay-ympäristössä normaaliin Nodejs pohjaiseen kehittämiseen verrattuna, joten suurin osa Googlen tarjoamista ohjeista on valideja. Joitakin asioita on kuitenkin jouduttu muuttamaan, ja niistä tarjottava dokumentaatiot Liferayn puolesta ovat heikkoja.

Kuten Reactilla, niin myös Angularilla on paljon käyttäjiä. Tämän vuoksi internetissä on reilusti ohjeita ja vinkkejä erilaiseen kehittämiseen ja useaan ongelmaan löytyy ratkaisu jo hetken etsimisen jälkeen. Näiden perusteilla Angularin dokumentaatiosta neljä pistettä.

5 Tutkimuksen yhteenveto

Tutkimuksen aikana kehitettiin esimerkkisovelluksena toimivaa palautelomaketta vastaavat versiot React. ja Angular-ohjelmistokehyksillä Liferay-ympäristöön. Tutkimukselle asetettuihin tutkimuskysymyksiin saadut vastaukset arvioitiin pisteyttämällä.



Kuvio 21. Tutkimuksen tulokset pisteytettynä tutkimuskysymyksien mukaan.

Käyttöliittymän vaivattomuuteen selvittävä kysymys jaettiin useampaan osioon, johon lukeutuivat havainnot kehittämisen nopeudesta ja rakentumisesta, ohjelmistokehyksestä löytyvästä dokumentaatiosta, sekä ohjelmistokehyksen rakenteelliset vastaavuudet toimeksiantajan vaatimuksiin.

Kummallakin ohjelmistokehyksellä kehittäminen on suhteellisen nopeaa ja selkokielistä, sillä syntaksiltaan kummatkin muistuttavat HTML kieltä. Tämä on varsinkin uuden kehittäjän, jolla ei ole kokemusta sovelluskehittämisestä, silmään helpompaa lukea kuin JavaServer Faces.

Ohjelmistokehysten dokumentoinnissa havaittiin hieman poikkeavaisuutta tutkittavien kehysten kesken, sillä Angularilla oli jonkin verran kattavampi selostus käytetävistä tekniikoista. Kummaltakin löytyy dokumentaatio peruskehitykselle, mutta Angularin vastaavassa oli laajemmin käyty läpi sovelluskehitykselle ominaisia asioita.

Suurin eroavaisuus käyttöliittymän vaivattomuuteen liittyvissä tutkimuskysymyksissä oli rakenteellisten vaatimusten täyttämisessä. Tutkimusta tehtäessä oli suuria vaikeuksia Liferayn npm bundlerin, sekä npm-moduulien kompiloinnin kanssa, josta syystä Reactin puolella jouduttiin kokeilemaan useampaa komponenttia, ennen kuin

löytyi sopivat. Angularin kanssa rakenteellisia vaatimuksia ei edes saatu toteutettua juuri samaisen kompilointiongelman vuoksi.

Ohjelmistokehyksen ylläpitoon asetettu tutkimuskysymys jaettiin pysyvyyteen sekä ohjelmistokehyksen historiaan.

Pysyvyyden selvittävässä kysymyksessä havaittiin kummankin ohjelmistokehyksen olevan nykyiseltään suhteellisen vakaa ja suuri muutoksia ei ole ollut hetkeen. Kummassakin on alkuaikoina tapahtunut suurempia muutoksia, joka on uusille ohjelmistokehyksille ominaista. Ohjelmistokehysten numerointi seuraa loogista kaavaa ja suurimmat muutokset tulevat ns. major-päivityksissä, jolloin versionumeroinnissa muutetaan joko ensimmäistä desimaalia tai kokonaislukua. Kummatkin ohjelmistokehykset saavat tasaiseen tahtiin päivityksiä.

Käyttöhistorian valossa Reactia käytetään enemmän, joskin se on ollut hieman pidempäänkin markkinoilla. Angularin käyttöä laskenee rinnakkain olevan AngularJS:n olemassaolo, joka on nykyisen Angular 2 kehyksen aiempi versio.

Kaiken kaikkiaan kummatkin ohjelmistokehykset ovat vahvoja ja hyviä kehyksiä nykypäivän sovelluskehitykseen ja niille on omat tarkoituksensa ja käyttökohteensa. Ongelmaksi kuitenkin tässä tutkimuksessa kuitenkin osoittautui ongelmat Liferayn nykyisen version kanssa. Angularin kanssa rakenteellisia vaatimuksia ei saatu täytettyä ollenkaan ja Reactin puolella ne täyttyivät juuri ja juuri. Ongelmaksi osoittautunee myös se, että Angular ja React ovat asiakaspään ohjelmistokehysiä siinä missä JavaServer Faces pyörii palvelimen päässä. Angularille ja Reactille tarvitaan omat palvelut tiedon välittämiseen Java-luokille, kun taas JavaServer Facesissa se onnistuu ilman.

Tämä osoittaa, että ainakaan tässä vaiheessa Liferayn kanssa Angular tai Reactikaan eivät ole sopivia korvaajia JavaServer Facesille ja PrimeFacesille.

6 Pohdinta

6.1 Tutkimuksen tavoite

Tutkimuksen tavoitteena oli vertailla nykypäiväisiä ohjelmistokehyksiä Angularia ja Reactia, sekä kartoittaa niiden soveltuvuutta toimeksiantajan vaatimuksiin JavaServer Facesin tilalle. Tutkimuksen vertailukohdaksi saatiin JavaServer Faces-sovellus, jota vastaavat käyttöliittymät luotiin. Tutkimus toteutettiin laadullisena tutkimuksena, jonka vertailukohtiin haastateltiin toimeksiantajalle työskenteleviä kehittäjiä. Tutkimuksen tuloksena tuotettiin esimerkkiä vastaavat käyttöliittymät, joihin lisättiin toiminnallisuudeksi tietokantaan datan tallentaminen sekä erillinen DevMode-elementti, jossa esiteltiin kehysten välistä toteutusta toimeksiantajalta saatuihin vaatimuksiin.

6.2 Tutkimuksen luotettavuus

Ohjelmistokehykset elävät ja muuttuvat jatkuvalla tahdilla. Se, mikä tänään on uutta ja hienoa, voi huomenna olla jo vanhentunutta. Tämä esiintyy erityisesti Javascript- ja Typescript-pohjaisissa ohjelmistokehyksissä, joissa suuretkin muutokset voivat olla arkipäivää. Omalta osaltaan React ja Angular ovat suhteellisen vakaita ohjelmistokehyksiä, joita käytetään laajasti mobiili- ja web-kehitykseenkin.

Tutkimuksen toteuttajan kokemus valituista ohjelmistokehyksistä on myöskin suhteellisen vähäinen, ja näin ollen syväluotaavaa vertailua ei pystytä täysin toteuttamaan. Myöskään ihan täyttä soveltuvuutta toimeksiantajan tarpeisiin ei pystyä evaluoimaan, sillä tutkimus rajattiin pelkän käyttöliittymän tekemiseen, sekä tutkimuksen aikana ilmenneitä ongelmia ei saatu ratkaistua.

6.3 Tutkimuksessa havaittuja hidasteita

Suurimpia hidasteita tutkimuksen aikana oli ohjelmistokehysten tuntemus tutkijalle. Tämä näkyi erityisesti riippuvuuksien hallinnassa, joiden kanssa tutkimuksen aikana oli useammankin kerran ongelmaa. Osa niistä varmasti tutkijan puolesta, mutta osa Liferayn asettamien muutosten ja dokumentaatioiden puutosten vuoksi.

Ongelmaa tutkimuksessa tuotti myöskin Liferayn npm bundler, joka jostain syystä ei halua toimia läheskään kaikkien npm-pakettien kanssa. Node-moduulit se onnistuu lataamaan, mutta niiden käyttöönottamisessa oli erittäin paljon ongelmia mm. node-moduulien kääntäminen OSGi-paketeiksi ei onnistunut useimmissa tapauksissa. Tästä syystä esimerkiksi koko Angular-osion rakenteelliset vaatimukset jäivät täyttämättä.

6.4 Kehitys ja jatkotutkimukset

Tutkimuksen rajauksen ja ajan vuoksi Liferayn NPM bundlerin uudempaa versiota ei saatu toimimaan. Tämä voisi mahdollistaa riippuvuuksien paremman toiminnan, jolloin rakenteellisia vaatimuksia voitaisiin kokeilla uudelleen. Mikäli myöhemmin tähän huomataan olevan tarvetta ja resursseja, voi tutkimuksen tekijä jatkaa vertailua laajemmassa mittakaavassa.

Lähteet

Angular ChangeLog. 2018. Tekninen dokumentaatio Angularin sivustolta. Viitattu 2.4.2018. https://Github.com/Angular/Angular/blob/master/CHANGELOG.md.

Apache Tomcat. N.d. Tekninen dokumentaatio Apache Tomcatin sivustolta. Viitattu 7.3.2018. http://tomcat.apache.org/.

Architecture Overview. 2018. Tekninen dokumentaatio Angularin sivustolta. Viitattu 25.4.2018. https://Angular.io/guide/architecture#architecture-overview.

BBVAOPEN4U. 2016. REST API: What is it, and what are its advantages in software development?. Viitattu 2.7.2018. https://bbvaopen4u.com/en/actualidad/rest-api-what-it-and-what-are-its-advantages-project-development.

Blade CLI. N.d. Tekninen dokumentaatio Liferayn sivustolta. Viitattu 17.3.2018. https://dev.liferay.com/develop/tutorials/-/knowledge_base/7-0/blade-cli.

Definition - what does build mean?. N.d. Tekninen kuvaus Technopedian sivustolla. Viitattu 17.3.2018. https://www.techopedia.com/definition/3759/build.

Docs. N.d. Tekninen dokumentaatio Reactin sivustolla. Viitattu 2.5.2018. https://Reactjs.org/docs/hello-world.HTML.

HTTP Methods. N.d. Tekninen dokumentaatio Restful apin sivustolla. Viitattu 2.7.2018. https://restfulapi.net/http-methods/.

Fetch API. N.d. Tekninen dokumentaatio Mozillan sivustolta. Viitattu 9.8.2018. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.

Glenn Flanagan 2018. React-collapsible. Viitattu 29.4.2018. https://Github.com/glennflanagan/React-collapsible.

Installing Apache Maven. N.d. Tekninen dokumentaatio Apache Mavenin sivustolta. Viitattu 10.7.2018. https://Maven.apache.org/install.HTML.

Installing Blade CLI. N.d. Tekninen dokumentaatio Liferayn sivustolta. Viitattu 17.3.2018. https://dev.liferay.com/develop/tutorials/-/knowledge_base/7-0/installing-blade-cli.

Installing Liferay Portal on Tomcat 8. N.d. Tekninen dokumentaatio Liferayn sivustolta. Viitattu 17.3.2018. https://dev.liferay.com/discover/deployment/-/knowledge_base/7-0/installing-liferay-on-tomcat-8.

Introducing Jsx. N.d. Tekninen dokumentaatio Reactin sivustolla. Viitattu 31.1.2018. https://Reactjs.org/docs/introducing-jsx.HTML.

Introduction to Liferay Development. 2017. Tekninen dokumentaatio Liferayn sivustolta. Viitattu 17.3.2018 https://dev.liferay.com/develop/tutorials.

Introduction to the Dependency Management. N.d.Tekninen dokumentaatio Apache Mavenin sivustolta. Viitattu 20.8.2018. https://Maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.HTML.

JavaServer Faces Technology User Interface. 2010. Tekninen dokumentaatio Oraclen sivustolta. Viitattu 11.4.2018. https://docs.oracle.com/Javaee/5/tutorial/doc/gentextid-10788.HTML.

Joaocarmo. 2018. React-smart-data-table. Viitattu 29.4.2018. https://Github.com/joaocarmo/React-smart-data-table.

JSF Architecture. N.d. Tekninen dokumentaatio Oraclen sivustolta. Viitattu 16.4.2018. http://www.oracle.com/technetwork/topics/index-090910.HTML.

Mapping with JPA. 2004. Tekninen dokumentaatio Jbossin sivustolta. Viitattu 2.7.2018. https://docs.jboss.org/hibernate/stable/annotations/reference/en/HTML/entity.HTML.

NPM Packages. 2018. Tekninen dokumentaatio Angularin sivustolta. Viitattu 25.4.2018. https://Angular.io/guide/npm-packages.

Paul Vorbach 2018. Download statistics for packages. Viitattu 18.4.2018. https://npm-stat.com/charts.HTML?package=React&package=Angular&package=%40Angular%2Fcore&from=2013-04-18&to=2018-04-18.

ReactDOM. N.d. Tekninen dokumentaatio Reactin sivustolla. Viitattu 18.3.2018. https://Reactjs.org/docs/React-dom.HTML.

React Changelog. 2018. Tekninen dokumentaatio Reactin Githubissa. Viitattu 2.4.2018. https://Github.com/facebook/React/blob/master/CHANGELOG.md.

React.Component. N.d. Tekninen dokumentaatio Reactin sivustolla. Viitattu 18.3.2018. https://Reactjs.org/docs/React-component.HTML.

Restful API N.d. tekninen dokumentaatio Restful apin sivustolla. Viitattu 2.7.2018. https://restfulapi.net/.

Software Development Life cycle. N.d. Artikkeli Tutorialspointin sivustolla. . Viitattu 21.3.2018. https://www.tutorialspoint.com/software_engineering/software_development_life_cycle.htm.

Template syntax. 2018. Tekninen dokumentaatio Angularin sivustolta. Viitattu 13.7.2018. https://Angular.io/guide/template-syntax.

The State of Javascript Frameworks,2017. 2018. Artikkeli NPMJS sivustolta. Viitattu 18.4.2018. https://www.npmjs.com/npm/state-of-Javascript-frameworks-2017-part-1.

What is Angular?. 2018. Tekninen dokumentaatio Angularin sivustolta. Viitattu 2.5.2018, https://Angular.io/docs.

Wojtekmaj. 2018. React-calendar. Viitattu 29.4.2018 https://Github.com/wojtekmaj/React-calendar.

Zarnekow, R. Brenner, W. 2005. Distribution of Costs over the Application Lifecycle – a multicase study. Viitattu 20.8.2018.

https://www.researchgate.net/publication/221408114_Distribution_of_Cost_over_the_Application_Lifecycle_-_a_Multi-case_Study.