Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Lauri-Kiril Tsereh

React-komponentin kehitys ja julkaisu NPM-moduuliksi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

25.09.2020

Tekijä Otsikko Sivumäärä Aika	Lauri-Kiril Tsereh React-komponentin kehitys ja julkaisu NPM-moduuliksi 33 sivua 25.09.2020
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Jorma Räty Kehityspäällikkö Antti Palomäki

Opinnäytetyön aiheena oli React-komponenttikirjaston kehitys ja julkaisu yksityiseen NPM-kirjaamoon. Komponentin julkaisu on toteutettu jatkuvana julkaisuna koontipalvelimelta, mikä edellyttää myös jatkuvan toimituksen sekä jatkuvan integraation ohjelmistokehitysmenetelmien käytön. Sekä React-komponentin kehitykseen että sen jatkuvaan julkaisuun tarvitaan montaa eri kirjastoa ja työkalua, joita kaikkia käsitellään tässä opinnäytetyössä ja kerrotaan niiden käytöstä projektissa tarkemmin.

Käytännön esimerkkinä käsiteltäville aiheille toimii Alma Medialle kehitetty komponenttikirjasto, jota käytetään Alma Median eri sivustoilla, kävijäliikenteen dynaamiseen ohjaamiseen Alma Median omistamien sivustojen välillä.

Avainsanat	Kirjaston julkaisu, jatkuva julkaisu, npm, React

Author Title Number of Pages Date	Lauri-Kiril Tsereh Development of React Component and Its Publication As NPM-Module 33 pages 25 September 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Jorma Räty, Senior Lecturer Antti Palomäki, Development Manager

The topic of the study was the development of React component library and its publication as a private package at NPM. The deployment of the component was implemented as continuous deployment, which also requires the usage of continuous delivery and continuous integration practices. Several libraries and tools are needed when developing a React component and implementing its continuous deployment, all discussed in the thesis.

A component library developed for Alma Media is used as a practical example. The component library was developed for use on different Alma Media web sites to route user traffic to other web sites owned by Alma Media.

Keywords	Publication of library, continuous deployment, npm, React

Sisällys

Lyhenteet

1 Johdanto 1

2 Selainpohjaisen käyttöliittymäkehityksen ohjelmointikielet 1

2.1 JavaScript 2
2.2 ECMAScript 2
2.3 TypeScript 4

3 Selainpohjaisen käyttöliittymäkehityksen kirjastot 5

3.1 React 5
3.1.1 JSX 6
3.1.2 Virtual DOM 7
3.1.3 React Fiber 7
3.2 JavaScript-testaus – Jest 8
3.3 Paketointi – Webpack 9
3.4 Kääntäminen – Babel 10

4 Paketinhallinta ja julkaisu 11

4.1 Paketinhallinta – NPM 11
4.2 Versionhallinta – GitHub 13
4.3 Jatkuva integraatio, toimitus ja julkaisu – Travis CI 14

5 Komponentin kehitys ja julkaisu 14

5.1 Komponentin vaatimukset 14
5.2 Komponentin toteutus 16
5.3 Paketin julkaisu 21
5.3.1 Koontipalvelimen määritys 21
5.3.2 Paketin koonti 24
5.3.3 Paketin testaus 27
5.3.4 Paketin julkaisu 29

6 Yhteenveto 31

Lyhenteet

DOM	Document Object Model. Tiedoston rakennemalli.
CD	Continuous Delivery. Jatkuva toimitus.
CD	Continuous Deployment. Jatkuva julkaisu.
CI	Continuous Integration. Jatkuva integraatio.
CTR	Click Trough Rate. Klikkisuhde.
ES	ECMAScript standardimäärittely.
JS	JavaScript ohjelmointikieli.
MVC	Model View Controller -ohjelmistoarkkitehtuuri.
NPM	Node Package Manager. Paketinhallintajärjestelmä.
SPA	Single Page Application. Web-sovellus malli, jossa sivusiirtymissä päivitetään vain muuttuva osa, perinteisten sivulatausten sijaan.
TS	TypeScript ohjelmointikieli.

1 Johdanto

Käyttöliittymän kehitys ReactJS-ohjelmistokehyksellä koostuu uudelleen käytettävien komponenttien kehityksestä. Jos on tarve käyttää samanlaista komponenttia useammassa kuin yhdessä koodikannassa, kannattaa komponentti julkaista komponenttikirjastona, jolloin samaa asiaa ja muutoksia ei tarvitse tehdä useampaan paikkaan, vaan riittää, että tarvittavat muutokset tehdään komponenttikirjastoon, jonka jälkeen se vain päivitetään sitä käyttävässä koodikannassa. Tässä insinöörityössä käydään läpi vastaavan React-komponenttikirjaston kehittämisen ja julkaisun eri vaiheita, sekä siihen tarvittavia teknologioita.

Insinöörityö tehtiin Alma Media Oyj:lle, digitaalisiin palveluihin ja kustannustoimintaan keskittyvälle suomalaiselle mediayhtiölle, kun syntyi tarve ohjata dynaamisesti kävijäliikennettä Alma Median omistamien sivustojen välillä. Alma Media koostuu monesta eri yksiköstä, ja eri sivustot voivat olla eri digikehitystiimien vastuulla, jotka voivat kuulua eri yksiköihin ja sijaita eri kaupungeissa, sekä luonnollisesti omistavat omat, toisistaan riippumattomat koodikannat ja ympäristöt. Kuitenkin Alma Median kaikki merkittävimmät sivustot on toteutettu React-kirjaston pohjalla. Tästä syystä komponentti päätettiin kehittää npm-rekisteriin julkaistavana React-komponenttikirjastona, ja se tuli mm. kävijäliikenteestä vastaavan eCom-tiimin vastuulle.

2 Selainpohjaisen käyttöliittymäkehityksen ohjelmointikielet

Tapa kehittää verkkosivustojen käyttöliittymiä on muuttunut viime vuosikymmenen loppupuoliskolla rajusti. Jos ennen laatimalla HTML oli hoidettu sivuston rakenne, CSS:llä tyylit ja JavaScriptillä tarvittavat muutokset edellä mainittuihin, nykyään on hyvin usein kaikki yhdistetty JavaScriptin hoidettavaksi erilaisten kehysten ja kirjastojen, kuten tässä insinöörityössä käsiteltävän hyvin suositun React-kirjaston avulla. Yksi syy tähän on kyseisen kehitysmenetelmän tuotoksena saatava yhden sivun web-sovellus, eli SPA (Single Page Application). Yhden sivun web-sovelluksen tärkeimpänä hyötynä perinteiseen verkkosivuarkkitehtuuriin on parantunut käyttökokemus, sulavamman siirtymän ansiosta eri näkymien välillä. Tämä saavutetaan sillä, että JavaScriptillä päivitetään ainoastaan muuttuva osa, jolloin myös verkon yli siirrettävän tiedon määrä

laskee, sekä näkymien väliset latausajat pienenevät. [1.] Yhden sivun web-sovelluksen voi kehittää puhtaallakin, niin sanotulla Vanilla JavaScriptillä, mutta tarjolla olevat sovelluskehykset ja kirjastot selkeyttävät kehitysprosessia sekä ratkovat monia ongelmia, kuten esimerkiksi käyttöliittymän pitämistä synkronoituna sovelluksen sisäiseen tilaan. [2.]

2.1 JavaScript

JavaScript, tunnettu myös lyhenteellä JS, on kaikista yleisimmin käytetyistä verkkoselaimista löytyvä ohjelmointikieli, jota käytetään useimmiten verkkoselainpohjaisten käyttöliittymien ohjelmointiin. Viime aikoina JavaScript on yleistynyt myös palvelinten toteutuksissa, usein Node.js-ajoympäristöä käyttäen. Myös natiivisovellusten kehityksessä on alettu usein käyttämään JavaScriptiä. Esimerkiksi sekä iOS- että Android-käyttöjärjestelmiin tarkoitettuja sovelluksia saa kätevästi kehitettyä React Native -ohjelmistokehystä käyttäen.

JavaScript on korkean tason ohjelmointikieli. Se soveltuu olio-ohjelmointiin, funktionaaliseen ja imperatiiviseen ohjelmointiin sekä skriptaukseen.

2.2 ECMAScript

Loppukäyttäjän selaimessa JavaScript-koodia suorittaa JavaScript-moottori. Jokaisella merkittävällä selaimella on olemassa oma JavaScript-moottori, jonka kehitystä ylläpitää yleensä selaimen valmistaja.

Taulukko 1. Yleisimmät JavaScript-moottorit ja niiden sovellukset.

JavaScript moottori	**Sovellukset**
Chakra	Microsoft Explorer
Nitro	WebKit, Safari, Qt 5
SpiderMonkey	Firefox, Adobe Acrobat
V8	Google Chrome, Node.js, Opera, Microsoft Edge, MarkLogic

Koska näiden moottoreiden kehityksestä vastaavat eri yhtiöt, tarvitsee JavaScript-kieli oman standardimäärittelyn, jotta kehittäjät voisivat kehittää JavaScript-ohjelmia, jotka toimisivat samalla tavalla kaikilla moottoreilla, ja sitä kautta selaimilla. Ecma International -standardointiorganisaatio ylläpitää vuonna 1997 luomaansa ECMA-262 -standardia, eli ECMAScript -määrittelyä, jota kaikki edellisessä taulukossa mainitut JavaScript-moottorit noudattavat.

Koska uusien selainversioiden mukana, teknologian kehityksen ja kehittäjien tarpeiden mukaan JavaScriptiin tulee uusia toiminnallisuuksia, ECMAScriptistä on olemassa eri versioita. Vuonna 2015 Ecma International päätti siirtyä ECMAScript:n vuosittaiseen julkaisuun, ja samalla version nimitys muuttui järjestysnumeromallisesta (ES6) vuosilukumalliseen (ES2015).

Taulukko 2. ECMAScript -versiot ja niiden tuki eri selaimissa.

ECMAScript versio	**Julkaistu**	**Tuki selaimissa**
ECMAScript 5 (ES5)	Joulukuu 2009	Kaikki yleisimmät selaimet
ECMAScript 2015 (ES 2015) Usein käytetään nimitystä ES6	Kesäkuu 2015	Kaikki ajantasaiset selaimet. IE ei tue.
ECMAScript 2016 (ES 2016)	Kesäkuu 2016	Googlen V8 moottorilla toimivat selaimet (Chrome, Edge, Opera).
ECMAScript 2017 (ES 2017)	Kesäkuu 2017	Googlen V8 moottorilla toimivat selaimet (Chrome, Edge, Opera).
ECMAScript 2018 (ES 2018)	Kesäkuu 2018	Googlen V8 moottorilla toimivat selaimet (Chrome, Edge, Opera).
ECMAScript 2019 (ES 2019)	Kesäkuu 2019	Googlen V8 moottorilla toimivat selaimet (Chrome, Edge, Opera).

2.3 TypeScript

JavaScriptin heikko tyypitys helpottaa sen oppimista, mutta kääntöpuolena hankaloittaa laajempien ohjelmien ylläpitoa. Tähän ratkaisuksi Microsoft ylläpitää Microsoftin kehittämää avoimen lähdekoodin ohjelmointikieltä, joka on TypeScript tai lyhyesti TS. TypeScript on luokilla ja staattisella tyypityksellä jatkokehitetty JavaScript. TypeScript:in käyttö nopeuttaa laajojen verkkosovellusten kehitystä, tekee koodista luettavampaa, helpommin refaktoroitavaa sekä helpottaa virheiden paikannusta, ja lopputuloksena parantaa ohjelman laatua. TypeScript-ohjelma käännetään JavaScriptiksi esimerkiksi Babel-kirjastoa tai TypeScriptin omaa kääntäjää käyttäen, jonka jälkeen se toimii kaikilla selaimilla.

```
function tervehdi(henkilo) {
     return "Hei, " + henkilo;
}

let kaveri = "Mikko Mallikas";

document.body.textContent = tervehdi(kaveri);
```

Esimerkkikoodi 1. JavaScript-esimerkki.

```
function tervehdi(henkilo: string) {
     return "Hei, " + henkilo;
}

let kaveri: string;
kaveri = "Mikko Mallikas";

document.body.textContent = tervehdi(kaveri);
```

Esimerkkikoodi 2. Edellinen esimerkkikoodi TypeScriptinä.

Kun vertaillaan edellistä TypeScript-esimerkkikoodia idealtaan identtiseen, ensimmäisessä esimerkkikoodissa esitettyyn ohjelmaan, huomataan, että TypeScriptin tapauksessa ”henkilo”-parametri ja ”kaveri”-muuttuja ovat tyypitettyinä string-tyyppisiksi. Oikein asetetussa ohjelmointiympäristössä staattinen analyysi antaa virheilmoituksen, jos jompaankumpaan koitetaan sitoa jonkin muun tyyppistä arvoa kuin string.

3 Selainpohjaisen käyttöliittymäkehityksen kirjastot

Nykyään puhtaalla JavaScriptillä toteutetaan yleensä jotkin pienet skriptit joihinkin erikoisempiin tarkoituksiin. Kun lähdetään kehittämään laajempaa käyttöliittymää, yleensä käytetään apuna kirjastoja ja ohjelmistokehyksiä. Työn kirjoittamishetkellä yleisimmin käytettyjä ovat Facebookin kehittämä React-kirjasto, Googlen kehittämä Angular-ohjelmistokehys, sekä Vue.js -ohjelmistokehys. Näiden keskeisten käyttöliittymäkehitys ohjelmistokehysten lisäksi on olemassa paljon muita kehitystä helpottavia työkaluja, kuten esimerkiksi testausautomaatio-ohjelmistokehys Jest tai JavaScript-kääntäjä-kokooja Babel.

3.1 React

React (tunnettu myös React.js tai ReactJS) on vuonna 2013 Facebookin julkaisema avoimen lähdekoodin JavaScript-kirjasto selainpohjaisten käyttöliittymien kehitykseen. React nopeuttaa ja selkeyttää laajojen käyttöliittymäprojektien kehitystä sekä tuo skaalautuvuutta.

Reactia verrataan usein Angular-ohjelmistokehykseen, mutta toisin kuin Angular, React on vain kirjasto. Tämä ero tulee siitä, että Angular hoitaa MVC-arkkitehtuurin kaikki kolme osuutta, kun React hoitaa vain View- eli näkymäosuuden. Tämän lisäksi Angulariin sisältyy muutenkin paljon toiminnollisuuksia vakiona, mutta Reactin kohdalla kehittäjälle jää vapaus valita erilaiset kirjastot käyttöön projektin tarpeiden mukaan. React-projekteissa suosittuja kirjastoja ovat esimerkiksi React-router reititykseen, Redux sisäiseen tilanhallintaan tai Jest-testaukseen. Kääntöpuolena tälle vapaudelle on lisääntyneen konfiguroinnin lisäksi vastuu riippuvuuksien pitämisestä ajan tasalla. [3.]

Käyttöliittymän kehittäminen Reactilla on komponenttipohjaista. Tämä tarkoittaa sitä, että React-sovelluksen kehittäminen tapahtuu kehittämällä React-komponentteja ja yhdistelemällä niitä. Nämä komponentit ovat itsenäisiä ohjelman lohkoja, jotka palauttavat html-elementeistä ja/tai muista React-komponenteista koostuvan käyttöliittymän osan (esim. nappi, kirjautumislomake, etusivu). React-komponentit voivat sisältää logiikan, oman tilan ja voivat ottaa parametreja vastaan isäntäkomponentilta.

React-komponenttien kuuluu olla helposti uudelleenkäytettävissä esimerkiksi toisessa React-komponentissa, tai jopa toisessa React-sovelluksessa.

Verkkosovellusten lisäksi Reactilla voi kehittää myös natiivisovelluksia Androidille, iOS:lle sekä Windows 10:lle React Native -kirjastoa käyttäen.

3.1.1 JSX

Perinteisesti verkkoselainpohjaiset käyttöliittymät koostuvat erillisestä logiikasta JavaScriptin muodossa, rakenteesta HTML:n muodossa ja tyylimäärittelystä CSS:n muodossa. Toisin kuin on totuttu näkemään missään muussa laajasti käytössä olevassa kirjastossa, Reactissa on logiikka ja rakenne yhdistetty JSX-syntaksilaajennoksen alle. JSX-nimi tuleekin lyhenteestä ”JavaScript XML”, tai TypeScriptiä käytettäessä TSX-lyhenteestä ”TypeScript XML”. Avoimessa käytössä on kirjastoja, joilla saa myös tyylimäärittelyn hoidettua samassa tiedostossa. Kyseisellä lähestymistavalla on mukava kehittää laajojakin sovelluksia, muun muassa paremmin toimivan staattisen analyysin ansiosta.

Vaikka JSX:n käyttö on suositeltua React-sovelluksia kehitettäessä, se ei ole pakollista. JSX:n poisjättäminen voi olla paikallaan siinä tapauksessa, jos rakennusympäristössä ei ole kokoamista. [4.]

```
const otsikko = “JSX on jees!”;
const Element = <h1>{otsikko}</h1>;

class EsimKomponentti extends React.Component {
      render () {
            return (
                  <div>
                        <Element />
                  </div>
            );
      };
}

ReactDOM.render(<EsimKomponentti />, document.getElementById(“main”));
```

Esimerkkikoodi 3. JSX-syntaksi. ”Element”-muuttujan arvo on hyvä esimerkki JSX:stä. Se ei ole tekstijono, eikä se ole HTML-elementti, vaikka se siltä näyttääkin.

3.1.2 Virtual DOM

React on tunnettu sen erityisen nopeasta toiminnasta, ja ehdottomasti suurin syy Reactin nopealle toiminnalle on Reactin tapa manipuloida DOM:a. DOM:in manipulointi tarkoittaa sitä, että jotakin sivulla näytettävää objektia muutetaan, ja tämä muutos on yleisesti JavaScriptin toiminnassa kaikkein raskainta. Muissa yleisissä selainpohjaisten käyttöliittymien ohjelmistokehysksissä yhden DOM-objektin muututtua päivitetään koko DOM, joka on oletettavasti hyvin raskasta, varsinkin moderneissa käyttöliittymissä, joissa näitä muutoksia tapahtuu paljon. Reactissa tämä ongelma on ratkottu sillä, että muutos tehdään ensiksi Virtual DOM:iin, eli varsinaisen DOM:n kevyeeseen kopioon. Virtual DOM:iin muutosten tekeminen on huomattavasti kevyempää, sillä Virtual DOM:lla ei ole graafista esitystä. Ennen muutoksia Virtual DOM:sta otetaan ”kuva”, johon muutosten jälkeistä Virtual DOM:a vertaamalla eli ”diffaamalla” selvitetään muuttuneet DOM-objektit, jonka jälkeen varsinaiseen DOM:iin päivitetään vain nämä muuttuneet DOM-objektit.

3.1.3 React Fiber

26. syyskuuta 2017 julkaistiin React 16.0, joka sisälsi muun muassa suuren parannuksen React-sovellusten toiminnan sulavuuteen, nimeltään React Fiber. Aiemmissa Reactin versioissa Reactin kaikki operaatiot lisättiin peräkkäin JavaScriptin kutsupinoon. Jos jokin operaatio kesti pidempään kuin 16,67 ms, joka on yleinen kuvan päivitysväli suurimmassa osassa näytöistä, aiheutti tämä kuvan pätkimistä, sillä yksisäkeisessä JavaScriptissä käyttöliittymän päivitys hoidetaan samassa pääsäkeessä. Tähän ratkaisuksi tuli React Fiber, joka on nimitys Reactin sisäisten algoritmien läpikäynnin kokonaisuudelle, jonka tuloksena React voi

- priorisoida eri tyyppiset operaatiot
- keskeyttää operaation suorittamisen, ja palata takaisin sen suorittamiseen, kun suuremman prioriteetin operaatiot on suoritettu (esimerkiksi kuvan päivitys)
- hylätä operaation suorittamisen, jos tarve sille katoaa
- käyttää uudelleen aikaisemmin suorittamiaan operaatioita.

Sen lisäksi, että nämä muutokset tekivät käyttöliittymän toiminnasta sulavampaa, sivutuotoksena Reactiin saatiin tapa käsitellä virheitä, nimeltään ”Error Boundary”, jonka

avulla voidaan esittää luodun varanäkymän virheen tapahtuessa sen sijaan, että näytettäisiin rikkinäistä komponenttipuuta. [5.]

3.2 JavaScript-testaus – Jest

Yksikkötestien tehtävänä on testata automaattisesti ohjelmiston osia, eli yksikköjä. Testattavana voivat olla esimerkiksi yksittäiset funktiot tai luokat. Yksikkötesteillä saadaan ohjelmistosta laadukkaampaa, sillä esimerkiksi julkaisuputkea käytettäessä testit ajetaan koonnin yhteydessä automaattisesti, ja jonkin osan toimiessa väärin julkaisu peruuntuu, ja saadaan ilmoitus virheellisesti toimivasta yksiköstä. Yksikkötestien lisäksi on olemassa muitakin tapoja automaattisesti testata ohjelmistoa.

Jest on Facebookin ylläpitämä ohjelmistokehys JavaScriptin (myös TypeScriptin) ja erityisesti React-sovellusten yksikkötestaamiseen. Koska myös React on Facebookin kehittämä, Jest toimii erittäin hyvin Reactin kanssa, ja tästä syystä onkin yleisin tapa testata React-sovelluksia. Jestin käyttöönotto on hyvin helppoa ja normaalitilanteissa onnistuu ilman mitään konfigurointia. Jest sisältää tehokkaan rajapinnan väitteiden testaamiseen, jolla voidaan testata JavaScript-funktioista ja React-komponenteista saatuja arvoja oletuksia vastaan.

```
function summa(a, b) {
	return a + b;
}
module.exports = summa;
```

Esimerkkikoodi 4. sum.js -tiedostoon luotu esimerkkifunktio, joka palauttaa parametrien summan.

```
const summa = require(”./summa.js”);
test(“summa 1 + 2 on yhtä kuin 3”, () => {
	expect(summa(1, 2)).toBe(3);
}
```

Esimerkkikoodi 5. sum.test.js -tiedostoon luotu testi, esimerkkikoodi 4:ssä luodun funktion testaamiseen. Käytetään Jestin toBe-sovitinta, jonka läpäisemiseksi toBe-metodin parametrin on täsmättävä expect-funktion parametrin kanssa.

Testejä voidaan ajaa kutsumalla Jestin omaa testiajajaa komentokehotteesta, joka ajaa kaikki ”__test__” -hakemistossa olevat JavaScript- tai TypeScript- tiedostot, tai tiedostot, joiden nimen tiedostopäätteen edestä löytyy ”test”-etuliite.

```
PASS summa.test.js
✓ summa 1 + 2 on yhtä kuin 3 (5ms)
```

Esimerkkikoodi 6. Esimerkkikoodi 5:ssä luodun testin läpäistessä Jest tulostaa komentokehoitteeseen seuraavanlaisen viestin.

3.3 Paketointi – Webpack

Alun perin JavaScriptia luotiin tuomaan verkkosivuille toiminnallisuutta lyhyiden skriptien avulla, eikä se tukenut modulaarisuutta lainkaan. Nykyisin JavaScriptillä rakennettavat laajat ohjelmistot tarvitsevat modulaarisuutta, jotta koodia saataisiin uudelleenkäytettäväksi projektien sisällä sekä kirjastoina. Vuonna 2015 julkaistussa ECMAScript 6:ssa määriteltiin import- ja export-lausekkeet, jotka toivat modulaarisuuden JavaScriptiin, josta meni kuitenkin vielä aikaa, ennen kuin yleisimmät selaimet toteuttivat tämän, eikä esimerkiksi Internet Explorerin mikään versio tue ECMAScript 6:tta lainkaan työn kirjoittamishetkenä.

Vuonna 2012 julkaistiin ensimmäinen versio avoimen lähdekoodin kirjastosta nimeltään Webpack, joka ratkoo edellä mainitun ongelman paketoimalla kaikki moduulit yhteen JavaScript-tiedostoon, selaimilla käytettäväksi. Webpackin Code Splitting -ominaisuudella ohjelman voi myös paketoida yhden ison tiedoston sijaan useampaan pienempään, jolloin sivuston ensimmäisellä sivulatauksella ladataan vain välttämättömät asiat, minkä seurauksena on vähemmän ladattavaa, ja sivusto latautuu nopeammin. Paketoinnin ohella Webpack voi minifioida koodia, eli pienentää paketoinnin tuloksena saatua JavaScript-tiedostoa poistamalla siitä ohjelman toiminnan kannalta tarpeettomat asiat, kuten kommentit, tarpeettomat välilyönnit, rivinvaihdot ja käyttämättömät koodin osat, sekä nimeämällä muuttujat ja funktiot lyhyillä generoiduilla merkkijonoilla.

```
function tervehdi(e){return"Hei, "+e}let kaveri="Mikko
Mallikas";document.body.textContent=tervehdi(kaveri);
```

Esimerkkikoodi 7. Esimerkkikoodi 1 minifioituna.

JavaScriptin lisäksi Webpack voi käsitellä myös esimerkiksi tyylitiedostoja.

Dev Server on vielä yksi tärkeä Webpackin ominaisuus, joka nopeuttaa sovellusten kehittämistä päivittämällä selainta aina sen mukaan, kun projektin tiedostoihin tallennetaan uusia muutoksia.

Lisäosien avulla Webpack voidaan konfiguroida käsittelemään projektin tiedostoja monella muullakin tavalla, esimerkiksi kääntämään koodia kieleltä tai standardilta toiselle.

3.4 Kääntäminen – Babel

Babel on avoimen lähdekoodin JavaScript-kääntäjä, jota yleensä käytetään ECMAScriptin uudempien versioiden kääntämiseen ECMAScript 5:een. Tällä saavutetaan se, että kehittäjät voivat kirjoittaa vanhemmillakin selaimilla toimivia ohjelmia käyttäen uusien ECMAScript-versioiden ominaisuuksia sekä syntaksia.

```
[1, 2, 3].map((n) => n + 1)
```

Esimerkkikoodi 8. ECMAScript 2015 -versiossa määritelty nuolifunktio.

```
[1, 2, 3].map(function(n) {
     return n + 1;
}
```

Esimerkkikoodi 9. Babelin tuottama ECMAScript 5 määrittelyn mukainen funktio, 8. esimerkkikoodin funktiosta.

Reactilla kehittäessä Babel on hyödyllinen JSX- tai TypeScriptillä ohjelmoitaessa TSX-syntaksilaajennoksen kääntämiseen normaaliin, selaimella ajettavaan JavaScriptiin, babel-preset-lisäosan avulla.

```
const otsikko = "JSX on jees!";
const Element = /*#__PURE__*/React.createElement("h1", null, otsikko);

class EsimKomponentti extends React.Component {
     render() {
          return /*#__PURE__*/React.createElement("div", null,
/*#__PURE__*/React.createElement(Element, null));
     }
}

ReactDOM.render(/*#__PURE__*/React.createElement(EsimKomponentti, null),
document.getElementById('main'));
```

Esimerkkikoodi 10. Kolmannen esimerkkikoodin JSX-koodi käännettynä JavaScriptiksi Babelia käyttäen.

4 Paketinhallinta ja julkaisu

Kuten kolmannessa luvussa mainittiin, React-sovelluksen kehitys koostuu uudelleenkäytettävistä komponenteista. Yleensä suurin osa komponenteista kehitetään projektikohtaisesti, mutta osa komponenteista voi olla uudelleenkäytettävissä useammassa eri projektissa. Oletettavasti komponenttien manuaalinen kopioiminen projekteihin ei ole vaihtoehto, sillä kun komponenttiin tulisi muutoksia, olisi käsin päivittäminen jokaiseen projektiin erikseen hyvin työlästä ja virhealtista. Komponentit voidaankin paketoida komponenttikirjastoon ja julkaista pakettirekisteriin, josta niiden hallinta projektissa, eli asennus, päivitys, konfigurointi sekä poistaminen on helppoa ja automatisoitua.

4.1 Paketinhallinta – NPM

NPM eli Node Package Manager on JavaScript-paketeille tarkoitettu paketinhallintajärjestelmä. Nimi tulee siitä, että alun perin NPM luotiin käytettäväksi palvelinten kehitykseen tarkoitetun Node.js -ajoympäristön kanssa, jonka oletus paketinhallintajärjestelmä se on. Tästä huolimatta NPM:n käyttö on hyvin yleistä myös kehittäessä käyttöliittymiä JavaScriptillä.

NPM-paketti on kansio, joka sisältää JavaScript-ohjelman sekä ohjelman kuvauksen package.json -tiedostossa. [6.] package.json -tiedosto sisältää tietoja paketista, paketin riippuvuuksista, eli muista paketin käyttämistä paketeista sekä muuta NPM:ään liittyvää konfiguraatiota.

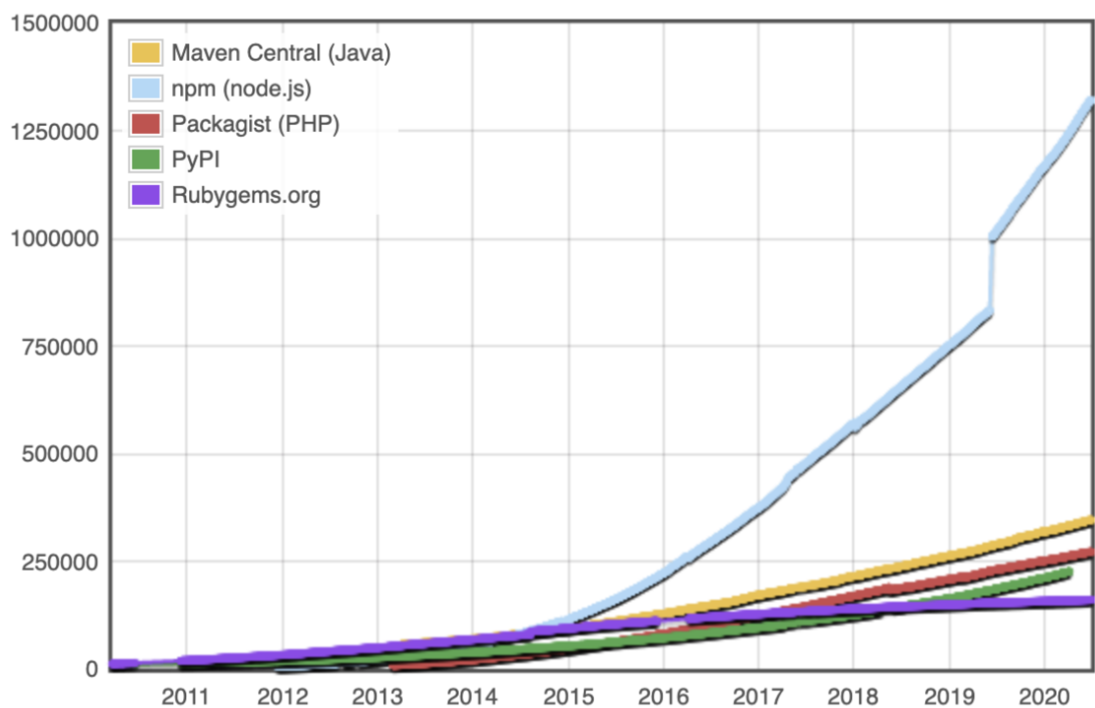
Avoimen lähdekoodin pakettien julkaisu NPM:ään sekä asentaminen NPM:stä on ilmaista, eikä jälkimmäinen vaadi edes tunnistautumista. Yksityisten pakettien hallinta on mahdollista maksullisilla tunnuksilla.

NPM koostuu paketteja sisältävästä npm-rekisteristä, verkkosivuista pakettien mukavaan hakemiseen ja selaamiseen sekä npm-nimisestä komentorivityökalusta, jolla pakettien hallinta onnistuu.

```
npm install <paketin nimi>
```

Esimerkkikoodi 11. NPM:n eniten käytetty komento, pakettien asentamiseen laitteeseen, tai projektikohtaisesti.

NPM on maailman suurin ohjelmistokirjaamo sekä paketinhallintajärjestelmä. NPM:ssä on yli 1,3 miljoonaa avoimen lähdekoodin pakettia.



Kuva 1. Avoimessa käytössä olevien pakettien määrien vertailu suurimpien paketinhallintajärjestelmien välillä. [7.]

JavaScript-paketinhallintaan on olemassa paljon muitakin työkaluja, esimerkiksi NPM:n jälkeen suurin, Facebookin kehittämä yarn. Yarn ratkoi monia NPM:stä löytyviä teknisiä ongelmia, esimerkiksi niitä, jotka liittyvät suorituskykyyn. Yleensä vaihtoehtoiset paketinhallintajärjestelmät käyttävät kuitenkin NPM:n rekisteriä pakettien säilömiseen. Viime vuosina NPM:ssä on kuitenkin korjattu kaikki yleisimmät puutteet vaihtoehtoisiin paketinhallintajärjestelmiin verrattuna, eikä nykyään ole syytä käyttää muuta

paketinhallintajärjestelmää kuin NPM:ää, lukuun ottamatta joitain spesifejä käyttötapauksia.

NPM:ssä käytetään semanttista versionumerointia. Semanttinen versionumero merkataan kolmella numerolla (esim. 1.2.3), joista ensimmäisellä numerolla merkataan merkittävintä versiota, eli julkaisua, joka sisältää rikkovia muutoksia. Paketin päivittäminen merkittävällä versiolla voi aiheuttaa ohjelmassa ongelmia ja vaatia suurempaa työtä. Toisella numerolla merkataan vähäistä versiota, eli julkaisua, joka sisältää uusia toimintoja, mutta on taaksepäin yhteensopiva. Viimeisellä numerolla merkataan paikkaavaa versiota, eli taaksepäin yhteensopivia korjauksia sisältävää julkaisua. package.json -tiedostoon voidaan merkata riippuvuudelle sallitut päivitykset merkitsemällä riippuvuuden versionumeron tietyllä tavalla:

- 1.0, 1.0.x tai ~1.0.4 – päivittäminen sallittu vain paikkaavalla versiolla.
- 1, 1.x tai ^1.0.4 – päivittäminen sallittu myös vähäisellä versiolla.
- * tai x – päivitetään aina uusimpaan tarjolla olevaan versioon.

4.2 Versionhallinta – GitHub

Versionhallinnan avulla tiedostojen muutoksia voi versioida ja tarvittaessa palata tiettyyn versioon, esimerkiksi jos halutaan peruuttaa jotkin ohjelman rikkovat muutokset. Tämän lisäksi hajautettu ja keskitetty versionhallinta helpottaa projektin kehitystä useamman kehittäjän voimin, mikä vähentää ristiriitaisia tilanteita sillä, että tiedostot, joilla kehittäjät työskentelevät, eroavat toisistaan vain kehittäjien työn alla olevilla muutoksilla. Lisäksi joillain versionhallintajärjestelmillä voi luoda erillisiä haaroja, eli kopioita viimeisimmästä versiosta, joissa voidaan kehittää tiettyä ominaisuutta, mikä yhdistää sen valmistuttua päähaaraan. Ennen muutosten yhdistämistä päähaaraan muutokset voidaan arvioida vertaiskatselmoinnissa, joka parantaa ohjelman laatua sekä kehittäjien omaa ja yhteistä osaamista.

GitHub on hajautettu ja maailman suosituin versionhallintajärjestelmä. Hajautettu versionhallinta eroaa keskitetystä sillä, että koko versiohistoria kopioidaan kehittäjien laitteille sen sijaan, että sitä pidettäisiin yhdellä palvelimella, jonka ansiosta ratkotaan ongelma, jossa palvelimella sattuva virhetilanne keskeyttää kehittäjien työskentelyn, tai jopa hävittää tiedot versiohistoriasta. [8, s. 5.]

4.3 Jatkuva integraatio, toimitus ja julkaisu – Travis CI

Jatkuvalla integraatiolla (engl. Continuous Integration, CI) tarkoitetaan ohjelmistokehitysmenetelmää, jossa versionhallinnan päähaaraan integroidaan jatkuvasti muutoksia pienissä erissä. Tällä saavutetaan se, että projektin kehittäjien käsillä olevat versiot ohjelmasta eivät eroa toisistaan paljoa. Toinen tärkeä jatkuvan integraation tuoma hyöty on se, että virheet huomataan nopeasti ja niiden paikantaminen pienistä muutoksista on paljon helpompaa. Tyypillisesti automaattinen kokoaminen sekä testaus ovat osa jatkuvaa integraatiota. Kun versionhallintaan viedään muutoksia, ohjelma kasataan automaattisesti, tarvittaessa erilaisiin ympäristöihin tai eri ohjelmointikielien versioita käyttäen, minkä jälkeen sillä ajetaan automaattiset testit.

Usein jatkuvaa integraatiota jatketaan jatkuvalla toimituksella (engl. Continuous Delivery, CD), eli menetelmällä, jossa automaattiset testit läpäisevää ja versionhallinnan päähaarassa olevaa ohjelmaa voidaan semmoisenaan julkaista tuotantoon. Julkaisemisen on myös oltava vaivatonta, eli julkaisuun liittyvän konfiguroinnin on löydyttävä versionhallinnasta.

Joskus jatkuvaa integraatiota ja toimitusta jatketaan vielä jatkuvalla julkaisulla (engl. Continuous Deployment, CD). Jatkuvan julkaisun menetelmässä edelliset vaiheet läpäisevä ohjelma myös julkaistaan automaattisesti.

Näiden prosessien automatisointiin on olemassa monia palveluita, joista yksi on Travis CI. Travis CI automatisoi GitHubiin tai BitBucketiin vietävän ohjelman koonnin, testauksen sekä julkaisun eri vaiheita. Travis CI on saatavilla omalla palvelimella itse ylläpidettävänä ohjelmistona sekä pilvipalveluna, joista jälkimmäinen on suotuisampi.

5 Komponentin kehitys ja julkaisu

5.1 Komponentin vaatimukset

Tavoitteena oli kehittää React-komponentti, jota olisi mahdollisimman helppo ottaa käyttöön Alma Median sivustoilla, esimerkiksi jonkin mediasivuston etusivulla

käytettäväksi. Komponentin tehtävänä on ohjata kävijäliikennettä Alma Median sivustoilta toisille Alma-median sivustoille, tiettyä jakelulogiikkaa käyttäen. Kävijäliikennettä ohjataan näyttämällä käyttäjälle sivuston sisältöä, jolle käyttäjää halutaan ohjata, esimerkiksi 2. kuvassa on esitetty ohjaus Talouselämän sivuille Iltalehden sivuilta. Kehitettävä komponentti sisältää vain jakelulogiikan. Esitettävät sisältökomponentit asennetaan jakelukomponenttiin erillisinä paketteina Alma Median yksityisestä rekisteristä, sillä osaa niistä käytetään muilla Alma Median sivustoilla semmoisinaan.

Kuva 2. Talouselämän sisältökomponentti esitettynä Iltalehden etusivulla sijaitsevassa jakelukomponentissa.

Jotta komponentin toiminnan tehokkuutta voitaisiin analysoida ja jakelulogiikkaa säätää tehokkaammaksi, komponentin on myös lähetettävä analytiikkatietoa Google Analyticsiin, kun käyttäjä näkee komponentin ja kun käyttäjä klikkaa jotain sen osiota. Näillä tiedoilla saadaan laskettua komponenttien klikkisuhteen (CTR), joka on hyvä mittari komponentin tehokkuudelle.

Komponentilla on myös oltava mahdollista testata erilaisia jakelulogiikoita, sekä tehdä väliaikaisia muutoksia jakelulogiikkaan ilman, että jakelulogiikkaa pitäisi päivittää komponenttiin, joka vaatisi komponentin päivityksen sivustojen lähdekoodiin sekä sivustojen viennin tuotantoon. Väliaikaiset muutokset sekä testaukset jakelulogiikkaan tehdään Google Optimize -työkalulla, jolla voidaan ajaa tarvittavia JavaScript-komentoja erilaisin ehdoin. Google Optimize on asennettuna sivuston asiakaspuoleen, jonka seurauksena sillä on pääsy vain asiakaspuolella oleviin tietoihin. Tästä syystä komponentti on sidottava globaaliin muuttujaan, ja sillä on oltava funktiot näytettävän sisällön vaihtamiseen, sekä näytettävän sisällön selvittämiseen.

5.2 Komponentin toteutus

Komponentti kehitettiin TypeScript-ohjelmointikielellä React-kirjastoa käyttäen. Opinnäytetyön kirjoittamishetkellä komponenttia käytettiin Iltalehden ja Uuden Suomen sivuilla hieman eroavilla jakelulogiikoilla ja sisällöillä. Molemmille kehitettiin omat komponentit samaan moduuliin, jolla saatiin julkaisuun liittyvät konfiguraatiot ja komponenttien yhteiset koodit tehtyä kerralla yhteen paikkaan, josta molemmat komponentit voivat niitä käyttää. Lisäksi hyötynä on se, että kun tulevaisuudessa tullaan tarvitsemaan jakelukomponentteja uusilla jakelulogiikoilla ja sisällöillä, on tämmöisten komponenttien luominen mahdollisimman helppoa, sillä se ei vaadi muuta kuin jakelulogiikan ja sisällön määrittävän komponentin luontia. Kaikki yleiset asiat saadaan uudelleenkäytettyä. Molemmat kehittämämme jakelukomponentit käyttävät esimerkiksi samaa käärekomponenttia, joka sisältää analytiikkatapahtumien lähettämisen sekä komponentin kiinnittymisestä ilmoittavan JavaScript-tapahtuman lähettämisen.

Käärekomponentti on Reactin vaihtoehto Decorator pattern -mallille. Yleensä React-kehityksessä import-lausekkeella tuodut komponentit sijoitetaan renderöitävän komponentin sisälle, kuten 12. esimerkkikoodissa. Käärekomponentin tapauksessa import-lausekkeella tuotu käärekomponentti sijoitetaan renderöitävän komponentin ympärille, kuten 13. esimerkkikoodissa, jolla renderöitävään komponenttiin saadaan lisättyä käärekomponentissa määriteltyä toiminnallisuutta tai tyylejä.

```
return (
	<div>
		<TuotuTavallinenKomponentti />
	</div>
)
```

Esimerkkikoodi 12. import lausekkeella tuodun komponentin käyttö normaaliin tapaan renderöitävän komponentin sisällä.

```
return (
     <EcomEmbed locationMedia={this.props.locationMedia}
locationPlace={this.props.locationPlace} publication={this.state.publication}
gaId={this.props.gaId} contentCategory={contentCategory}>
           {SelectedComponent}
     </EcomEmbed>
)
```

Esimerkkikoodi 13. import lausekkeella tuodun EcomEmbed-käärekomponentin käyttö Uuden Suomen jakelukomponentissa. Tällä saadaan lisättyä Uuden Suomen komponenttiin EcomEmbed-käärekomponentissa määriteltyä toiminnallisuutta.

Käärekomponentti saa children-argumenttina siihen käärityn komponentin, jota käyttäen kääritty komponentti sijoitetaan oikeaan kohtaan käärekomponentin sisällä, kuten voidaan nähdä kehittämämme käärekomponentista palautettavassa datassa, 14. esimerkkikoodin lopussa.

```
import * as React from 'react'
import { isElementInViewPort } from './utils'

declare global {
    interface Window { ecomEmbed: React.ReactNode }
}

export type Props = {
    gaId: string
    locationMedia: string
    locationPlace: string
    publication: string
    contentCategory: string
}

const EcomEmbed : React.FunctionComponent<Props> = props => {
    const [element, setElement] = React.useState<HTMLDivElement>()

    React.useEffect(() => {
        //Emit event that notifies that elements methods are accessible, for
example if lazyload is used
        if(typeof(Event) === 'function') {
            var renderedEvent = new Event('ecomJakeluRendered')
        } else {
            var renderedEvent = document.createEvent('Event')
            renderedEvent.initEvent('ecomJakeluRendered', false, false)
        }
        document.dispatchEvent(renderedEvent)
    }, [])

    React.useEffect(() => {
        let eventSent = false
        function gaEventsOnScroll() {
```

```
            if(element && !eventSent && isElementInViewPort(element)) {
                window.removeEventListener('scroll', gaEventsOnScroll)
                const eventCategory = 'internalbox-' + props.locationMedia +
'-' + props.locationPlace
                const eventLabel = props.publication + '-' +
props.contentCategory
                try {
                    window.ga('create', props.gaId, 'auto', 'jk')
                    window.ga('jk.send', 'event', eventCategory, 'show',
eventLabel)
                    const anchors = element.querySelectorAll('a')
                    for (let i = 0; i < anchors.length; i++) {
                        anchors[i].onclick = function() {
                            window.ga('jk.send', 'event', eventCategory,
'click', eventLabel)
                        }
                    }
                } catch(err) {
                    console.log(err)
                }
                eventSent = true
            }
        }

        // Timeout to prevent sending of show event when lazyload loads
component right when it enters the viewport
        setTimeout(function() {
            window.addEventListener('scroll', gaEventsOnScroll)
        }, 150)

        return function cleanupListener() {
            setTimeout(function() {
                window.removeEventListener('scroll', gaEventsOnScroll)
            }, 150)
        }
    })

    const refCallback = (element: HTMLDivElement) : void => {
        setElement(element)
    }

    return (
        <div ref={refCallback}>
            {props.children}
        </div>
    )
}

export default EcomEmbed
```

Esimerkkikoodi 14. Analytiikkatapahtumien ja komponentin kiinnityksestä ilmoittavan JavaScript-tapahtuman lähettämisen hoitava EcomEmbed.tsx-käärekomponentti.

Kehittämällä 14. esimerkkikoodissa näkyvää käärekomponenttia saatiin hoidettua tapahtumien lähetykseen liittyvät asiat yhdessä paikassa sen sijaan, että samaa koodia jouduttaisiin kopioimaan Iltalehden ja Uuden Suomen jakelukomponentteihin.

Molemmat, sekä Iltalehden että Uuden Suomen jakelukomponentit kehitettiin omiin tiedostoihin, jotka sisältävät kunkin komponentin vakion jakelulogiikan, näytettävän sisällön nimen palauttavan funktion ja funktion näytettävän sisällön manuaaliseen asettamiseen.

```
import * as React from 'react'
import AtEmbed from '@almamedia/mepa-autotalli-external-embed'
import { MbEmbed, TvEmbed } from '@almamedia/talent-news-embed'
import EoEmbed from '@almamedia/mepa-eo-wp-external-embed'
import MixedEmbed from '@almamedia/ecom-alma-mixed-embed'
import EcomEmbed from '../EcomEmbed'
import { weightedRandom } from '../utils'

export type Props = {
    className?: string
    gaId: string
    locationMedia: string
    locationPlace: string
}

export type State = {
    publication: string
}

const DIGTAL_SEGMENT = 'uhwfhq8gd'
const UNTARGETED_PUBLICATIONS = ['autotalli', 'etuovi', 'kooste', 'kooste-
branded']//Publications to pick from when user has no targeted publication
export const ALL_PUBLICATIONS = ['tivi', 'mikrobitti',
...UNTARGETED_PUBLICATIONS]

export default class UsSharedEmbed extends React.Component<Props, State> {
    constructor(props : Props) {
        super(props)
        this.state = {
            publication: 'none'
        }
    }

    componentDidMount() {
        this.setPublicationBySegment()
    }

    setPublicationByName = (name: string) : void => {
        name = name.toLowerCase()
        if (ALL_PUBLICATIONS.includes(name)) {
            this.setState({publication: name})
        }
    }

    getPublicationName() {
        return this.state.publication
    }

    setPublicationBySegment = () : void => {
        let dtSegment = false
        let decidedPublication

        if (this.state.publication === 'none') {//Don't set new publication by
segment if publication is already set
            decidedPublication =
UNTARGETED_PUBLICATIONS[Math.floor(Math.random() *
```

```
UNTARGETED_PUBLICATIONS.length)]//Pick randomly from publications targeted to
any user
            if (window.ALMA && window.ALMA.dmp) {
                const segments = window.ALMA.dmp.segments()
                dtSegment = segments.includes(DIGTAL_SEGMENT)

                if (dtSegment) {
                    decidedPublication = weightedRandom([{weight: 0.4, pub:
'tivi'}, {weight: 0.4, pub: 'mikrobitti'}, {weight: 0.1, pub: 'kooste'},
{weight: 0.1, pub: 'kooste-branded'}])
                }
            }
            this.setState({publication: decidedPublication})
        }
    }

    render() {
        const cn = this.props.className
        let SelectedComponent
        switch(this.state.publication) {
            case "mikrobitti":
                SelectedComponent = <MbEmbed category="digitalous"
className={cn} />
                break
            case "tivi":
                SelectedComponent = <TvEmbed category="digitalous"
className={cn} />
                break
            case "autotalli":
                SelectedComponent = <AtEmbed className={cn}
publication='autotalli' />
                break
            case "etuovi":
                SelectedComponent = <EoEmbed />
                break
            case "kooste":
                SelectedComponent = <MixedEmbed className={cn} />
                break
            case "kooste-branded":
                SelectedComponent = <MixedEmbed className={cn} branded />
                break
            default:
                return <span style={{"display": "none"}}></span>
        }

        const contentCategory = this.state.publication === 'mikrobitti' ||
this.state.publication === 'tivi' ? 'digitalous' : 'luetuimmat'

        return (
            <EcomEmbed locationMedia={this.props.locationMedia}
locationPlace={this.props.locationPlace} publication={this.state.publication}
gaId={this.props.gaId} contentCategory={contentCategory}>
                {SelectedComponent}
            </EcomEmbed>
        )
    }
}
```

Esimerkkikoodi 15. Uuden Suomen jakelukomponentti UsSharedEmbed.tsx. Iltalehden jakelukomponentissa erona tuodut ja käytetyt sisältökomponentit sekä vakion jakelulogiikan sisältävä setPublicationBySegment()-funktio.

Jotta projektiin, jossa jompaakumpaa jakelukomponenttia tullaan käyttämään, saataisiin tuotua vain tarvittava jakelukomponentti, kuten 17. esimerkkikoodissa, on moduuliprojektista vietävä molemmat jakelukomponentit nimettyinä, joka onnistuu moduuliprojektin index.ts-tiedostossa, 16. esimerkkikoodissa esitetyllä tavalla.

```
export { default as IlSharedEmbed } from './embeds/IlSharedEmbed'
export { default as UsSharedEmbed } from './embeds/UsSharedEmbed'
```

Esimerkkikoodi 16. Komponenttien nimetty vienti projektista, moduuliprojektin index.ts-tiedostossa.

```
import { UsSharedEmbed } from '@almamedia/ecom-il-jakelukomponentti'
```

Esimerkkikoodi 17. Uuden Suomen jakelukomponentin tuonti, esimerkiksi Uuden Suomen sivuston projektiin.

5.3 Paketin julkaisu

Koska paketti on kehitetty vain Alma Median sivustoilla käytettäväksi, se julkaistaan Alma Median yksityiseen NPM-rekisteriin. Tässä luvussa käydään läpi komponentin jatkuvan integraation ja julkaisun eri vaiheita sekä tarvittavia toimenpiteitä yksityisen paketin julkaisemiseen.

5.3.1 Koontipalvelimen määritys

Koontipalvelimena käytettiin Travis CI:n tarjoamaa pilvipalvelinta. Koontipalvelin määritettiin käynnistämään koonnin, ajamaan yksikkötestit ja julkaisemaan oikeantyyppisen version moduulista aina, kun GitHub-versionhallinnassa sijaitsevaan komponentin sisältävään repositoryyn viedään muutoksia. Käytettäessä GitHub:a versionhallintana saadaan Travis CI:n koontipalvelin käyttöön lisäämällä projektiin .travis.yml-määritystiedoston, ja viemällä sen versionhallintaan, jonka jälkeen koontipalvelin huomaa kaikki versionhallintaan vietävät muutokset automaattisesti, ja käsittelee ne määritystiedostossa määritetyllä tavalla. 18. esimerkkikoodissa on esitetty kehittämämme projektin .travis.yml-määritystiedosto.

```
notifications:
  email: false

language:
- node_js

node_js:
- "8"

install:
- make install
script:
- make build
- make test

env:
  global:
    secure:
"etfsXc+bca8bj7bFkRahbi2CuDD4HfHEm5jseqfUL3a8C7QL34LCsIxriE9hFjaGnChMCH2eoiYiV
HQRF5SnrmpdHclMt5BHucYhEAM0lWZyxTDJJug+61ShigHGWN9K6YYmlgSSHWeLCsqr3un/VCHjIBK
lwBcfwjlGeAyKi/5b/pzZdLGOSPj38u8GIXu+mTuUdJSRyj2e7qK/QAgw2Ayqi/02UeUCYZULneVnp
EPX/nbewCbMF1PLUbMw9Le+2xSgeRoohit9ZrKUPUbU+c8SfSkXhRawow4v4r5o217xLoqxrPd6+ar
EWgvwDemTrFsH1MjjIOUnXhfpr1G7nrniQRH7RLkxPTfpplf67tuygrjX56ipp47r1vTTBvjq875qp
OBkd5hVHzXYkW1FnFwc//LTifm2affhwZPfvd3U+r+UeQOclOzK/hvrsjXGyMweim2qU0+m69UjIWP
RkMrayXW2D4q7Lurh4gefAlCbO/oDVw5E5iftPrG5FJA/BXoIL7mLsMZpHrGSBObUL2yjaJfQXyWIE
Ss4B0wmwoqOcYSLb/q8nF8cqBS5UQVVwhnsY1QqXJ0lHmvgZFnIESdc/1rBLAgd1Xhrqs+/bHRqfnK
DDsbLy7O6HaIebgKBIwoIFYPhwUkhgTcXdp7qakLLRLHUirH1IwQQ2aYeEFc="

deploy:

# STAGING deploy
- provider: script
  skip_cleanup: true
  script: make deploy
  on:
    branch: master

# FEATURE branch deploy
- provider: script
  skip_cleanup: true
  script: make deploy
  on:
    all_branches: true
    condition: ${TRAVIS_PULL_REQUEST_BRANCH:-$TRAVIS_BRANCH} =~ ^feature\/.*$

# PRODUCTION DEPLOY
- provider: script
  skip_cleanup: true
  script: make deploy
  on:
    tags: true
```

Esimerkkikoodi 18. Kehitettävän projektin .travis.yml-määritystiedosto.

18. esimerkkikoodissa näkyvässä kehittämämme projektin määritystiedostossa on alussa määritetty koontiympäristön asetuksia. Ensimmäinen asetus poistaa käytöstä koontiprosessien tulosten lähettämisen sähköpostiin. Tuloksia voidaan tarvittaessa tarkastella travis-ci.com-verkkosivuilta löytyvästä hallintapaneelista. Sen jälkeen määritystiedostossa määritetään koontiympäristön ohjelmointikieleksi 8. version

node_js. Seuraavaksi on määritetty koontiympäristön pystyttämisen yhteydessä ajettava komento, joka asentaa tarvittavat riippuvuudet, jonka onnistuttua ajetaan seuraavissa luvuissa tarkemmin käsiteltävät koonti- ja testikomennot. Nämä, sekä määritystiedoston lopussa olevat julkaisukomennot ajetaan make-komentoa käyttäen. Komennolla make ohjataan UNIX:in alkuperäistä, make-nimistä koontityökalua, joka löytyy myös projektin käyttämästä koontipalvelimesta, sillä koontipalvelimen käyttöjärjestelmänä on UNIX. make-koontityökalun komennot saadaan määriteltyä projektin hakemistoon, Makefile-nimiseen tiedostoon, kuten 19. esimerkkikoodissa esitetyssä kehittämämme projektin Makefile-määritystiedostossa.

```
# marks targets as "PHONY" so that they can be run even if build/dist and
other folders exist
.PHONY: build test invoke

# define default target (which will be run if only 'make' command with no
targets issued)
.DEFAULT_GOAL := help

# VARIABLES
THIS_FILE := $(lastword $(MAKEFILE_LIST)) #
http://stackoverflow.com/a/27132934

C_LCYAN=\033[1;36m
C_CYAN=\033[0;36m
C_GREEN=\033[0;32m
C_YELLOW=\033[0;33m
C_DEFAULT=\033[0m # No Color

# TARGETS

clear:
    @(clear)

# Taken from http://marmelab.com/blog/2016/02/29/auto-documented-makefile.html
help: clear ## Prints this help message
    @echo "\n${C_GREEN}Available options:${C_DEFAULT}"
    @grep -E '^[a-zA-Z-]+:.*?## .*$$' $(MAKEFILE_LIST) | awk 'BEGIN {FS =
":.*?## "}; {printf "\033[36m%-30s\033[0m %s\n", $$1, $$2}'

install: ## Install depedencies
    @(npm install)

build: ## Compile TS to JS
    @(npm run build)

dev: ## Compile TS to JS in watch mode
    @(npm run watch)

test: build ## Runs tests
    @(npm run test)

resolve-version: ## Resolves version from Git
    @(npx @almamedia/alma-package-version-resolver)

deploy: resolve-version ## Publishes to NPM in this case
    @(npm publish)
```

Esimerkkikoodi 19. Kehitettävän projektin Makefile-määritystiedosto.

Noin puolessavälissä projektimme koontipalvelimen määritystiedostoa on 20. esimerkkikoodissa näkyvä osa. Se on ympäristömuuttuja, joka sisältää salatun NPM-tokenin, jota tarvitaan koontiympäristössä komponentin julkaisemiseen Alma Median yksityiseen NPM-rekisteriin.

```
env:
  global:
    secure: "
etfsXc+bca8bj7bFkRahbi2CuDD4HfHEm5jseqfUL3a8C7QL34LCsIxriE9hFjaGnChMCH2eoiYiVH
QRF5SnrmpdHclMt5BHucYhEAM0lWZyxTDJJug+61ShigHGWN9K6YYmlgSSHWeLCsqr3un/VCHjIBKl
wBcfwjlGeAyKi/5b/pzZdLGOSPj38u8GIXu+mTuUdJSRyj2e7qK/QAgw2Ayqi/02UeUCYZULneVnpE
PX/nbewCbMF1PLUbMw9Le+2xSgeRoohit9ZrKUPUbU+c8SfSkXhRawow4v4r5o217xLoqxrPd6+arE
WgvwDemTrFsH1MjjIOUnXhfpr1G7nrniQRH7RLkxPTfpplf67tuygrjX56ipp47r1vTTBvjq875qpO
Bkd5hVHzXYkW1FnFwc//LTifm2affhwZPfvd3U+r+UeQOclOzK/hvrsjXGyMweim2qU0+m69UjIWPR
kMrayXW2D4q7Lurh4gefAlCbO/oDVw5E5iftPrG5FJA/BXoIL7mLsMZpHrGSBObUL2yjaJfQXyWIES
s4B0wmwoqOcYSLb/q8nF8cqBS5UQVVwhnsY1QqXJ0lHmvgZFnIESdc/1rBLAgd1Xhrqs+/bHRqfnKD
DsbLy7O6HaIebgKBIwoIFYPhwUkhgTcXdp7qakLLRLHUirH1IwQQ2aYeEFc="
```

Esimerkkikoodi 20. NPM-token salattuna, koontiympäristön ympäristömuuttujana.

5.3.2 Paketin koonti

18. esimerkkikoodissa esitetyssä koontipalvelimen määritystiedostossa kutsutaan koontikomentoa 21. esimerkkikoodissa esitetyllä tavalla.

```
- make build
```

Esimerkkikoodi 21. Koontikomento .travis.yml-koontipalvelimen määritystiedostossa.

19. esimerkkikoodissa esitetyssä Makefile-määritystiedostossa nähdään, että build -komento viittaa NPM:n build skriptiin.

```
build: ## Compile TS to JS
    @(npm run build)
```

Esimerkkikoodi 22. build-komennon määritelmä Makefile-määritystiedostossa.

```
"build": "tsc --module commonjs && ./node_modules/.bin/webpack --
mode=production && babel dist/bundle -d dist --copy-files"
```

Esimerkkikoodi 23. package.json eli NPM:n määritystiedostossa määritetty koontikomento.

23. esimerkkikoodissa esitetyssä koontikomennossa on yhdistetty kolme komentoa ”&&”-operaattoria käyttäen, eli komennot suoritetaan vuorotellen, vain jos edellinen komento suoriutuu onnistuneesti. Ensimmäinen komento on 24. esimerkkikoodissa esitetty komento, joka kääntää TypeScriptin JavaScriptiksi, TypeScriptin kääntäjän määritystiedostossa tsconfig.json:ssa määritetyllä tavalla. 25. esimerkkikoodissa on esitetty kehittämämme projektin tsconfig.json-määritystiedosto.

```
tsc –module commonjs
```

Esimerkkikoodi 24. Komento TypeScriptin kääntämiseen JavaScriptiksi, TypeScriptin omaa kääntäjää käyttäen.

```
{
  "compilerOptions": {
    "moduleResolution": "node",
    "target": "es5",
    "module": "es2015",
    "lib": ["es2015", "es2016", "es2017", "dom"],
    "jsx": "react",
    "strict": true,
    "sourceMap": true,
    "declaration": true,
    "allowSyntheticDefaultImports": true,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "removeComments": true,
    "declarationDir": "dist/types",
    "outDir": "dist/lib",
    "typeRoots": ["node_modules/@types"],
    "skipLibCheck": true,
    "esModuleInterop": true
  },
  "include": ["src"]
}
```

Esimerkkikoodi 25. Kehittämämme projektin TypeScript-määritys tsconfig.json-määritystiedostossa.

Käännettyään TypeScriptin onnistuneesti JavaScriptiksi seuraavaksi kutsutaan 26. esimerkkikoodissa näkyvää Webpack:n paketointi komentoa, joka paketoi TypeScript-kääntäjän aikaansaamaa koodia webpack.config.js, eli Webpackin määritystiedostossa määritetyllä tavalla. Kehittämämme projektin Webpack-määritystiedosto on esitetty esimerkkikoodissa 27.

```
./node_modules/.bin/webpack --mode=production
```

Esimerkkikoodi 26. Webpack-paketointi komento.

```
var path = require('path');
module.exports = {
  entry: './dist/lib/index.js',
  output: {
    path: path.resolve(__dirname, 'dist/bundle'),
    filename: 'index.js',
    libraryTarget: 'commonjs2'
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        include: path.resolve(__dirname, 'dist/lib'),
        exclude: /(node_modules|bower_components|build)/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: ['@babel/preset-env']
          }
        }
      }
    ]
  },
  externals: {
    'react': 'commonjs react'
  }
};
```

Esimerkkikoodi 27. Kehittämämme projektin Webpack-määritys webpack.config.js-määritystiedostossa.

Webpack paketoi ohjelman koodin, eli yhdistää kaikki moduulit yhteen tiedostoon, minifioi sen sekä jättää siitä pois käyttämättömän koodin. Onnistuneen paketoinnin jälkeen ajetaan 28. esimekkikoodissa näkyvä Babel-komento, joka kääntää Webpackin tuottaman koodin Babelin määritystiedostossa, .babelrc:ssä määritetyllä tavalla. Projektimme Babel-määritys on esitetty 29. esimerkkikoodissa, joka on määritetty kääntämään ohjelman JavaScript-koodin myös vanhimmilla selaimilla toimivaksi.

```
babel dist/bundle -d dist --copy-files
```

Esimerkkikoodi 28. Babel-kääntäjän käynnistävä komento.

```
{
    "presets": ["@babel/preset-env", "@babel/preset-react"],
    "plugins": [
```

```
        "@babel/plugin-proposal-object-rest-spread",
        "@babel/plugin-transform-react-jsx"
    ]
}
```

Esimerkkikoodi 29. Babel-määritystiedosto, .babelrc.

Ennen tässä luvussa käsiteltyjä koonti-prosesseja ajetaan kuitenkin vielä 30. esimerkkikoodissa esitettyä komentoa, joka on määritetty NPM:n määritystiedostossa, eli package.json-tiedostossa, ”prebuild”-kenttään. Tämä komento ajetaan aina ennen ”build”-kenttään määritettyjä komentoja. Meidän tapauksessamme se tyhjentää dist-hakemiston, joka on määrittämämme koonnin kohdehakemisto käyttäen apuna rimraf-kirjastoa.

```
rimraf dist
```

Esimerkkikoodi 30. Rimraf-komento dist-hakemiston tyhjentämiseksi.

5.3.3 Paketin testaus

18. esimerkkikoodissa esitetyssä koontipalvelimen määritystiedostosta nähdään, että onnistuneen koonnin jälkeen kutsutaan testikomentoa 31. esimerkkikoodissa esitetyllä tavalla.

```
- make test
```

Esimerkkikoodi 31. Testikomento .travis.yml-koontipalvelimen määritystiedostossa.

19. esimerkkikoodissa esitetyssä Makefile-määritystiedostossa nähdään, että test-komento viittaa NPM:n test skriptiin.

```
test: build ## Runs tests
    @(npm run test)
```

Esimerkkikoodi 32. test-komennon määritelmä Makefile-määritystiedostossa.

```
"test": "jest"
```

Esimerkkikoodi 33. package.json eli NPM:n määritystiedostossa määritetty testikomento.

33. esimerkkikoodissa esitetyssä testikomennossa kutsutaan ”jest”-komentoa, joka ajaa automaattisesti kaikki Jest-yksikkötestit. Projektissamme on määritetty omat testisarjat molemmalle jakelukomponentille. Koska komponenttien toiminta on hyvin samanlaista, eivät testitkään eroa toisistaan paljoa. 34. esimerkkikoodissa on esitetty Uuden Suomen komponentin testaava testisarja.

```
import React from "react"
import ReactDOM from "react-dom"
import UsSharedEmbed, { ALL_PUBLICATIONS } from "../embeds/UsSharedEmbed"
import { commonTests } from '../commonTests'
global.fetch = require("node-fetch")

describe("UsSharedEmbed", () => {
    it("renders without crashing", () => {
        const div = document.createElement("div")
        ReactDOM.render(
            <UsSharedEmbed
                ref={ecomEmbed => {
                    global.window.ecomEmbed = ecomEmbed
                }}
                gaId="UA-53865955-1"
                locationMedia="uusisuomi"
                locationPlace="etusivu"
            />,
            div)
    })

    commonTests(ALL_PUBLICATIONS)
})
```

Esimerkkikoodi 34. UsSharedEmbed.test.tsx, Uuden Suomen komponentin testaava testisarja.

Tietyn komponentin testisarjassa on hoidettu vain komponentin kiinnitys ja kiinnityksen toimivuuden testaus. Loput yksikkötestit on ulkoistettu 35. esimerkkikoodissa esitettyyn commonTests.ts-tiedostossa sijaitsevaan funktioon, sillä ne ovat identtisiä Iltalehden ja Uuden Suomen tapauksissa. Näissä testeissä testataan komponenttien funktioita, joilla saa haettua esitetyn sisällön nimen sekä asetettua esitettävän sisällön.

```
import { act } from 'react-dom/test-utils'

export function commonTests(ALL_PUBLICATIONS: string[]) {
    it("creates globally accessible object", () => {
        expect(global.window.ecomEmbed!.getPublicationName()).not.toEqual("")
    })

    it("renders actual publication, and returns its name from
ecomEmbed.getPublicationName()", () => {

expect(ALL_PUBLICATIONS).toContain(global.window.ecomEmbed!.getPublicationName
())
    })
```

```
    it("sets its publication manually with ecomEmbed.setPublicationByName()",
() => {
        act(() => {
            global.window.ecomEmbed!.setPublicationByName("autotalli")
        })

expect(global.window.ecomEmbed!.getPublicationName()).toEqual("autotalli")
        act(() => {
            global.window.ecomEmbed!.setPublicationByName("etuovi")
        })

expect(global.window.ecomEmbed!.getPublicationName()).toEqual("etuovi")
    })

    it("renders successfully all publications it has", () => {
        for (var i = 0; i < ALL_PUBLICATIONS.length; i++) {
            act(() => {

global.window.ecomEmbed!.setPublicationByName(ALL_PUBLICATIONS[i])
            })

expect(ALL_PUBLICATIONS[i]).toEqual(global.window.ecomEmbed!.getPublicationNam
e())
        }
    })
}
```

Esimerkkikoodi 35. Jakelukomponenttien yhteiset yksikkötestit sisältävä commonTests.ts-tiedosto.

5.3.4 Paketin julkaisu

18. esimerkkikoodissa esitetyssä koontipalvelimen määritystiedoston lopussa on määritetty julkaisuprosessi 36. esimerkkikoodissa esitetyllä tavalla.

```
deploy:

# STAGING deploy
- provider: script
  skip_cleanup: true
  script: make deploy
  on:
    branch: master

# FEATURE branch deploy
- provider: script
  skip_cleanup: true
  script: make deploy
  on:
    all_branches: true
    condition: ${TRAVIS_PULL_REQUEST_BRANCH:-$TRAVIS_BRANCH} =~ ^feature\/.*$

# PRODUCTION DEPLOY
- provider: script
  skip_cleanup: true
  script: make deploy
  on:
    tags: true
```

Esimerkkikoodi 36. Julkaisuprosessin määritys .travis.yml-koontipalvelimen määritystiedostossa.

Julkaisuprosessi on määritetty niin, että muutosten vienti versionhallinnan päähaaraan julkaisee automaattisesti paketista staging-version, jonka saa asennettua, lisättyään paketin nimen perään ”@staging”-päätteen, kuten 37. esimerkkikoodissa on esitetty.

```
npm install @almamedia/ecom-il-jakelukomponentti@staging
```

Esimerkkikoodi 37. Komento, jolla NPM-rekisteristä saadaan asennettua kehittämämme paketin staging-version.

Lisäksi, paketista saadaan julkaistua feature -versio, viemällä haaran versionhallintaan, jonka nimessä on ”feature/”-etuliite. Tällä saadaan julkaistua esimerkiksi jotain uutta testitoiminnallisuutta sisältäviä versioita, joita saadaan asennettua testattavaksi vain johonkin tiettyyn palveluun. Esimerkiksi ”feature/new-analytics”-haaraan viedyn version saadaan asennettua 38. esimerkkikoodissa esitetyllä tavalla.

```
npm install @almamedia/ecom-il-jakelukomponentti@feature-new-analytics
```

Esimerkkikoodi 38. Komento, jolla NPM-rekisteristä saadaan asennettua kehittämämme komponentin ”feature/new-analytics”-haaraan viedyn feature-version.

Tuotantoversion komponentista saadaan julkaistua viemällä versionhallintaan tagin, joka on nimetty julkaistavan version semanttisena versionumerona. Esimerkkikoodissa 39 on esitetty 1.4.0-version julkaisemiseen tarvittavat komennot.

```
git tag -a v1.4.0 -m ”update contents”
git push origin v1.4.0
```

Esimerkkikoodi 39. ”v1.4.0”-tagin vienti versionhallintaan, joka käynnistää vastaavan version julkaisuprosessit.

Tähän on käytetty apuna Alma Median kehittämää, Alma Median yksityisestä NPM-rekisteristä asennettavaa ”alma-package-version-resolver”-pakettia, joka asettaa automaattisesti tagin version tarvittaviin paikkoihin.

Jotta koontipalvelin saisi julkaistua paketin yksityiseen rekisteriin, on projektista löydyttävä NPM:n määritystiedosto, .npmrc, 40. esimerkkikoodissa esitetyllä sisällöllä. Jottei salaista NPM-tokenia tarvitsisi julkaista versionhallintaan, sen paikalla käytetään

”NPM_TOKEN”-ympäristömuuttujaa, jonka arvoksi on määritetty koontipalvelimella salattu NPM-token -arvo.

```
//registry.npmjs.org/:_authToken=${NPM_TOKEN}
```

Esimerkkikoodi 40. .npmrc, NPM:n määritystiedosto.

Lisäksi projektistamme löytyy .npmignore-tiedosto, jossa on määritetty tiedostot, joita ei viedä NPM-rekisteriin, eli kaikki muut paitsi koonnin tuloksena saadut tiedostot.

6 Yhteenveto

Vaikka muutosten teosta on tehty mahdollisimman helppoa, muutosten teko sisältökomponenttiin vaatii sisältökomponentin julkaisun, jonka jälkeen se on päivitettävä jakelukomponenttiin, jakelukomponentista on julkaistava uusi versio ja päivitettävä tämä vielä käytettävälle sivustolle. Tämä lisää muutosten tekoon toistuvaa manuaalista työtä. Toistuva komponenttien julkaiseminen ja päivittäminen johtuu siitä, että sisältö ja jakelulogiikka ovat sijoitettu erillisiin komponentteihin ja moduuleihin, sillä yksittäisiä sisältökomponentteja käytetään joissain tapauksissa ilman jakelulogiikkaa, ja osa näistä sisältökomponenteista on myös kehitetty ennen jakelukomponenttia, joka niitä yhdistää. Tätä voisi parantaa lisäämällä kaikki sisältökomponentit jakelukomponentin moduuliin, josta niitä saisi tuotua yksittäin. Tässä tapauksessa kaikki muutokset, sekä erillisten digikehitystiimien muutokset sisältö komponentteihinsa että jakelulogiikkaan, saataisiin tehtyä yhdessä monosäiliössä.

Kehitettyä komponenttikirjastoa käytetään tällä hetkellä Iltalehden ja Uuden Suomen etusivuilla. Jakelukomponentilla on testattu muutamia jakelulogiikoita, ja suunnitteilla on vielä testejä, joiden perusteella jakelukomponentteihin tulee mahdollisesti vielä muutoksia jakelulogiikkaan, jonka lisäksi näytettäviä sisältökomponentteja on todennäköisesti tulevaisuudessa tulossa lisää. Lisäksi jakelukomponenttia otetaan mahdollisesti käyttöön uusilla paikoilla. Projektin toteutuksessa käytettyjen ja opinnäytetyössä käsiteltyjen toimintamenetelmien ansiosta, edellä mainitut toimenpiteet kuten komponentin toiminnan muokkaus tai käyttöönotto uudella sijainnilla ovat selkeitä toteuttaa ja automatisoitujen prosessien sekä kehityksessä käytettyjen suunnittelumallien ansiosta vaativat minimaalista työtä.

Lisäksi projektin muut tärkeät vaatimukset, kuten mahdollisuus muuttaa väliaikaisesti komponentin jakelulogiikkaa ilman sen päivitystä sekä tuotantoon vientiä, sekä analytiikkadatan keruu komponentin käytöstä on saatu toteutettua.

Lähteet

1 Heiskanen Henri. 2013. Yhden sivun web-sovellukset tulevat, oletko valmis? Verkkoaineisto. Gofore<https://gofore.com/yhden-sivun-web-sovellukset-tulevat-oletko-valmis/>. Luettu 01.03.2020.

2 Alberto Gimeno. 2018. The deepest reason why modern JavaScript frameworks exist. Medium<https://medium.com/dailyjs/the-deepest-reason-why-modern-javascript-frameworks-exist-933b86ebc445>. Luettu 01.03.2020.

3 Mosh Hamedani. 2018. React vs. Angular: The Complete Comparison. <https://programmingwithmosh.com/react/react-vs-angular/> Luettu 13.04.2020.

4 React Without JSX. Verkkodokumentti<https://reactjs.org/docs/react-without-jsx.html> Luettu 02.05.2020.

5 Karthik Kalyanaraman. 2019. A deep dive into React Fiber internals. <https://blog.logrocket.com/deep-dive-into-react-fiber-internals/> Luettu 09.05.2020.

6 About packages and modules. Verkkodokumentti<https://docs.npmjs.com/about-packages-and-modules> Luettu 29.06.2020.

7 Module Counts. Verkkodokumentti<http://www.modulecounts.com/> Luettu 29.06.2020.

8 Mikko Jämiä. 2019. Tietoturvaongelmien havaitseminen Node.js-ympäristöissä. <http://www.modulecounts.com/> Luettu 02.07.2020.