Tampereen yliopisto

Petteri Ranto

WEB-SOVELLUKSEN PDF-RAPORTOINNIN PARANTAMINEN: TEKNOLOGIAKARTOITUS

Diplomityö
Informaatioteknologian ja viestinnän tiedekunta
Yliopistonlehtori Terhi Kilamo
Tenure Track -tutkija Outi Sievi-Korte
Syyskuu 2020

TIIVISTELMÄ

Petteri Ranto: Web-sovelluksen PDF-raportoinnin uudistaminen: teknologiakartoitus
Diplomityö
Tampereen yliopisto
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Syyskuu 2020

Raportointi on osa yritysten liiketoimintaa. Raportteja tuotetaan usealle sidosryhmälle ja eri tarkoituksiin. Raportit ovat usein PDF-muotoisia ja tulostamisen lisäksi niitä tallennetaan tietokoneille ja älylaitteille.

Tämä työ on tehty tamperelaiselle ohjelmistotalolle, joka tuottaa yrityksille myytävää Web-sovellusta. Raportointi, ja tarkemmin PDF-raportointi, on tärkeä osa kohdeyrityksen sovellusta. Kohdeyritys suunnittelee uudistavansa sovelluksensa PDF-raportointia. PDF-raportointia on tarkoitus uudistaa sekä teknologia- että käytettävyystasolla.

Tässä työssä tutkittiin, vertailtiin ja testattiin eri toteutusvaihtoehtoja PDF-raportoinnin toteuttamiseksi kohdeyrityksen sovellukseen. Toteutusvaihtoehtoja valittiin kolme, ja valinta perustui osin kohdeyritykseltä tulleisiin ehdotuksiin ja osin ennen työtä tehtyyn esiselvitykseen. Työhön valitut toteutusvaihtoehdot olivat selainpohjainen ratkaisu käyttäen apuna React-kirjastoa, mikropalvelupohjainen ratkaisu käyttäen apuna React- ja Puppeteer-kirjastoja sekä mikropalvelupohjainen ratkaisu käyttäen apuna Handlebars- ja Puppeteer-kirjastoja. Työn tarkoitus oli tuottaa esitietoa kohdeyrityksen sisäiselle PDF-raportoinnin uudistusprojektille.

Työssä käsiteltäviä toteutusvaihtoehtoja vertailtiin toiminnallisten ja laadullisten vaatimuksien sekä toteutusvaihtoehdon käyttöönoton ja käyttämisen helppouden suhteen. Jokaisesta toteutusvaihtoehdosta tehtiin pienimuotoinen sovellus, joiden avulla vertailu ja testaaminen tapahtui. Vaatimukset uudelle PDF-raportoinnin toteutustavalle tunnistettiin pääosin kohdeyrityksen asiakastiimin esittämistä käyttäjätarinoista ja kohdeyrityksen tämänhetkisen sovelluksen PDF-raportointiominaisuuksien pohjalta.

Työssä tehtiin arkkitehtuurin arviointi toteutusvaihtoehdolle, joka koettiin hyödyllisimmäksi arvioitavaksi vaihtoehdoksi. Arkkitehtuurin arviointi suoritettiin kohdeyrityksessä DCAR-arviointimenetelmällä (Decision-Centric Architecture Reviews).

Työn tuloksena saatiin arvio parhaasta toteutusvaihtoehdosta kohdeyrityksen tarpeisiin, ottaen huomioon annetut vaatimukset ja toteutusvaihtoehdon käyttöönoton sekä käytön helppous. DCAR:illa suoritetusta arkkitehtuurin arvioinnista kohdeyritys sai tietoa arvioidusta arkkitehtuurista, sen soveltuvuudesta kohdeyrityksen sovellukseen sekä arkkitehtuurin hyvistä ja huonoista puolista. Näiden lisäksi kohdeyritys sai tietoa DCAR-menetelmästä ja kuinka hyödyntää sitä tulevissa projekteissa.

Avainsanat: PDF, PDF-raportointi, React, Handlebars, Puppeteer, Node.js, mikropalvelu

ABSTRACT

Petteri Ranto: Renewing PDF reporting of a web application: technology mapping
Master's thesis
Tampere University
Master's Degree Programme in Information Technology
September 2020

Reporting is a part of everyday business. Reports are produced for several stakeholders and for different purposes. Reports are often in PDF format and, in addition to printing, they are stored on computers and smart devices.

This thesis was done for a Tampere-based software company, which produces a web application for other companies. Reporting, and more specifically PDF reporting, is an important part of the target company's application. The company is planning to improve both technological and usability aspects of the PDF reporting of their application.

In this thesis, three different options for implementing PDF reporting in the target company's application were studied, compared and tested to produce preliminary information for the renewal project of the PDF reporting. The methods were selected partly based on proposals from the target company and partly on a preliminary study carried out before starting this thesis. The implementation options chosen for this thesis were a browser-based solution utilizing React library, a microservice-based solution utilizing React and Puppeteer libraries and a microservice-based solution utilizing Handlebars and Puppeteer libraries.

The implementation options were compared in terms of functional and qualitative requirements as well as the ease of their implementation and usage. Each implementation option was carried out as a small-scale application that was used for comparison and testing. The requirements for the new PDF reporting method were identified mainly from the user stories presented by the target company's customer team and the features of the currently used application.

This thesis also included an evaluation of the architecture of an implementation option, that was evaluated as the most beneficial to evaluate. The evaluation of the architecture was performed in the target company using DCAR (Decision-Centric Architecture Reviews) evaluation method.

The thesis resulted in a suggestion of the best implementation option considering the needs of the target company, as well as the given requirements and the ease of implementation and usage. As a result of the architecture evaluation performed with DCAR, the target company received information about the assessed architecture, its suitability and the pros and cons of the architecture. In addition to these, the target company received information about the DCAR method and learned how to utilize it in their future projects.

Keywords: PDF, PDF reporting, React, Handlebars, Puppeteer, Node.js, microservice

ALKUSANAT

Kirjoitin tämän diplomityön kesän 2020 aikana työnantajalleni, jolla olen työskennellyt opintojen ohessa jo puolitoista vuotta. Yhdessä esimieheni kanssa keksimme hyvän aiheen monien aiheiden joukosta, ja tämä valittu aihe vastasi hyvin työnkuvaani ja oli työnantajalleni ajankohtainen.

Diplomityön teko oli iso prosessi päivittäisen työn rinnalla. Haluankin kiittää perhettäni ja ystäviäni, jotka olivat tukemassa ja kannustamassa. Olen lopputulokseen tyytyväinen.

Haluan kiittää Anssi Väisästä diplomityön mahdollistamisesta ja hänen antamastaan luottamuksesta ja vapaudesta työtä kohtaan. Haluan kiittää myös Heikki Parkkosta hyvistä vinkeistä ja tuesta diplomityöhön. Kiitän myös työn ohjaajaa Terhi Kilamoa palautteista ja neuvoista diplomityön aikana.

Tampereella, 18.09.2020

Petteri Ranto

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

ALMA	Architecture-Level Modifiability Analysis, arkkitehtuurinarviointimenetelmä
ARID	Active Review of Intermediate Designs, arkkitehtuurinarviointimenetelmä
ATAM	Architecture Tradeoff Analysis Method, arkkitehtuurinarviointimenetelmä
CSS	Cascading Style Sheets, HTML- tai XML-muotoisten dokumenttien tyylittelyyn tarkoitettu kieli
DCAR	Decision-Centric Architecture Reviews, arkkitehtuurinarviointimenetelmä
FAAM	Family-Architecture Assessment Method, arkkitehtuurinarviointimenetelmä
HTML	HyperText Markup Language, merkintäkieli Web-dokumenteille
IEEE	Institute of Electrical and Electronics Engineers, ammatillinen yhdistys
JSON	JavaScript Object Notation, tiedonsiirtoformaatti
PDF	Portable Document Format, dokumenttiformaatti
SAAM	Software Architecture Analysis Method, arkkitehtuurinarviointimenetelmä
SaaS	Software as a Service, ohjelmistojen toimitusmalli
URI	Uniform Resource Identifier, tietoverkoissa käytetty tunniste
W3C	World Wide Web Consortium, World Wide Web:in kansainvälinen standardointijärjestö
XML	Extensible Markup Language, merkintäkieli

.

1. JOHDANTO

Erilaiset raportit ovat tärkeä osa jokaisen yrityksen liiketoimintaa. Raportteja täytyy tuottaa useaan eri tarkoitukseen ja usealle eri sidosryhmälle. Raportointitarve voi liittyä esimerkiksi yrityksen kirjanpitoon, laskutukseen, turvallisuuteen tai riskienhallintaan.

PDF-dokumentit (Portable Document Format) ovat olleet olemassa jo 27 vuotta, ja ne ovat edelleen laajalti käytössä. Formatoitujen dokumenttien lopullinen muoto onkin usein PDF. Ensimmäisen version PDF:stä julkaisi Adobe Systems Incorporated vuonna 1993. PDF-dokumentit voivat sisältää monentyyppistä sisältöä, kuten tekstiä, kuvia, hyperlinkkejä, multimediaa, kirjanmerkkejä ja metadataa. PDF tukee myös digitaalisia allekirjoituksia. [1] Tätä ominaisuutta käytetään esimerkiksi työsopimusten allekirjoittamisessa.

Työn tavoitteena on tutkia, miten toteuttaa moderni, suorituskykyinen ja raportointia helpottava PDF-generointipalvelu. Työ toimii tutkimuksena ja esitietona tamperelaisen ohjelmistotalon projektille. Järjestelmä, johon projekti kohdistuu, on yritysasiakkaille myytävä Web-sovellus. Kohdejärjestelmän käyttö perustuu vahvasti raportointiin, joten PDF-raporttien tuottaminen on tärkeää. Projekti, jolle tutkimusta tehdään, on sisäinen refaktorointiprojekti, jonka tarkoituksena on korvata kohdejärjestelmän vanha tapa generoida PDF-raportteja.

Työssä vertaillaan kolmea eri vaihtoehtoa PDF-generointipalvelun toteuttamiseen ja tehdään käytännön kokeiluja, joilla selvitetään, mikä vaihtoehdoista soveltuisi parhaiten kohdeyrityksen PDF-generointipalveluksi. Käytännön kokeilujen jälkeen otetaan yksi vaihtoehto tarkempaan arkkitehtuurin arviointiin. Lopuksi tulokseksi saadaan arvio soveltuvimmasta vaihtoehdosta ja lisäksi tietoa vaihtoehtojen hyvistä ja huonoista puolista, ja siitä kuinka tutkimusta kannattaisi jatkaa.

Työn toisessa luvussa tutustutaan työn ohjelmistoihin liittyviin käsitteisiin ja arkkitehtuureihin sekä osaan työssä tarvittavista teknologioista. Kolmannessa luvussa tutustutaan arkkitehtuurien arviointiin ja kahteen arviointimenetelmään. Kohdejärjestelmä, sen PDF-generointitapa ja generointitavan ongelmat kuvataan neljännessä luvussa. Viides luku koostuu uuden PDF-generointipalvelun arkkitehtuurin vaatimuksista ja toteutusvaihtoehdoista. Kuudennessa luvussa tehdään käytännön kokeiluja, joilla yritetään selvittää kolmesta vaihtoehdosta paras toteutustapa uudelle PDF-generointipalvelulle. Seitsemännessä luvussa arvioidaan työssä hyödyllisimmäksi arvioitavaksi todettu vaihtoehto yh-

dellä arkkitehtuurienarviointimenetelmällä, tarkastellaan sekä arvioinnin että työn tuloksia ja työssä tunnistettuja jatkotutkimusmahdollisuuksia. Viimeisessä luvussa on työn yhteenveto.

2. TAUSTAA

Tässä luvussa käydään läpi työhön liittyviä käsitteitä ja teknologioita, jotka auttavat ymmärtämään PDF-generointipalvelun toteutusvaihtoehtoja. Ensin käydään läpi ohjelmistoihin ja niiden arkkitehtuureihin liittyviä palvelukäsitteitä. Palvelukäsitteiden jälkeen käydään läpi asiakas-palvelin-malli, jota työn kohdejärjestelmä noudattaa. Lopuksi tutustutaan työssä käytettäviin pääteknologioihin.

2.1 Web-palvelut

Sovellusta, joka on muiden sovellusten saatavilla Webissä, kutsutaan usein Web-palveluksi [2, s. 124]. Web-palvelut voidaan määritellä internetiin suuntautuneina, itsenäisinä, modulaarisina yrityssovelluksina, joilla on avoimet standardipohjaiset rajapinnat [3, s. 1]. W3C-standardointijärjestö (World Wide Web Consortium) määrittelee Web-palvelun URI-tunnisteella (Uniform Resource Identifier) identifioitavana sovelluksena, jonka julkiset rajapinnat ja kytkökset on määritelty ja kuvattu käyttäen XML-kieltä (Extensible Markup Language). Ohjelmistojärjestelmät voivat olla vuorovaikutuksessa Web-palvelun kanssa sen määrittelemällä tavalla käyttäen internet-protokollilla välitettyjä XML-pohjaisia viestejä. [4] Vaikka W3C määritteleekin Web-palveluille tietyt standardit, eivät ne kuitenkaan ole Web-palvelu-teknologian ydin [2, s.125].

Web-palvelut yhdistävät laitteita toisiinsa käyttäen internetiä datan välittämiseen ja yhdistämiseen uudella tavalla. Web-palvelut voidaan määritellä ohjelmisto-objekteina, jotka kootaan yhteen internetin välityksellä suorittamaan erilaisia toiminnallisuuksia tai liiketoimintaprosesseja. Ohjelmisto voidaan toimittaa sekä maksaa jatkuvana virtana palveluita, toisin kuin pakettituotteet. Yrityspalvelut voidaan hajauttaa kokonaan Web-palveluiden avulla. Ne voidaan jakaa internetin yli ja niihin voidaan ottaa yhteyttä monilla eri laitteilla. Tämä helpottaa yrityksien taakkaa monimutkaisista, hitaista ja kalliista ohjelmistointegroinneista. Yritykset voivat keskittyä kriittisempiin tehtäviin ja kasvattaa oman tarjontansa arvoa. [5, s. 38]

2.2 Palveluorientoitunut arkkitehtuuri

Palveluorientoitunut arkkitehtuuri on järjestelmäsuunnittelun lähestymistapa, jossa järjestelmän suunnittelulla ja kehityksellä pyritään rakentamaan uudelleen käytettäviä palveluita. Palveluiden ajatellaan olevan kokoelma ohjelmistokomponentteja, ja niiden teh-

tävänä on hoitaa liiketoimintaprosesseja itsenäisesti. Palveluilla tulisi olla hyvin määritellyt alustariippumattomat rajapinnat. Palveluiden tulisi olla myös omavaraisia ja riippumattomia muista palveluista. Palveluiden riippumattomuus, tai toisin sanoen niiden löyhät kytkökset (eng. loose coupling), tarkoittavat, että palveluiden ei tarvitse tietää toisten palveluiden yksityiskohtia, jotta ne pystyisivät kommunikoimaan niiden kanssa. [6, s. 325–326][7]

Palveluiden löyhät kytkökset muihin ovat tärkeitä palveluja käyttäville sovelluksille ylläpidettävyyden ja skaalautumisen kannalta. Ilman löyhiä kytköksiä muutokset palveluissa vaikuttaisivat suoraan palveluita käyttäviin osapuoliin. [6, s. 326]

Palveluorientoituneen arkkitehtuurin tyypillisiä ominaisuuksia ovat [8]:

- Palvelu on sen toiminnan pohjalta määritelty ”looginen näkymä” ohjelmista, tietokannoista ja liiketoimintaprosesseista.
- Palvelu on muodollisesti määritelty palveluntarjoajien ja palveluja pyytävien välillä vaihdettujen viestien, eikä osapuolten ominaisuuksien perusteella.
- Palvelulla tulisi olla julkinen kuvaus, joka sisältää vain tärkeää tietoa palvelun käytöstä.
- Palvelut käyttävät vähän operaatioita suurien ja monimutkaisten viestien kanssa.
- Palvelut ovat suuntautuneet käytettäväksi verkon ylitse.
- Palvelut välittävät viestejä alustariippumattomilla standardoiduilla formaateilla erilaisten rajapintojen kautta.

2.3 Mikropalvelut

Mikropalvelut (eng. microservices) on termi ohjelmistoarkkitehtuurille, jossa yksittäiset sovellukset koostuvat monista pienistä palveluista. Nämä pienet palvelut toimivat omana prosessinaan ja käyttävät kommunikointiin kevyitä mekanismeja. Palvelut voidaan ottaa itsenäisesti käyttöön automaation avulla, kirjoittaa eri ohjelmointikielillä ja ne voivat käyttää erilaisia tallennustekniikoita. [9]

Isoja järjestelmiä voidaan pilkkoa pienemmiksi joukoiksi itsenäisiä palveluita ja näin hallita kasvavaa kompleksisuutta. Mikropalvelut korostavat löyhiä kytköksiä tekemällä palveluista kokonaan itsenäisiä kehitys- ja käyttöönottomielessä. Tämä lähestymistapa tuo hyötyjä esimerkiksi ylläpidettävyyteen ja skaalautuvuuteen. [10, s. 201]

Mikropalvelujen ominaisuuksia [10, s. 198]:

- Mikropalvelut toteuttavat rajatun määrä toiminnallisuuksia, joten niiden koodikanta on pieni ja se puolestaan vähentää bugien laajuutta.
- Uusia versioita palveluista voidaan ottaa käyttöön vanhan rinnalle. Vanhasta palvelusta riippuvat palvelut voidaan vähitellen muokata vuorovaikuttamaan uuden kanssa.
- Yhden moduulin vaihtaminen mikropalveluarkkitehtuurissa ei vaadi koko järjestelmän uudelleenkäynnistystä. Uudelleenkäynnistys koskeen ainoastaan mikropalveluita kyseisessä moduulissa.
- Mikropalvelut soveltuvat säiliöintiin (eng. containerization) ja käyttöönottoympäristön konfigurointi voidaan tehdä kehittäjien tarpeisiin vastaten.
- Mikropalveluarkkitehtuurin skaalaaminen ei tarkoita järjestelmän kaikkien komponenttien monistamista, vaan kehittäjät voivat helposti ottaa käyttöön tai poistaa käytöstä palveluiden instansseja niiden kuorman mukaan.
- Mikropalvelut eivät edellytä tietyn kielen, ohjelmistokehyksien tai muiden resurssien käyttöä. Ainoa rajoitus yhteen toimiville mikropalveluille on niiden kommunikointiin käytetty tekniikka.

2.4 Asiakas-palvelin-malli

Asiakas-palvelin-mallissa asiakassovellus pyytää palveluita palvelinprosesseilta. Asiakassovellus ja palvelinprosessit kommunikoivat tietoverkon ylitse ja niitä ajetaan tyypillisesti eri tietokoneilta. Esimerkkinä voidaan pitää internetin käyttöä, jossa asiakas-palvelin-malli on käytössä. [11, s. 215]

Asiakassovellus tässä mallissa on prosessi tai ohjelma, joka lähettää suorituspyyntöjä palvelimille. Suorituspyynnöt palvelimelle tehdään verkon kautta. Tällainen pyyntö voi olla asiakastiedon hankkiminen tietokannasta tai tiedoston palauttaminen palvelimen muistista. [11, s. 215]

Palvelinprosessi tässä mallissa on osapuoli, joka odottaa asiakassovelluksen pyyntöjä. Kun palvelinprosessi vastaanottaa asiakassovellukselta tulleen pyynnön, se joko hylkää pyynnön tai suorittaa sen ja palauttaa vastauksen asiakassovellukselle. Palvelinprosessi on usein päättymätön prosessi, ja se palvelee yhtä tai useampaa asiakassovellusta. [11, s. 215][12]

2.5 Pääteknologiat

Seuraavaksi käydään lyhyesti läpi työn kannalta merkittävimmät teknologiat. Kaikki tässä käytävät teknologiat ovat käytössä työn jokaisessa PDF-generointipalvelun toteutusvaihtoehdossa.

2.5.1 HTML

Webin tavallisin rakennuspalikka on HTML (HyperText Markup Language). HTML määrittelee Web-sisällön merkityksen ja rakenteen. [13] HTML on verkkosivujen rakentamiseen tarkoitettu merkintäkieli. HTML-dokumentit ovat tekstitiedostoja, jotka kertovat niitä lukeville Web-selaimille, mitä tietoja halutaan näyttää ja miten ne halutaan näyttää. [14]

HTML koostuu erityisistä elementeistä, joilla pyritään merkitsemään tekstiä, kuvia ja muuta sisältöä näytettäväksi Web-selaimessa [13]. Elementit koostuvat aloitustunnisteesta (eng. tag) ja useimmiten lopetustunnisteesta. Tunnistepari sulkee sisäänsä näytettävän sisällön, esimerkiksi tekstin. Ohjelmassa 1 on esimerkki HTML-elementistä. [14]

```
<h1>PDF-generointipalvelu</h1>
```

Ohjelma 1. *Esimerkki HTML-otsikkoelementistä, muokattu lähteestä [14].*

Aloitustunnistetta merkitään ”<” ja ”>” -merkeillä, joiden väliin tulee tunnisteen tyyppi, joka on ohjelman 1 tapauksessa otsikko 1 (eng. header 1). Lopetustunniste eroaa aloitustunnisteesta vinoviivalla, joka tulee ”<”-merkin jälkeen. [14]

2.5.2 CSS

CSS (Cascading Style Sheets) on mekanismi tyylien lisäämiseen Web-dokumentteihin [15]. CSS:llä voidaan kontrolloida muun muassa tekstien väriä tai fonttia, tekstikappaleiden etäisyyttä toisistaan, sarakkeiden näkyvyyttä ja kokoa, sivujen taustavärejä ja paljon muuta. Yleinen käyttötapaus CSS:lle on Web-sivujen tyylittely yhdessä HTML:än ja JavaScript:in kanssa. [16]

Ohjelmassa 2 on esimerkki CSS-säännöstä. Sääntö koostuu valitsimesta: *h1* ennen aaltosulkeita, selityksistä: riveistä aaltosulkeiden sisällä, ominaisuuksista: *font-size* ja *color* ja niiden arvoista: *24px* ja *red*. [16]

```
h1 {
    font-size: 24px;
    color: red;
}
```

Ohjelma 2. *CSS-sääntö HTML-elementille h1, muokattu lähteestä [16].*

Valitsin kertoo, mihin HTML-elementtiin säännöt kohdistuvat. Valitsimia on tyyppiin perustuvia, kuten ohjelmassa 2, sekä luokkaan että id:seen. Luokka ja id merkitään HTML-elementeille avainsanoilla *class* ja *id.* Selitys koostuu ominaisuudesta, kaksoispisteestä ja ominaisuuden arvosta. Ominaisuuksia voivat olla esimerkiksi fontin koko ja tekstin väri. Arvot annetaan ominaisuuksien mukaan. [16]

2.5.3 JavaScript

JavaScript on maailman yksi suosituimmista ja eniten käytetyistä ohjelmointikielistä. Lähestulkoon jokainen kaupallinen verkkosivu sisältää tänä päivänä JavaScript:iä. Se nähdään olennaisena osana web-kehitystä ja se on ollut varhaisessa vaiheessa myötävaikuttamassa Webin ja internetin nopeaa kasvua. [17]

JavaScript:in lisääminen Web-dokumenttiin on helppoa. Dokumenttiin lisätään vain *script*-tunnisteet, joiden sisään lisätään JavaScript-koodi. Tunnisteet ovat kuin HTML-elementtejä. Ohjelmassa 3 on esimerkki JavaScript-ohjelmakoodista. [18]

```
<script>
    function sum(number1, number2) {
          var sumResult = number1 + number2;
          return sumResult;
    }
</script>
```

Ohjelma 3. *Esimerkki JavaScript-ohjelmakoodista, muokattu lähteestä [18].*

Ohjelmassa 3 on funktio *sum*, joka ottaa vastaan kaksi numeroa ja laskee ne yhteen. Vastaus tallennetaan *sumResult*-muuttujaan, jota merkitään kieleen sisäänrakennetulla avainsanalla *var*. Funktion lopuksi vastaus palautetaan funktiota kutsuvalle. Funktiota voidaan kutsua muualla koodissa esimerkiksi *var result = sum(7, 7);,* jolloin *result*-muuttuja saa arvon 49. [18]

3. ARKKITEHTUURIEN ARVIOINTI

Ohjelmistokehityksessä arkkitehtuurin arviointi on hyvin tärkeää, sillä arvioinnissa pystytään havaitsemaan arkkitehtuurin puutteet ja mahdollisesti korjaamaan ne aikaisessa vaiheessa, jotta vältytään suurilta kustannuksilta. Arkkitehtuurin arviointia voidaan kuvata järjestelmän testauksena, ennen kuin järjestelmä on yksityiskohtaisesti suunniteltu. Arkkitehtuurin arvioinnissa tarkastellaan arkkitehtuurin laadullisten vaatimusten täyttymistä. [19, s. 222–225]

Arkkitehtuurien arviointimenetelmillä pyritään tuottamaan tietoa täyttyvätkö järjestelmän vaatimukset tietyllä arkkitehtuurilla, mikä on paras arkkitehtuuriratkaisu järjestelmälle ja esimerkiksi tuleeko järjestelmän ylläpito kalliiksi [19, s. 227]. Arviointimenetelmiä on useita. Tunnetuimmat niistä perustuvat skenaarioihin, kuten SAAM (Software Architecture Analysis Method), ATAM (Architecture Tradeoff Analysis Method), ALMA (Architecture-level Modifiability Analysis), FAAM (Family-architecture Assessment Method) ja ARID (Active Review of Intermediate Designs) [20]. Skenaarioihin pohjautuvia menetelmiä voidaan pitää kypsinä, sillä niitä on sovellettu ja validoitu useiden vuosien aikana [21, s. 638].

Arviointimenetelmien hyvistä puolista huolimatta, monet niistä vievät kohtuullisen paljon aikaa ja vaivaa toteuttaa. Esimerkiksi edellä mainitun SAAM-arvioinnin on suunniteltu kestävän yhden kokonaisen päivän, missä paikalla on useita sidosryhmiä. Keskikokoinen ATAM taas voi viedä 70 henkilötyöpäivää. [20]

Seuraavaksi tarkastellaan arviointimenetelmistä ATAM:ia ja DCAR:ia (Decision-Centric Architecture Reviews). ATAM käydään läpi tässä työssä, siitä syystä, että se on yksi tunnetuimmista arkkitehtuurin arviointimenetelmistä [20]. DCAR:ia taas siitä syystä, että se vie [22, s. 69–70] mukaan suhteellisen vähän resursseja ja on näin mahdollista toteuttaa kohdeyrityksessä.

3.1 ATAM-arviointimenetelmä

ATAM on arviointimenetelmä arkkitehtuuritason suunnitelmille. ATAM:in tarkoituksena on selvittää vastaako suunnitellun arkkitehtuurin toteutus sille annettuja vaatimuksia. Menetelmä ottaa huomioon useita eri laatuominaisuuksia, kuten muunneltavuuden, suorituskyvyn, luotettavuuden ja turvallisuuden. ATAM:in avulla tunnistetaan kohtia, jossa

kyseiset ominaisuudet aiheuttavat kompromisseja. Menetelmän on tarkoitus myös helpottaa sidosryhmien kommunikointia, selventää ja jalostaa arkkitehtuurin vaatimuksia ja tarjota runko meneillään olevan järjestelmän suunnitteluun ja analyysiin. [23]

ATAM koostuu neljästä osiosta, jotka voidaan jakaa kahteen vaiheeseen. Ensimmäisessä vaiheessa on esittely- ja analyysiosio. Toisessa vaiheessa on testaus- ja raportointiosio. Järjestelmän arkkitehtuuri, sen liiketoimintatavoitteet ja ATAM-menetelmä esitellään esittelyosiossa. Analyysiosiossa analysoidaan järjestelmän laatuvaatimuksia ja etsitään arkkitehtuuriratkaisuja, jotka ovat vaikuttaneet laatuominaisuuksiin. Analyysiosiossa kuvataan laatuvaatimuksiin liittyvät skenaariot, jotta voidaan tunnistaa arvioitavan arkkitehtuurin kriittiset kohdat. Testausosiossa tehdään muokkauksia skenaarioihin käyttäjien näkökulmasta ja analysoidaan jälleen arkkitehtuuriratkaisuja. Raportointiosio koostuu arvioinnin tuloksien esittämisestä. [19, s. 229]

Esittelyosion ensimmäinen askel on ATAM:in esittely sidosryhmille. Sidosryhmiin kuuluvat muun muassa asiakasedustajat, järjestelmän arkkitehti tai arkkitehtuurista vastaava tiimi, ylläpitäjät ja johtajat. Seuraavaksi ATAM:issa projektipäällikkö esittelee liiketoimintatavoitteet, jotka motivoivat kehitystyötä, ja ensisijaiset arkkitehtoniset ajurit, kuten esimerkiksi järjestelmän korkea saatavuus. Esittelyosion lopuksi esitellään arkkitehtuuri. Arkkitehdin tehtävänä on esitellä ehdotettu arkkitehtuuri ja miten se vastaa annettuihin liiketoimintatavoitteisiin. [24, s. 7]

Analyysiosiossa ensimmäiseksi tunnistetaan arkkitehtoniset lähestymistavat, jotka arkkitehti tunnistaa, mutta ei analysoi. Seuraavaksi kerätään laatuun vaikuttavat tekijät, jotka pitävät sisällään järjestelmän hyödyllisyyden, esimerkiksi suorituskyky ja turvallisuus. Nämä määritellään skenaarioiden tasolle ja priorisoidaan. Analyysiosion lopuksi analysoidaan arkkitehtoniset lähestymistavat, jotka vaikuttavat edellä mainittuihin tekijöihin. Lisäksi identifioidaan arkkitehtoniset riskit, herkkyyskohdat (eng. sensitivity points) sekä tasapainokohdat (eng. trade-off points). [24, s. 7] Herkkyyskohta on jonkin laatuominaisuuden kannalta kriittinen suunnittelupäätös, jota muuttamalla laatuominaisuus voi heikentyä. Tasapainokohta taas on useampaan kuin yhteen laatuominaisuuteen vaikuttava herkkyyskohta. [19, s. 230]

Testausosio alkaa skenaariojoukon kasvattamisella. Mallia otetaan analyysiosion skenaarioiden määrittelystä. Tässä vaiheessa lisätyt skenaariot priorisoidaan äänestämällä. Äänestyksessä on mukana kaikki sidosryhmät. Testausosion lopuksi analysoidaan jälleen arkkitehtonisia lähestymistapoja. Nyt kuitenkin testausosion korkeimmalle priorisoituja skenaarioita pidetään testitapauksina tähän mennessä määritettyjen arkkitehtonis-

ten lähestymistapojen analysoimiseksi. Testitapausskenaariot voivat vielä paljastaa arkkitehtonisia lähestymistapoja, riskejä, herkkyys- ja tasapainokohtia, jotka dokumentoidaan. [24, s. 8]

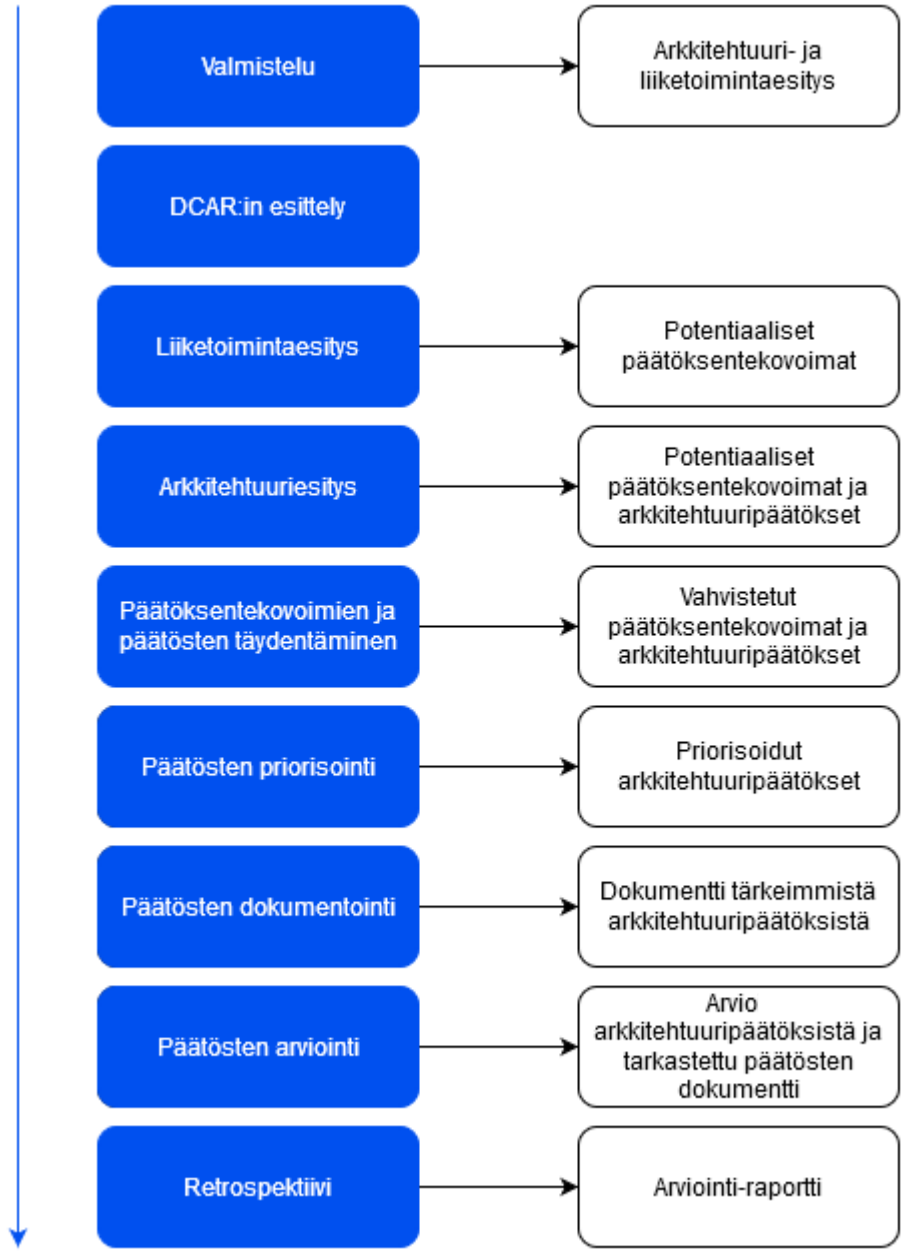
ATAM:in lopuksi onkin raportointiosio, jossa esitellään tulokset. ATAM-ryhmä esittää löydökset, jotka perustuvat ATAM:in aikana kerättyyn informaatioon. Ryhmä voi kirjoittaa raportin löydöksistä ja mahdollisista ”lieventämisstrategioista”. [24, s. 8]

3.2 DCAR-arviointimenetelmä

DCAR on päätöskeskeinen arkkitehtuurien arviointimenetelmä ja se tarkoittaa tässä yhteydessä sitä, että arviointi alkaa, kun sidosryhmät ovat valinneet joukon arkkitehtuuripäätöksiä analysoitavaksi projekti- ja yrityskohtaisten päätöksentekovoimien (eng. decision forces) suhteen. Arkkitehtuuripäätökset ovat päätöksiä, joita arkkitehdin on tehtävä koskien ohjelmistoa. Esimerkiksi, arkkitehti voi päättää olla käyttämättä avoimen lähdekoodin komponentteja ohjelmistossa lisensointia koskevista näkökohdista. Päätöstekovoimat ovat arkkitehtoniseen päätökseen vaikuttavia tekijöitä, jotka tulee ottaa huomioon. Tekijöitä, kuten riskit, poliittiset tai organisaatiolliset näkökohdat, henkilökohtaiset mieltymykset, kokemukset tai liiketoiminnan tavoitteet. [22, s. 69–71]

DCAR on tehty kahteen korkean tason vaatimukseen nojaten. Ensimmäinen vaatimus oli, että DCAR:in tulee olla kevyt käytetyn ajan ja resurssien suhteen. Toinen vaatimus oli, että DCAR:in tulee tukea päätöskeskeistä arkkitehtuurin arviointia, mikä mahdollistaa arvioinnin tekijöitä analysoimaan ja kirjaamaan arkkitehtuuripäätösten perusteet. Kokemukset ovat osoittaneet, että täysikokoinen DCAR-arviointi mukaan lukien raporttien teot, voidaan suorittaa alle viidessä henkilötyöpäivässä. Pienemmästä työmäärästä huolimatta sillä on tuotettu tyydyttäviä tuloksia sidosryhmille. [22, s. 69–70]

DCAR koostuu yhdeksästä eri vaiheesta. Vaiheet ovat kuvattuna kuvassa 1, jossa vaiheet on esitetty vasemmalla sinisellä ja vaiheista saatavat dokumentit oikealla valkoisella värillä. Ensimmäinen vaihe on DCAR-arvioinnin valmistelu. Valmisteluun kuuluu DCAR-arviointipäivän päivämäärän sopiminen sidosryhmien kanssa. Arvioitavan järjestelmän pääarkkitehti valmistelee esityksen, joka käsittelee korkean tason arkkitehtuurisuunnitelmia ja -vaatimuksia, eri lähestymistapoja ja teknologioita. Johtoa ja asiakkaita edustavat henkilöt valmistelevat myös esityksen vaiheessa yksi. Tämä on liiketoimintaesitys, johon kuuluu arvioitavan järjestelmän tai tuotteen, sen toimialueen, liiketoimintaympäristön ja liiketoiminnan vaatimusten ja rajoitusten esittely. Sekä arkkitehtuuri- että liiketoimintaesityksen kalvot annetaan jo nyt arviointiryhmälle, jotta he voivat opiskella materiaalin valmiiksi esityspäivää varten ja tunnistaa potentiaalisia arkkitehtuuripäätöksiä. [22, s. 71]



Kuva 1. *DCAR:in vaiheet, muokattu lähteestä [22].*

DCAR:in toinen vaihe tarkoittaa, että DCAR-sessiopäivä on alkanut. Sessiopäivän aluksi esitellään DCAR kaikille osallistujille. Esittelyyn kuuluu sessiopäivän aikataulusta ja DCAR:in metodeista, vaiheista ja osallistujien vastuista tiedottaminen. [22, s. 71–72]

Kolmas vaihe DCAR:issa on järjestelmän tai arkkitehtuurin esittely liiketoimintanäkökulmasta. Johtoa ja asiakkaita edustavat henkilöt pitävät ensimmäisessä vaiheessa valmistellun esityksensä. Tämä on lyhyt noin 15 minuutin esittely, jonka tarkoituksena on antaa kuva osallistujille liiketoimintatavoitteista ja liiketoimintaan liittyvistä päätöksistä, jotka tulee ottaa huomioon arkkitehtuurin arvioinnissa. Esityksen aikana arviointiryhmä panee

merkille potentiaalisia päätöksentekovoimia ja tiedustelee saadakseen täydennettyä jo ensimmäisessä vaiheessa havaitsemiaan. DCAR:in kolmannen vaiheen jälkeen johtoa ja asiakkaita edustavia henkilöitä ei tarvita. [22, s. 70–72]

Liiketoimintaesityksen jälkeen siirrytään DCAR:in neljänteen vaiheeseen. Neljännessä vaiheessa on pääarkkitehdin arkkitehtuuriesitys. Esityksen tarkoituksena on kuvata järjestelmän arkkitehtuuri osallistujille riittävällä tarkkuudella. Esityksen pituus on noin 45 minuuttia ja sen tulisi olla hyvin interaktiivinen. Kaikki osallistujat voivat kysyä kysymyksiä ja täydentää ymmärrystään järjestelmästä. Tässä vaiheessa arviointiryhmä tarkistaa, korjaa ja täydentää jo valmistelemaansa listaa tehdyistä arkkitehtuuripäätöksistä. [22, s. 72]

DCAR:in viidennessä vaiheessa selvennetään arkkitehtuuripäätöksiä ja niiden yhteyksiä toisiinsa. Näiden lisäksi täydennetään ja vahvistetaan näihin päätöksiin liittyviä päätöksentekovoimia. Viidennessä vaiheessa yksi arviointiryhmän jäsenistä muodostaa kaavion kuvaamaan päätösten välistä riippuvuutta. Alussa jokainen päätös, joka on kerätty edellisissä vaiheissa arvioijien toimesta, esitetään kaaviossa. Kun kaikki osallistujat ovat saaneet yhteisen ymmärryksen päätöksistä, päätösten väliset riippuvuudet merkitään kaavioon suorilla viivoilla. Riippuvuuksien tyyppi merkitään kahdella tavalla. Riippuvuuden tyyppi voi olla joko ”johtuu tästä” tai ”riippuu tästä”. Nämä merkinnät helpottavat arvioijia ja sidosryhmiä hahmottamaan jokaisen päätöksen tärkeyttä. Riippuvuudet ovat myös tärkeitä, jotta tiedetään mitkä päätökset otetaan huomioon päätöksentekovoimina, jonkun toisen päätöksen tarkastelussa. [22, s. 72–73]

Päätöksentekovoimien ja päätösten täydentämisen jälkeen kuudennessa vaiheessa valitaan sopiva määrä päätöksiä, jotka käsitellään jäljellä olevan DCAR session aikana. Tarkkaa määritelmää päätösten valintaperusteille ei ole, mutta esimerkiksi päätökset, joiden tiedetään sisältävän riskejä tai aiheuttavan kustannuksia tulisi olla priorisoidulla listalla. DCAR:issa päätöksiä priorisoidaan antamalla jokaiselle osallistujalle sata pistettä, jotka heidän tulee jakaa päätöksille. Kun pisteet on jaettu, valitaan sopiva määrä päätöksiä seuraaville vaiheille korkeimman pistemäärän mukaan. [22, s. 74]

Seitsemäs vaihe on päätösten dokumentointi. Järjestelmän pääarkkitehti ja muut osallistujat dokumentoivat priorisoidut päätökset. Jokainen osallistuja valitsee kaksi tai kolme päätöstä, joista hän on tietoinen. Nämä päätökset tulee dokumentoida niin, seuraavat asiat tulevat ilmi: ongelma, jonka päätös ratkaisee, arkkitehtuurillinen ratkaisu ongelmaan, vaihtoehtoiset ratkaisut sekä päätöksentekovoimat, jotka päätöksen arvioinnissa tulee ottaa huomioon. [22, s. 74]

Kahdeksannessa vaiheessa suoritetaan päätösten arviointi. Arviointi aloitetaan korkeimman prioriteetin omaavista päätöksistä. Osallistujat vuorollaan esittelevät dokumentoimansa päätökset, jotka osallistujat arvioivat. Arviointi tapahtuu yhdessä tarkastellen päätöstä ja tunnistamalla päätöksentekovoimia, jotka ovat kyseistä ratkaisua tai päätöstä vastaan. Päätöksentekovoimat ovat niitä, jotka on havaittu session aikana. Tässä vaiheessa käytetään myös kaaviota päätösten välisistä riippuvuuksista, jotta päätökset ymmärretään täysin. Päätöksiä ja kaaviota päivitetään jatkuvasti session aikana. Kahdeksannen vaiheen lopuksi päätetään äänestämällä ovatko päätökset hyviä vai pitääkö niitä pohtia uudelleen. Kahdeksannen vaiheen aikana arvioijat kirjaavat ylös potentiaalisia ongelmia tai riskejä, jotka käyvät ilmi keskustelussa. [22, s. 74]

Viimeinen vaihe eli yhdeksäs vaihe DCAR:issa on retrospektiivi. Arviointiryhmä kerää muistiinpanot ja syntyneet dokumentit. Arviointiryhmän tehtävänä onkin niiden perusteella muodostaa arviointiraportti kahden viikon aikana DCAR sessiosta. [22, s. 75]

4. KOHDEJÄRJESTELMÄ

Kohdejärjestelmä, johon työn tutkimus kohdistuu, on yrityksille myytävä Web-sovellus. Järjestelmää käytetään sekä SaaS-pohjaisesti että suoraan asennettuna asiakkaiden palvelimille. SaaS (Software as a Service) on ohjelmistojen toimitusmalli, jossa sovellus on myyjän hallussa ja myyjä sitoutuu toimittamaan toimintoja tilauksesta asiakkaalle [25]. Järjestelmä koostuu pääosin palvelinpään PHP-ohjelmistosta, PostgreSQL-tietokannasta ja selainpään React-ohjelmistosta. PHP on suosittu yleiskäyttöinen skriptikieli, joka soveltuu erityisesti Web-kehitykseen [26]. PostgreSQL on tehokas, avoimen lähdekoodin objekti-relaatiotietokantajärjestelmä [27]. React on JavaScript-kirjasto [28], johon tutustutaan paremmin seuraavassa luvussa.

Kohdejärjestelmän voidaan ajatella jakaantuvan kolmeen eri osaan. Järjestelmässä on perusnäkymä, ylläpito ja järjestelmäylläpito. Perusnäkymä on järjestelmän osa, joka näkyy kaikille käyttäjille, mutta sitäkin voidaan rajoittaa käyttäjäkohtaisesti. Käyttäjät viettävät suurimman osan ajastaan perusnäkymässä.

Ylläpito on yleensä asiakkaiden pääkäyttäjien tai kohdejärjestelmän vastuuhenkilöiden käytössä. Ylläpidossa asiakkaat pystyvät muokkaamaan organisaatiotietojaan ja järjestelmän teemaa, hallitsemaan käyttäjiä ja käyttäjien käyttöoikeuksia.

Järjestelmäylläpito on kohdejärjestelmän osa, johon vain tietyt järjestelmän pääkäyttäjät tai vastuuhenkilöt pääsevät. Järjestelmäylläpidossa hallitaan järjestelmän sisältöä ja käyttöoikeuksia, tarkastellaan lokeja ja määritellään järjestelmäasetuksia. Kaikki SaaS-asiakkaat eivät pääse järjestelmäylläpitoon.

Kohdejärjestelmän PDF-generointi on toteutettu palvelinpään PHP-ohjelmistossa. Apuna on käytetty WKHTMLTOPDF-kirjastoa ja Smarty-mallineita (eng. templates) [29]. WKHTMLTOPDF on avoimeen lähdekoodiin perustuva komentorivityökalu, joka ottaa vastaan HTML-formaattia ja muodostaa siitä PDF-tiedoston [30]. Smarty on mallinejärjestelmä (eng. template engine) PHP:lle, jonka tarkoituksena on erottaa Web-sovelluksen bisneslogiikka ja sisällön esitys toisistaan [31].

Smarty-mallineet toimivat kohdejärjestelmässä PDF-raporttipohjina. Mallineisiin syötetään PHP:llä dataa ja sen jälkeen ne ajetaan WKHTMLTOPDF:än läpi. WKHTMLTOPDF tuottaa PDF-tiedoston, joka lähetetään selaimelle. [29]

Kohdejärjestelmässä PDF-raporttien tulostaminen on miltei kaikkien käyttäjien käytettävissä, tätäkin on mahdollista kuitenkin käyttöoikeuksilla rajoittaa. Käyttäjät tulostavat ra-

portteja useaan eri tarkoitukseen. Raporttipohjat ovat usein kuitenkin samoja. Asiakkaiden organisaatioilla on valmiiksi määritellyt raporttipohjat tiettyihin tarkoituksiin ja näin ollen kaikki käyttäjät eivät voi vaihtaa käytettäviä raporttipohjia. Järjestelmä tarjoaa kuitenkin järjestelmän pääkäyttäjille tai vastuuhenkilöille, jotka pääsevät järjestelmäylläpitoon, mahdollisuuden valita käytettävät raporttipohjat muutamista vaihtoehdoista. Pääkäyttäjät eivät voi muokata raporttipohjia, vaan raporttipohjien teko ja niiden muokkaaminen on kohdeyrityksen vastuulla.

Kohdejärjestelmän PDF-raportit ovat tyyliltään hieman vanhanaikaisia ja kaipaisivat uutta muotoilua. Raportit tarvitsisivat useissa tapauksissa myös yksityiskohtaisempaa tietoa. Yksi nykyisen toteutuksen ongelmista on myös se, että osa asiakkaista, jotka käyttävät järjestelmää SaaS-pohjaisesti, eivät pysty itse määrittämään raporttipohjiaan vaan käyttävät SaaS-ympäristölle määriteltyjä yleisiä pohjia. Ongelma on myös, että raporttipohjien muokkaaminen suhteellisen vaikeaa ja työlästä. Raporttipohjien muokkaaminen onkin jäänyt kohdeyrityksen teknologiatiimin vastuulle niiden teknisyyden vuoksi.

5. UUSI PDF-GENEROINTIPALVELU

Kun lähdetään toteuttamaan uutta PDF-generointipalvelua, kerätään ja tunnistetaan vaatimukset palvelulle. Palvelun toteuttamiseen on useita ratkaisuja ja vaihtoehtoja, joihin on myös hyvä tutustua.

Tässä luvussa käydään ensin läpi vaatimukset, jotka on esitetty kohdeyrityksen ja kohdeyrityksen asiakkaiden toimesta uudelle PDF-generointipalvelulle. Sen jälkeen tutustutaan kolmeen eri vaihtoehtoon toteuttaa generointipalvelu.

5.1 Vaatimukset

Vaatimukset ovat kuvauksia siitä, kuinka ohjelmiston tulisi toimia. Vaatimuksella viitataan tyypillisesti uuden tai parannetun palvelun johonkin osa-alueeseen. [32, s. 3] IEEE:n (Institute of Electrical and Electronics Engineers) julkaisema standardi 619.12-1990 määrittelee vaatimuksen [33, s. 62]:

1. Edellytyksenä tai kykynä, jota käyttäjä tarvitsee ongelman ratkaisemiseksi tai tavoitteen saavuttamiseksi.
2. Edellytyksenä tai kykynä, jonka järjestelmän tai sen komponentin tulee täyttää täyttääkseen sopimuksen, standardin, spesifikaation tai muun virallisesti asetetun asiakirjan.
3. Dokumentoituna kuvauksena edellytyksestä tai kyvystä.

Kuten IEEE:en standardista voidaan päätellä, vaatimukset eivät sisällä ainoastaan käyttäjien tarpeita vaan myös niitä tarpeita, jotka syntyvät organisaatio-, hallinto- ja teollisuusstandardeista. Vaatimus on kokoelma täytettävistä tarpeista, jotka tulevat käyttäjältä ja muilta sidosryhmiltä. Ideaalitapauksessa vaatimukset ovat riippumattomia järjestelmän suunnittelusta. Ne osoittavat, mitä järjestelmän pitäisi tehdä eikä miten järjestelmä pitäisi tehdä. [32, s. 4]

Vaatimukset voidaan luokitella monella eri tavalla. Tässä työssä vaatimukset luokitellaan funktionaalisiin eli toiminnallisiin ja ei-funktionaalisiin eli laadullisiin. Toiminnalliset vaatimukset kertovat, mitä palvelu tekee, ja laadulliset ovat rajoitteita, kuten suorituskyky tai turvallisuus, sellaisille ratkaisutyypeille, jotka täyttävät toiminnalliset vaatimukset. [32, s. 4]

Vaatimuksia PDF-generointipalvelulle tarkastellaan tässä työssä yleisellä tasolla. Suurin osa toteutettavaa generointipalvelua koskevista vaatimuksista on järjestelmäspesifejä,

eikä niiden mainitseminen tässä työssä ole tarpeellista eikä suotavaa. Generointipalvelun vaatimukset on tunnistettu kohdeyrityksen asiakastiimin esittämistä käyttäjätarinoista, dokumenteista sekä nykyisen raportointipalvelun ominaisuuksista. Käyttäjätarinat liittyvät pitkälti PDF-raporttien sisältöön. Sisällöltä kaivataan muun muassa monimuotoisuutta ja tukea usealle eri lomakkeelle sekä kenttätyypille. Nykyiset PDF-raportit kaipaavat myös uutta modernia ilmettä.

5.1.1 Toiminnalliset vaatimukset

Kohdejärjestelmässä käytetään lomakkeita, joten mahdollisuus lomakkeiden tulostamiseen on olennainen vaatimus. Lomakkeiden tulostamisen lisäksi niiden tulisi olla näyttäviä ja selkeitä PDF-raportilla. Kohdejärjestelmässä on tällä hetkellä mahdollista muun muassa tulostaa yksittäisiä lomakkeita ja uudelta PDF-generointipalvelulta vaaditaan samaa.

PDF-generointipalvelussa tulee olla mahdollista tulostaa määrämuotoisia PDF-raportteja. Kohdeyrityksen asiakkailla on usein tarve tulostaa tietynlainen raportti, esimerkiksi vuositason raportointiin. Käyttäjien tulee kuitenkin tietyissä tilanteissa pystyä valitsemaan tulostettava määrämuotoinen raportti eri raporttipohjista. Määrämuotoisten raporttipohjien lisäksi generointipalvelun tulee tukea muokattavaa raporttipohjaa. Tiettyjen käyttäjien tulee pystyä määrittämään raportin esikatselussa, mitä tietoja raportille halutaan ja missä järjestyksessä.

Asiakasräätälöinti on yksi tärkeä osa kohdeyrityksen liiketoimintaa ja strategiaa, joten uuden PDF-generointipalvelun raporttipohjien tulee olla räätälöitävissä kohdeyrityksen toimesta. Raporttipohjien tulee olla helposti ymmärrettäviä ja muokattavia, jotta räätälöintiä voidaan tehdä kohtuullisen pienellä vaivalla ja ilman merkittävää ohjelmointiosaamista.

PDF-raporttien sisällön järjestäminen muun muassa kronologisesti on yksi toiminnallinen vaatimus, joka nousi käyttäjätarinoissa esiin. Raporteilla olevat tiedot halutaan järjestykseen esimerkiksi päivämäärän tai käyttäjän valinnan mukaan. Kohdejärjestelmän lomakelistoissa lomakkeet on mahdollista järjestää lomakkeella olevien kenttien mukaan. Tämä on toiminnallinen vaatimus myös uudelle PDF-generointipalvelulle.

PDF-generointipalvelun tulee tukea erilaisia sisältöelementtejä. Generointipalvelussa tulisi pystyä tulostamaan esimerkiksi lomakelistoja, kuvaajia ja kuvia. Näiden lisäksi gene-

rointipalvelun tulisi tukea käyttäjän lisäämiä muistiinpanoja sekä hyperlinkkejä. Hyperlinkkien tulisi olla avattavissa PDF-raportilta. Nykyinen kohdeyrityksen PDF-generointitapa ei tue muistiinpanoja eikä hyperlinkkejä.

Tiivistettynä tärkeimmät toiminnalliset vaatimukset:

- PDF-generointipalvelussa on mahdollisuus tulostaa määrämuotoinen raportti.
- PDF-generointipalvelun käyttäjä voi valita eri raporttipohjista tulostettavan raportin.
- PDF-generointipalvelun käyttäjällä on mahdollisuus muokata tulostettavaa raporttia.
- PDF-generointipalvelun käyttäjä voi järjestää tulostettavan raportin lomakelistoja.
- PDF-generointipalvelun käyttäjä voi tulostaa lomakelistoja, lomakkeita, kuvia, kuvaajia, muistiinpanoja ja hyperlinkkejä.

5.1.2 Laadulliset vaatimukset

Lähes yhtä tärkeitä toiminnallisten vaatimusten lisäksi ovat laadulliset vaatimukset. Kohdeyritykselle on tärkeää, että raportit ovat moderneja niin muotoilultaan kuin teknologialtaan.

Yksi tärkeimmistä laadullisista vaatimuksista kohdeyrityksen uudelle PDF-generointipalvelulle on suorituskyky. Suorituskykyä tarkastellaan tässä työssä skaalautuvuuden ja saatavuuden näkökulmasta. Mikäli PDF-raportin tulostajia on useita samanaikaisesti, generointipalvelun tulisi pystyä skaalautumaan ja olla koko ajan saatavilla, jotta se pystyy vastaamaan tarpeeseen. Skaalautuvuudella tarkoitetaan tässä yhteydessä sitä, että generointipalvelua voidaan monistaa, jotta yksi generointipalveluinstanssi ei ruuhkaudu ja näin pysty palvelemaan tulostajia. Saatavuudella tarkoitetaan tässä yhteydessä taas sitä, että generointipalvelun tulee olla jatkuvasti käytettävissä.

Laadullisia vaatimuksia ovat myös ulkonäköön liittyvät vaatimukset. Uuden PDF-raportin tulisi olla muotoilultaan, tekstin asettelultaan, grafiikoiltaan ja fonteiltaan nykyaikaista ja modernia. Raportilla olevan tilan tehokas hyödyntäminen on tärkeää. Raporttien tulee olla myös samanlaisia riippumatta käytettävästä laitteesta, käyttöjärjestelmästä tai selaimesta.

Kohdeyritys pitää myös suuressa arvossa tietoturvaa. Tietoturvan kannalta tässä työssä otetaan huomioon mitä dataa ja miten data liikkuu PDF-generointipalvelun ja esimerkiksi selaimen välillä.

5.2 Vaihtoehdot

Tässä luvussa käymme läpi eri vaihtoehtoja PDF-generointipalvelun toteuttamiseen. Tutustumme kolmeen eri vaihtoehtoon, joiden käytäntöön soveltuvuutta testaamme kuudennessa luvussa. Vaihtoehdot valittiin kirjoittajan tekemän esitutkimuksen, kohdeyrityksen vastuuohjaajan ja kohdeyrityksen kehittäjien mieltymyksien mukaan. Vaihtoehtoja otettiin tarkasteluun kolme, jotta saadaan tarpeeksi kattava kokonaiskuva eri vaihtoehdoista, ottaen kuitenkin huomioon työhön käytettävissä oleva aika, sillä vaihtoehtojen toteuttaminen käytännössä on työlästä.

5.2.1 Selainten tulostusominaisuus

Ensimmäisenä vaihtoehtona on tarkoitus muodostaa PDF-generointipalvelu käyttäen hyväksi kohdeyrityksessä jo käytössä olevaa React-kirjastoa. React on käyttöliittymien rakentamiseen tarkoitettu kirjasto JavaScript:ille [28]. React:in avulla voidaan koota kompleksisia käyttöliittymiä pienistä ja eristetyistä palasista koodia, joita kutsutaan komponenteiksi [34]. Kuvassa 2 on esimerkki React-komponentista.

```
1   const Options = (props) => (
2       <div>
3           <h1>{props.title}</h1>
4           <ul>
5               <li>Selainten tulostusominaisuus</li>
6               <li>React- ja Puppeteer-kirjasto</li>
7               <li>Handlebars- ja Puppeteer-kirjasto</li>
8           </ul>
9       </div>
10  );
11
12  // Esimerkki käyttötapauksesta:
13  <Options title='Toteutusvaihtoehdot'/>
```

Kuva 2. *Esimerkki React-komponentista, muokattu lähteestä [34].*

Kuvan 2 *Options*-komponentti ottaa vastaan parametrin *props*, joka on lyhenne englannin kielen sanasta properties eli ominaisuudet. Komponentti palauttaa kuvauksen siitä, mitä halutaan nähdä ruudulla. React käsittelee tämän kuvauksen ja näyttää tuloksen. [34] *Options*-komponentti palauttaa *h1*-tyyppisen otsikon ja kolmen alkion *ul*-tyyppisen listan *div*:in sisässä. Otsikko saa tekstinsä *props*-objektin kautta kuvan 2 rivillä 17.

PDF-raportti ja sen esikatselutila muodostetaan React-komponenteista. Esikatselu tarkoittaa sitä vaihetta PDF-generointipalvelussa, kun käyttäjä on valinnut tulostettavan ra-

porttipohjan ja hänelle aukeaa näkymä, jossa hän saa tarkastella ja muokattavan raporttipohjan kohdalla valita mitä raportille tulee. Esikatselussa käyttäjä näkee, minkälainen lopullinen raportti on ja hän voi valita haluaako hän tulostaa raportin.

React-komponentit pyritään tekemään yleis- ja helppokäyttöisiksi, jotta ne ovat ymmärrettävissä muidenkin kuin kohdeyrityksen teknologiatiimin toimesta. Toiminnallisissa vaatimuksissa vaaditut määrämuotoiset raporttipohjat ja muokattava raporttipohja tehdään React-komponenteista omiin tiedostoihinsa. Muokattava raporttipohja tehdään dynaamiseksi ja ehdolliseksi. Tämä tarkoittaa sitä, että käyttäjälle näytetään laaja raportti, jossa ovat kaikki mahdolliset sisältöelementit datoineen näkyvillä ja käyttäjä saa valita, mitä haluaa raportin sisältävän.

PDF-raportin esikatselutila toteutetaan React-komponenteilla. Esikatselutilan tulee tukea kahta eri näkymää. Ensimmäinen on määrämuotoisen raportin näkymä, jossa käyttäjälle näytetään vain tulostettava raportti. Toinen näkymä on muokattava näkymä, joka sisältää raportin kokonaisuudessaan ja sisällysluettelon, josta käyttäjä voi valita, mitkä sivut tulostetaan ja missä järjestyksessä. Molemmat näkymät sisältävät ”Tulosta PDF” -painikkeen, joka käynnistää selaimen oman tulostusprosessin.

PDF-raportin tulostamiseen ja tallentamiseen käytetään selainten omaa tulostusominaisuutta, joka käynnistetään React-koodissa *window.print()*-komennolla. Komento avaa tulostusvalintaikkunan, josta voidaan tulostaa sen hetkinen dokumentti. Selaimet, kuten Firefox, Chrome, Edge, Internet Explorer, Opera ja Safari tukevat verkkosivujen tulostamista. [35]

Tämä vaihtoehto voi olla ongelmallinen, sillä osa kohdeyrityksen asiakkaista käyttää vanhoja selaimia, jotka voivat toimia eri tavalla kuin uudet. Tulostusvalintaikkuna on erilainen riippuen käytettävästä laitteesta, käyttöjärjestelmästä tai selaimesta. Tulostusvalintaikkunassa on vaihtoehtoja muun muassa marginaalien ja taustavärien valintaan, joten on mahdollista, että jotkut käyttäjät saavat erilaisen raportin kuin toiset. Näin ollen ei voida olla varmoja, että laadullinen vaatimus alustariippumattomuudesta täyttyy.

Tämän vaihtoehdon etuna kohdeyritykselle on, että PDF-raportin muodostaminen ja generointi tapahtuu selainpäässä React-kirjaston avulla, joten uuden teknologian, kirjastojen tai kielen opettelua ei tarvita, sillä raportit voidaan muodostaa jo käytössä olevilla teknologioilla. Vaihtoehdon etu on myös, että PDF-generointipalvelun voidaan liittää nykyiseen kohdejärjestelmän selainohjelmistoon. Tällöin generointipalvelun ei tarvitse olla yhteydessä kuin kohdejärjestelmän palvelimeen, jolloin tietoturvan taso kohdejärjestelmässä pysyy ennallaan. Tämän vaihtoehdon käyttöönottaminen ei vaadi juuri laisinkaan konfigurointia vaan ainoastaan kohdejärjestelmän selainohjelmiston laajentamista.

5.2.2 React- ja Puppeteer-kirjasto

Toisena vaihtona on tarkoitus muodostaa PDF-generointipalvelu, jossa on selainohjelmiston lisäksi erillinen mikropalvelu, joka hoitaa PDF-generoimiseen liittyvät toiminnallisuudet. Mikropalvelu kehitetään Node.js:llä.

Node.js on asynkroninen, tapahtumiin (eng. events) pohjautuvat JavaScript-ajonaikainen-ympäristö, joka on suunniteltu skaalautuviin verkkosovelluksiin [36]. Node.js on avoimeen lähdekoodiin perustuva ja se toimii monella eri alustalla. Node.js ajaa Google Chromesta tuttua V8-nimistä JavaScript-moottoria selaimen ulkopuolella. V8 tarjoaa ajonaikaisen ympäristön, jossa se suorittaa JavaScript-koodia. [37]

Tavallisesti, JavaScript:iä voidaan ajaa vain selaimissa. Node.js mahdollistaa kuitenkin JavaScript:in ajamisen koneella. Tämä mahdollistaa sen, että Node.js:llä on pääsy tiedostojärjestelmään ja se voi kuunnella koneen verkkoliikennettä. Lisäksi se pystyy käsittelemään HTTP-pyyntöjä, kuten Web-palvelin. Lyhyesti sanottuna kaikki, mikä voidaan tehdä PHP:llä, voidaan tehdä nyt käyttämällä JavaScript:iä. [38]

Tässä vaihtoehdossa hyödynnetään samaa React:illa muodostettua selainohjelmistoa, kuin vaihtoehdossa yksi. Selainohjelmistoon tehdään muutamia muutoksia, jotta se toimisi mikropalvelun kanssa yhdessä. Esimerkiksi ”Tulosta PDF” -painikkeen painaminen ei kutsu selaimen tulostusominaisuutta vaan lähettää käyttäjältä saatuja tietoja mikropalvelulle.

Selainohjelmiston lisäksi React-kirjasto otetaan käyttöön mikropalvelussa. React:ia voi renderöidä nykyisin myös palvelimella [28] ja näin sekä selainohjelmisto että palvelinohjelmisto voivat jakaa samoja tiedostoja, React-komponentteja ja React-komponenteista tehtyjä raporttipohjia. Mikropalvelun tehtävänä on vastaanottaa selainohjelmiston lähettämä data, hakea muu PDF-raportille tarvittava data kohdeyrityksen palvelimelta ja muodostaa React:in, React-raporttipohjien ja Puppeteerin avulla PDF-raportti.

Puppeteer on ohjelmistokirjasto Node.js:lle ja se tarjoaa korkean tason rajapinnan Chromen tai Chromiumin hallintaan Chrome DevTools protokollan ylitse [39]. Chrome DevTools on protokolla, joka mahdollistaa eri työkaluille esimerkiksi Chromen tarkistamisen (eng. inspect) ja vianmäärittämisen [40]. Käytännössä Puppeteer:ia käytetään ruudunkaappausten, PDF:ien ja valmiiksi renderöidyn sisällön generointiin [39].

Kuten vaihtoehdossa yksi, tässäkin erityisen hyvää on React-kirjaston käyttäminen. React on kohdeyrityksessä käytössä oleva ohjelmistokirjasto, joten on helppoa suunnitella ja ylläpitää raporttipohjia. Hyvää vaihtoehdosta tekee myös se, että samoja kom-

ponentteja voidaan käyttää selainohjelmiston lisäksi myös mikropalvelussa, jossa varsinainen PDF-raportti generoidaan. Edelliseen vaihtoehtoon verraten, tässä vaihtoehdossa generoitavissa PDF-raporteissa ei pitäisi olla laite-, käyttöjärjestelmä tai selainkohtaisia eroja, sillä nyt generointi hoidetaan mikropalvelussa.

Tässä vaihtoehdossa on konfiguroitavaa mikropalvelussa ja sen käyttöönotossa. Tämä vaihtoehto tuottaa edelliseen vaihtoehtoon verrattuna myös enemmän työtä kohdeyrityksen kehittäjille. Kohdejärjestelmän nykyisen palvelinohjelmiston rinnalle täytyisi konfiguroida mikropalvelu, joka on yhteydessä sekä selainohjelmistoon että palvelinohjelmistoon. Asiakkaiden raportointiin liittyvä data kulkisi tässä vaihtoehdossa myös mikropalvelulle, joten kohdejärjestelmän tietoturvan taso ei pysyisi ennallaan.

Mikäli tämä vaihtoehto otettaisiin kohdeyrityksessä käyttöön, niin mikropalveluinstansseja tulisi olemaan vähintään yksi jokaista kohdeyrityksen asiakasta tai ympäristöä kohden. Mikropalvelun skaalaaminen on mahdollista, mikäli lisää instansseja vaaditaan. Monen asiakkaan ympäristössä tulisi olemaan kuorman mukaan tarvittava määrä mikropalveluinstansseja, jotta kaikkia asiakkaita pystytään palvelemaan.

5.2.3 Handlebars- ja Puppeteer-kirjasto

Kolmantena ja viimeisenä vaihtoehtona on tarkoitus muodostaa PDF-generointipalvelu mikropalvelulla, jossa selainohjelmisto toimii vain mikropalvelun tukena. Tässä vaihtoehdossa PDF-raporttipohjat muodostetaan React:in sijasta Handlebars:illa Node.js-mikropalvelussa.

Handlebars on kieli, joka perustuu mallineisiin. Mallineet ottavat vastaan syöttötietona objektin, joista Handlebars generoi HTML-tiedoston. Handlebars:in mallineissa käytetään HTML-elementtejä ja tupla-aaltosulkeilla ilmaistaan, minne syöttötiedot tulee sijoittaa. Kuvassa 3 on esimerkki Handlebars:in syntaksista ja syöttötietona tulevasta objektista. [41]

```
1   // Handlebars syntaksi
2   <p>{{firstname}} {{lastname}}</p>
3
4   // Syöttöobjekti
5   {
6       firstname: 'Petteri'
7       lastname: 'Ranto'
8   }
```

Kuva 3. *Handlebars syntaksi ja syöttöobjekti, muokattu lähteestä [41].*

Kuvan 3 rivillä kaksi aaltosulkeiden sisään tulee syöttöobjektin f*irstname:*n ja *lastname*:n arvot ja niiden väliin välilyönti. Oikeassa mallineessa kuvan syntaksi palauttaisi *p*-tyyppisen HTML-elementin työn kirjoittajan nimellä varustettuna.

Tässä vaihtoehdossa hyödynnetään jälleen samaa selainohjelmistoa kuin vaihtoehdossa yksi, mutta nyt vain osia siitä. Selainohjelmiston tehtävänä on lähettää käyttäjältä saadut tiedot mikropalvelulle, kuten vaihtoehdossa kaksi. Mikropalvelun tehtävänä on ottaa vastaan käyttäjältä tulleet tiedot, hakea kohdeyrityksen palvelimen rajapinnasta tarvittavat tiedot raportille ja muodostaa Handlebars-mallineiden ja Puppeteer:in avulla PDF-raportti.

PDF-generointipalvelun vaatimus määrämuotoisista dokumenteista pystytään täyttämään Handlebars:in avulla. Voimme luoda useita erilaisia PDF-mallineita Handlebars:in avulla. Tämä vaihtoehto tukee myös käyttäjän tekemiä valintoja, mutta hieman eri tavalla kuin aiemmissa vaihtoehdoissa. Vaatimusta siitä, että käyttäjä voisi muokata raporttia esikatselutilassa, ei ole järkevä täyttää. Esikatselutila olisi varmasti mahdollista toteuttaa Handlebars:in avulla. Kohdeyritys on kuitenkin siirtymässä pois yhden mallinekirjaston käytöstä käyttöliittymäpuolella, joten uuden sellaisen käyttöönotto ei ole varsin haluttu vaihtoehto. Esikatselutila voitaisiin toteuttaa myös selainohjelmistossa React:illa, kuten aiemmissa vaihtoehdoissa. React ottaisi vastaan käyttäjän valinnat ja välittäisi ne eteenpäin Node.js-mikropalvelulle, jossa tehtäisiin ehdollinen valinta Handlbars:in mallineiden elementtien tulostamisesta. Mikäli esikatselutila tehtäisiin React:illa, niin raporttipohjia täytyisi päivittää ja ylläpitää sekä React-tiedostoissa että Handlebars-tiedostoissa. Tämä lisäisi ylläpitokustannuksia ja hankaloittaisi kehittäjien työtä.

Koska vaihtoehto ei tue esikatselutilaa, toiminnallisia vaatimuksia muistiinpanojen ja lomakelistan järjestämisestä ei kyetä myöskään toteuttamaan. Toiminnallinen vaatimus muokattavasta raporttipohjasta toteutetaan selainohjelmistossa käyttäjälle aukeavalla valintaikkunalla, josta käyttäjä voi valita PDF-raportille tulostettavat elementit. Nämä valinnat lähetetään mikropalvelulle.

Nykyinen kohdeyrityksen PDF-generointipalvelu käyttää Smarty-mallineita PDF-raportin muodostamiseen, joten uusi mallinekirjasto ei ole suosituimpien vaihtoehtojen joukossa. Tämä vaihtoehto toisi myös uuden teknologian opeteltavaksi kohdeyrityksen kehittäjille. Handlebars vaikuttaa kuitenkin verkosta löydettävän materiaalin perusteella kypsältä vaihtoehdolta. Lisäksi PDF-raportin muodostaminen Handlebars:in avulla eroaa kahdesta edellisestä vaihtoehdosta, joten on järkevää tehdä myös käytännön kokeiluja sillä.

Kuten aiemmissa vaihtoehdoissa, tämänkin vaihtoehdon etuna on tulostettavien raporttien riippumattomuus käytettävästä laitteesta, käyttöjärjestelmästä tai selaimesta, sillä niiden generointi tapahtuu mikropalvelussa. Tämän vaihtoehdon hyvänä ja huonona puolena voidaan pitää raportin esikatselutilan puuttumista. Sen puuttuminen säästää PDF-generointipalvelun kehittämiseen käytettäviä työtunteja. Esikatselutila on kuitenkin yksi toiminnallisista vaatimuksista ja sen voisi ajatella tuottavan asiakkaille lisäarvoa.

Tämän vaihtoehdon käyttöönotto kohdeyrityksessä vaatisi konfigurointia ja palvelun tietoturvasta täytyisi huolehtia, kuten vaihtoehdossa kaksi. Vaihtoehdon kaksi tapaan mikropalveluinstansseja tulisi olemaan yksi jokaista kohdeyrityksen asiakasta tai ympäristöä kohden.

6. PDF-GENEROINTIPALVELUN TOTEUTUS

Tässä luvussa suoritetaan kokeiluja edellisessä luvussa esitetyillä vaihtoehdoilla. Vaihtoehtoja arvioidaan viidennessä luvussa esitettyjen toiminnallisten ja laadullisten vaatimusten perusteella.

6.1 Tyylitiedostojen toteutus

Tämän työn PDF-generointipalvelun eri toteutusvaihtoehdoissa yhteistä on CSS-tyylitiedostot. Seuraavaksi toteutetaan vaihtoehtojen tarvitsemat tyylitiedostot.

Luodaan CSS-tyylitiedostot *frontpage.css, preview.css* ja *pdf.css*. Nämä tiedostot sisältävät CSS-säännöt siihen, miltä PDF-raporttipohjan valinnan, PDF-raportin esikatselun ja tulosteen tulee näyttää käyttäjälle. Tiedostot lisätään myöhemmin jokaiseen vaihtoehtoon.

Aloitetaan tyylitiedostojen toteuttaminen *frontpage.css*-tiedostosta. Tämä tiedosto määrittelee kaikissa vaihtoehdoissa miltä PDF-raporttipohjan valinnan tulee näyttää. Tämä tiedosto ei ole työn, eikä PDF-generointipalvelun kannalta merkittävin, mutta se antaa kuvan siitä, miltä raporttipohjan valinta voisi näyttää kohdejärjestelmässä. Tyylitiedostoon määritellään yksinkertaisesti raporttipohjan valintaan liittyvien painikkeiden tyylit ja sijoittelut.

Jatketaan tyylitiedostojen toteuttamista *preview.css*-tiedostosta. PDF-raportin esikatselun tyylit määritellään tähän tiedostoon. *Preview.css*-tiedostoon määritellään esikatselutilalle muun muassa taustaväri, PDF-raportin sivujen koko, marginaalit ja täytteet (eng. paddings). Esikatselutila tehdään niin, että tavallisen yhtäjaksoisen sisällön sijasta jokainen raportin sivu näytetään erillään toisistaan ja näin raportti näyttää samalta kuin tulostettaessa.

Lopuksi toteutetaan *pdf.css*-tiedosto, johon määritellään tyylit, jotka halutaan astuvan voimaan tulostettaessa. PDF-raportille halutaan tulostettaessa vain raportin sisältö, eikä esimerkiksi esikatselussa näkyvää sisällysluetteloa. Tähän ratkaisuna käytetään CSS-mediakyselyitä (eng. media queries). Mediakyselyillä voidaan muuttaa sivua sen mukaan minkä tyyppistä laitetta käytetään tai minkä kokoisella näytöllä sivua käytetään [42].

Pdf.css-tiedostossa käytetään CSS:än *print*-mediakyselyä, joka on tarkoitettu erillisiin sivuihin jaetulle materiaaleille ja dokumenteille, joita katsotaan tulostuksen esikatseluti-

lassa [42]. *Print*-mediakyselyä käytetään ohjelman 4 mukaan, jossa kaarisulkeiden sisään tulevat säännöt, jotka astuvat voimaan, kun käytössä on tulostin tai selaimen esikatselutila.

```
@media print {}
```

Ohjelma 4. *Mediakysely tulostimille ja esikatselutilalle [42].*

Näillä kolmella tyylitiedostolla voidaan täyttää PDF-generointipalvelun laadulliset vaatimukset, jotka liittyvät raportin ulkonäköön. Tyylitiedostoilla voidaan määrittää raporttien muotoilut, tekstin asettelut, grafiikat ja fontit nykyaikaan.

6.2 Vaihtoehto 1

Ensimmäisenä vaihtoehtona toteutetaan PDF-generointipalvelu käyttäen hyväksi React-kirjastoa. Tässä vaihtoehdossa tarvitaan ainoastaan selainohjelmistoa PDF-raportin muodostamiseen ja generointiin.

Aloitetaan PDF-generointipalvelun muodostaminen React-projektin luonnilla. Projektin luonti ja React-sovelluksen alustus tapahtuu komentorivillä *npx create-react-app* -komennolla, joka luo React-kehitysympäristön [43]. Kehitysympäristö sisältää mallisovelluksen, joka voidaan käynnistää komentorivillä *npm start* -komennolla.

Seuraavaksi luodaan projektiin kansio nimeltä *pdf*, jonka alle luodaan neljä alakansiota *components*, *templates, helpers* ja *styles*. *Components*-kansioon tulevat PDF-raporttiin liittyvät React-komponentit, kuten esimerkiksi lomaketta kuvaava komponentti. Näistä komponenteista pyritään tekemään helppo- ja yleiskäyttöisiä, joten niiden tulee olla riittävällä abstraktiotasolla. *Templates*-kansioon tulevat *components*-kansion komponenteista muodostetut React-raporttipohjat. *Helpers*-kansioon tulevat tarvittavat apufunktiot. *Styles*-kansioon taas tulevat raporttipohjan valintaan, raportin esikatseluun ja generointiin liittyvät tyylitiedostot, jotka toteutettiin luvussa 6.1. *Pdf*-kansion juureen tulevat esikatseluun ja raporttipohjan valintaan liittyvät React-komponentit.

Luodaan *components*-kansion alle React-komponentit: *Charts, Cover, Document, Form, Forms, ForrnList, Links, Notes, Page, PageHeader, Text* ja *TextPage*. *Charts*-komponentin tarkoitus on generoida kuvaajia raportille sille syötetyn datan verran. *Cover*-komponentti tuottaa kansilehden sille syötetystä datasta. *Document* on raportin ylin komponentti, jonka sisään tulevat kaikki raportille tulevat komponentit. *Form* on lomakekomponentti, joka generoi lomakkeen raportille, sille syötetyn datan perusteella. *Forms* taas generoi *Form*-komponentteja sille annetun datan perusteella. *FormList* on lomakelista-komponentti. *Links* on linkkejä ja *Notes* on muistiinpanoja vastaanottava ja näyttävä

komponentti. *Page* on vähän kuin *Document*, mutta sen sisään tulevat kaikki raportin tietylle sivulle tulevat komponentit. *PageHeader* on komponentti, joka kuvaa raportin sivun otsikkoa, *Text* tavallista tekstiä ja *TextPage* tekstisivua.

Templates-kansion alle muodostetaan neljä React-raporttipohjaa edellä mainituista komponenteista. Raporttipohjat ovat *Simple* eli yksinkertainen, joka sisältää osan komponenteista, *Comprehensive* eli laaja, joka sisältää kaikki komponentit, *Form,* joka kuvaa yhtä lomaketta ja *Editable*, joka on muokattava raporttipohja. Kuvassa 4 on esimerkki React:illa tehdystä *Editable*-raporttipohjasta.

```
const EditableTemplate = ({contents, coverData, textPageData, forms, formListData, charts, notes, links}) => (
    <Document className="editable">
        {contents.map(content => {
            if (!content.selected) {
                return null;
            }
            if (content.type === 'cover') {
                return <Cover coverData={coverData}/>;
            }
            if (content.type === 'textPage') {
                return <TextPage textPageData={textPageData}/>;
            }
            if (content.type === 'forms') {
                return (
                    <Page>
                        <PageHeader>{content.title}</PageHeader>
                        <Forms forms={forms}/>
                    </Page>
                );
            }
            if (content.type === 'formList') {
                return (
                    <Page>
                        <PageHeader>{content.title}</PageHeader>
                        <FormList formListData={formListData}/>
                    </Page>
                );
            }
            if (content.type === 'charts') {
                return (
                    <Page>
                        <PageHeader>{content.title}</PageHeader>
                        <Charts charts={charts}/>
                        <Notes notes={notes}/>
                    </Page>
                );
            }
            if (content.type === 'links') {
                return (
                    <Page>
                        <PageHeader>{content.title}</PageHeader>
                        <Links links={links}/>
                    </Page>
                );
            }
            return null;
        })}
    </Document>
);
```

Kuva 4. *React:illa tehty Editable-raporttipohja.*

Kuvan 4 *Editable*-raporttipohja sisältää edellä mainitut komponentit, joita työn PDF-generointipalvelussa käytetään. Raporttipohja saa parametreinaan props-objektin, joka on purettu erillisiksi muuttujiksi. Käyttäjän raportin sisältövalinnat tulevat *contents*-listassa,

jonka mukaan *Editable* näyttää komponentteja. Muut *Editable*:lle props:eina tulevat muuttujat ovat dataa *Editable*:n sisältämille komponenteille.

Helpers-kansioon luodaan *printPDF*- ja *viewableValue*-funktiot. *PrintPDF*-funktio kutsuu selaimen tulostusominaisuutta ja *viewableValue* on kohdejärjestelmässä käytetty apufunktio, joka muokkaa PDF-raportille tulevien lomakkeiden kentät luettaviksi merkkijonoiksi. *Styles*-kansioon lisätään nyt kolme aiemmin toteutettua tyylitiedostoa, joista yksi on *frontpage.css* ja toinen *preview.css* ja kolmas *pdf.css*.

Pdf-kansion juureen tulevat React-komponentit *EditView*, *PrintButton*, *TableOfContents*, *TemplateCard*, *TemplateSelector* ja *TemplateViewer*. *TemplateSelector* on komponentti, joka sisältää raporttipohjien lukumäärän verran klikattavia *TemplateCard*-painikkeita. *TemplateSelector:*illa käyttäjä tekee valinnan haluamastaan raporttipohjasta. Käyttäjän tekemän valinnan mukaan näytetään raportti *TemplateViewer:*in avulla. *TemplateViewer*-komponentti sisältää *PrintButton*:in, jolla käyttäjä voi tulostaa kyseisen raportin. *EditView*-komponenttia taas käytetään, mikäli käyttäjä haluaa valmiin raporttipohjan sijaan valita muokattavan raporttipohjan. *EditView* sisältää *Editable*-raporttipohjan ja *TableOfContents*-komponentin, joka kuvaa sisällysluetteloa, jossa käyttäjä voi järjestellä tai ottaa pois sisältöä.

PDF-tiedoston generointi tapahtuu PDF-generointipalvelussa siten, että generointipalvelu hakee raportille tarvittavan datan kohdejärjestelmän palvelimelta. Raportille tuleva data syötetään käyttäjän valitsemaan React-raporttipohjaan, jolla näytetään raportti käyttäjälle. PDF-raportin tulostaminen tapahtuu selaimen avulla painamalla ”Tulosta PDF”-painiketta generointipalvelussa tai suoraan käyttämällä selaimen tulostusominaisuutta

Tässä vaihtoehdossa PDF-generointipalvelu on suhteellisen yksinkertainen ja suoraviivainen toteuttaa. Raporttipohjat ovat helposti ymmärrettävissä ja niitä on helppo muodostaa ja räätälöidä. Myöskään muokattava raporttipohja ei eroa haastavuudeltaan muista raporttipohjista.

Toiminnallisissa vaatimuksissa vaaditut sisältöelementit: lomakelistat, lomakkeet, kuvat, kuvaajat, muistiinpanot ja linkit toimivat odotetulla tavalla. Muistiinpanoja käyttäjä voi syöttää esikatselutilassa. Toiminnallinen vaatimus lomakelistojen järjestämisestä pystytään myös täyttämään. Käyttäjä pystyy esikatselutilassa järjestämään PDF-raportille tulevan lomakelistan sen kenttien mukaan.

Tämän vaihtoehdon PDF-generointipalvelun suorituskyky ei ole suoraan riippuvainen siitä, kuinka moni tulostaa PDF-raporttia yhtä aikaa, sillä generointipalvelu on jokaisen käyttäjän omalla selaimella ja palvelemassa vain kyseistä käyttäjää. Generointipalvelua

ei voida eikä sitä tarvitse skaalata. Mikäli tämä vaihtoehto otettaisiin käyttöön kohdeyrityksessä, generointipalvelu olisi saatavilla aina kun kohdejärjestelmäkin. Generointipalvelun suorituskykyyn voi vaikuttaa kohdejärjestelmän palvelimen kuormitus, kun generointipalvelu hakee sieltä tarvittavat datat raportille. Generointipalvelun suorituskykyyn kuitenkin vaikuttaa suurimmaksi osin käyttäjän käyttämä laite ja sen suoritusteho, joka voi joissain tapauksissa olla liian alhainen. Generointipalvelu toimi nopeasti kokeilussa pienillä datamäärillä. Kokeilussa raportti oli heti valmis, kun selaimen tulostusominaisuuskin.

Kuten viidennessä luvussa mainittiin, kohdejärjestelmän tietoturvan tulisi pysyä lähes ennallaan, mikäli tämän vaihtoehdon PDF-generointipalvelu otettaisiin kohdeyrityksessä käyttöön. Generointipalvelu liitettäisiin kohdejärjestelmän selainohjelmistoon.

Viidennessä luvussa tuotiin esille myös, että tämän vaihtoehdon mahdollinen ongelma on, että raporttien tulostaminen riippuu käytettävästä alustasta. Tätä vaihtoehtoa toteutettaessa ja testattaessa ei kuitenkaan ilmennyt eroja tulosteissa eri tietokoneilla tai selaimilla tulostettaessa.

6.3 Vaihtoehto 2

Toisena vaihtoehtona toteutetaan PDF-raportin varsinainen generointi selaimen sijasta Node.js-mikropalvelussa. PDF-generointipalvelu sisältää sekä selainohjelmiston että mikropalvelun. Tässä vaihtoehdossa käytetään jälleen hyväksi React-kirjastoa. Toinen merkittävä kirjasto on PDF-tiedoston generoinnissa käytettävä Puppeteer.

6.3.1 Selainohjelmisto

Vaihtoehdon kaksi selainohjelmisto on lähes identtinen edellisen vaihtoehdon kanssa. Komponentteihin ja raporttipohjiin täytyy tehdä hieman muutoksia, jotta käyttäjän valitsema raporttipohja tai sisältövalinnat muokattavalla raporttipohjalla, muistiinpanot ja tieto lomakelistan järjestämisestä saadaan välitettyä mikropalvelulle. Esimerkiksi ”Tulosta PDF” -napin toiminnallisuus ei kutsukaan selainten tulostusominaisuutta vaan lähettää kyseisen datan mikropalvelulle. Data lähetetään JSON-muodossa. JSON (JavaScript Object Notation) on kevyt tiedonsiirtoformaatti [44]. Kuvassa 4 on esimerkki selainohjelmiston lähettämästä datasta.

```
{
    "contents": [
        {
            "id": 1,
            "title": "Kansilehti",
            "type": "cover"
        },
        {
            "id": 4,
            "title": "Lomakelista",
            "type": "formList"
        }
    ],
    "template": null,
    "notes": "",
    "sortBy": [{"id": "Päivämäärä", "desc": true}]
}
```

Kuva 5. *Selainohjelmiston lähettämä data JSON-muodossa.*

Käyttäjän muokattavassa raporttipohjassa tehdyt valinnat kulkevat kuvan 5 *contents*-listassa. *Contents*-lista sisältää käyttäjän valitseman sisällön eli toisin sanoen, tiedon siitä, mitkä React-komponentit halutaan raportille ja missä järjestyksessä. Kuvan 5 *template*-muuttujassa kulkee taas tieto siitä, minkä raporttipohjan käyttäjä haluaa tulostettavan. *Notes*-listassa kulkee tieto käyttäjän raportille lisäämistä muistiinpanoista ja *sortBy*-listassa tieto minkä kenttien mukaan lomakelista on järjestettävä.

Kuvan 5 esimerkkidatasta nähdään, että käyttäjä on valinnut muokattavan raporttipohjan, sillä *template*-arvo on *null*. Raportille käyttäjä on halunnut vain kansilehden ja lomakelistan. Muistiinpanoja käyttäjä ei ole lisännyt. Lomakelistan käyttäjä haluaa järjestettävän päivämäärän mukaan laskevaan järjestykseen.

Kun selainohjelmisto on lähettänyt käyttäjältä saadun datan mikropalvelulle, se jää odottamaan vastausta. Odotuksen aikana käyttäjälle näytetään latausanimaatio ja vastauksen saapuessa näytetään ilmoitus latauksen onnistumisesta ja tiedosto ladataan automaattisesti käyttäjän koneelle. Mikäli vastauksen saaminen epäonnistuu, käyttäjälle näytetään virheilmoitus.

Koska selainohjelmiston toiminnallisuus on miltei sama edellisen vaihtoehdon kanssa, PDF-raportin tulostaminen onnistuu myös ilman mikropalvelua. Käyttäjä voisi siis suoraan käyttää selaimen tulostusominaisuutta ja tulostaa PDF-raportin. ”Tulosta PDF”-painike kuitenkin toimii eri tavalla.

6.3.2 Mikropalvelu

Seuraavaksi toteutetaan mikropalvelu, selainohjelmiston rinnalle. Edellisessä vaihtoehdossa kaikki PDF-raportointiin liittyvät toiminnallisuudet suoritettiin selaimessa. Tässä vaihtoehdossa PDF-generoinnin tekee mikropalvelu.

Mikropalvelusta tehdään Node.js Express-palvelin, joka on yhteydessä selaimeen verkon välityksellä. Express on minimalistinen ja joustava Web-sovelluskehys Node.js:lle [45]. Tässä vaihtoehdossa mikropalvelukin hyödyntää React-kirjastoa, joten hieman konfigurointia täytyy tehdä.

Aloitetaan mikropalvelun konfigurointi. Luodaan React-projekti, kuten aiemmissa vaihtoehdoissa *npx create-react-app pdfGenerationService* -komennolla, jossa *pdfGenerationService* on projektin nimi. Seuraavaksi asennetaan Express, Puppeteer ja muut tarvittavat apukirjastot komentorivillä projektikansiossa komennolla *npm install [kirjaston nimi]*. Asennuksien jälkeen luodaan projektiin kansio nimeltä *server* ja sen alle samanniminen JavaScript-tiedosto. *Server.js* vastaa mikropalvelun palvelinkoodista, esimerkiksi selainohjelmiston yhteydenottoihin vastaamisesta ja PDF-tiedoston generoinnista. Serverkansion alle luodaan toinen tiedosto nimeltä *index.js*. Tähän tiedostoon lisätään Babel-konfiguroinnit, sekä *server.js*-tiedoston tuonti. Babel on JavaScript-kääntäjä [46], jota tarvitsemme projektissa, sillä Node.js ei tunnista React:in käyttämää JSX-syntaksia eikä kaikkia käytettäviä moduuleja. [47]

Näiden konfigurointien jälkeen projektiin lisätään *components*-, *helpers*- ja *templates*-kansiot selainohjelmistosta. *Components*-kansion komponenteista *Charts-, FormList- ja Notes*-komponentteja täytyy muuttaa hieman. *Charts*-komponenttia täytyy muuttaa, sillä kuvaaja muodostetaan hieman eri tavalla mikropalvelussa kuin selainohjelmistossa. *FormList* ei taas tarvitse mikropalvelussa lomakelistan järjestämiseen ja *Notes* muistiinpanojen vastaanottamiseen liittyvää toiminnallisuutta.

Lopuksi kirjoitetaan mikropalvelun ydintoiminnallisuus *server.js*-tiedostoon. Toiminnallisuuden kirjoittaminen alkaa reitittämisestä (eng. routing) Express:illä. Reitittäminen tarkoittaa, että miten sovellus vastaa asiakkaan pyyntöihin tiettyyn päätepisteeseen. Päätepiste on URI tai polku ja käytettävä HTTP-metodi, esimerkiksi GET tai POST. Kuvassa 6 on yksinkertainen Express-palvelin, jossa *app* on Express-instanssi, rivillä 5 on juurireitti “/”, HTTP-metodi GET ja sen reitin käsittelijäfunktio, joka palauttaa rivillä 6 asiakkaalle ”Hello”-tekstin. [48]

```
1   const express = require('express');
2
3   const app = express();
4
5   app.get('/', (req, res) => {
6       res.send('Hello');
7   });
8
9   app.listen(3000);
```

Kuva 6. *Yksinkertainen Express-palvelin, muokattu lähteestä [49].*

Luodaan mikropalveluun *post*-metodilla oleva */printPDF*-päätepiste. Päätepiste odottaa yhteydenottoja selainohjelmistolta. Päätepisteelle luodaan käsittelijäfunktio, joka ottaa vastaan selainohjelmistolta tulleen kuvan 4 mukaisen datan. Käyttäjältä tulleen datan perusteella käsittelijäfunktio valitsee oikean React-raporttipohjan, johon se syöttää kohdeyrityksen palvelimelta hakemansa datan ja kuvaajista muodostetut base64-merkkijonot. Tämän jälkeen raporttipohjasta muodostetaan *ReactDomServer.renderToString*-funktion avulla staattinen HTML-tiedosto. HTML-tiedoston muodostamisen jälkeen käsittelijäfunktio kutsuu Puppeteer:ia. Puppeteer avaa Chromium:in headless-tilassa, avaa uuden sivun ja syöttää staattisen HTML-tiedoston ja tyylitiedostot Chromium:iin. Puppeteer ottaa PDF:än sivusta ja palauttaa sen muuttujaan, joka palautetaan selainohjelmistolle.

Mikropalvelu rakennetaan *npm run build* -komennolla, joka kääntää Babel:in avulla koodiin Node.js:än ymmärtämään muotoon. Tämän jälkeen mikropalvelu käynnistetään *node server/index.js* -komennolla.

Mikropalvelun käyttöönotto ja rakentaminen on helppoa. Node.js käyttää JavaScript:iä, joten mikropalvelun ohjelmointi ei eroa kieleltään selainohjelmistosta. Node.js:än konventioiden opetteleminen vie kuitenkin aikaa.

Mikropalvelussa React:in käyttäminen on lähestulkoon yhtä helppoa kuin selainohjelmistossa. Tämä ei eroa vaihtoehdosta yksi raporttipohjien tai raportilla tarvittavien komponenttien muodostuksessa. Komponenteista ainoastaan kuvaajien toteuttaminen vaatii erillisen kirjaston mikropalveluun, jonka käyttöönotto oli kuitenkin suhteellisen yksinkertaista. Myös kaikki sisältöelementit toimivat, jotka PDF-generointipalvelun toiminnallisissa vaatimuksissa vaadittiin. Lomakelistan järjestäminen onnistui myös selainohjelmistolta tulevan datan avulla.

Tämän vaihtoehdon PDF-generointipalvelun suorituskyky on riippuvainen käyttäjien määrästä sekä palvelimen suoritustehosta, mihin mikropalvelu asennetaan. Mikropalve-

lua voidaan kuitenkin skaalata, mikäli niin halutaan. Mikropalvelua skaalaamalla voitaisiin palvella useampaa käyttäjää yhtä aikaa. Työn kirjoittajan testauksessa generointipalvelun nopeus oli hitaampi kuin vaihtoehdossa yksi.

Tietoturvaan tulisi kiinnittää huomiota, mikäli tämä vaihtoehto otettaisiin kohdeyrityksessä käyttöön. Mikropalvelu olisi yhteydessä kohdejärjestelmän palvelimeen ja selainohjelmistoon. Raportille tuleva asiakasdata kulkisi näiden lisäksi myös mikropalvelulle.

6.4 Vaihtoehto 3

Viimeisenä vaihtoehtona PDF-generointipalvelu toteutetaan Node.js:llä ja Handlebars-kirjastolla. Vaihtoehto sisältää myös React:illa tehdyn selainohjelmiston, mutta se toimii vain mikropalvelun tukena.

6.4.1 Selainohjelmisto

Varsinaisesti erillistä selainohjelmistoa tämän vaihtoehdon PDF-generointipalvelussa ei tarvita. Todellisuudessa selainohjelmistona toimisi kohdejärjestelmän selainohjelmisto, jossa tapahtuisi esimerkiksi PDF-raporttipohjan valinta. Tässä vaihtoehdossa toteutetaan kuitenkin pienimuotoinen selainohjelmisto React:illa mikropalvelun tueksi, jotta voidaan mallintaa miltä generointipalvelu näyttäisi kohdejärjestelmässä. Selainohjelmistoon tulee sivu, jossa käyttäjä voi valita, minkälaisen PDF-raportin hän haluaa tulostaa, eri raporttipohjien väliltä. Tuki sille, että käyttäjä voi muokata tulostettavaa raporttia toteutetaan ruudulle aukeavalla ikkunalla, josta käyttäjä saa valita, mitä hän haluaa raporttiin tulostuvan. Tässä vaihtoehdossa raportin esikatselu ei ole tuettuna.

Luodaan jälleen React-projekti ja projektiin kansio nimeltä *pdf*, kuten vaihtoehdossa yksi. Lisätään projektiin edellisistä vaihtoehdoista tutut React-komponentit *TemplateSelector* ja *TemplateCard* sekä *printPDF*-apufunktio. Komponentit toimivat, kuten aiemmissa vaihtoehdoissa. Erona aiempiin vaihtoehtoihin on *printPDF*-funktio, johon lisäyksenä tulee ikkunan avautuminen, mikäli käyttäjä haluaa valita raportille tulevan sisällön. Tieto käyttäjän valitsemasta määrämuotoisesta raporttipohjasta tai käyttäjän valitsemasta sisällöstä lähetetään mikropalvelulle kuvan 5 tapaan, ilman muistiinpanoja ja tietoa lomakelistan järjestämisestä.

Selainohjelmistoon lisätään aiemmin määritelty *frontpage.css*-tyylitiedosto, joka määrittelee, miltä raporttipohjan valinnan tulee näyttää. Muita tyylitiedostoja tässä selainohjelmistossa ei tarvita.

Kuten voidaan huomata, tässä vaihtoehdossa selainohjelmisto vaatii vain muutaman React-komponentin, joten toteuttaminen on hyvin yksinkertaista. Tämän toiminnallisuuden tekeminen kohdeyrityksen sovellukseen voitaisiin tehdä suhteellisen nopealla aikataululla. Tämä vaihtoehto ei kuitenkaan tue raportin esikatselutilaa, muistiinpanoja tai lomakelistan järjestämistä, jotka ovat toiminnallisia vaatimuksia uudelle PDF-generointipalvelulle.

6.4.2 Mikropalvelu

Mikropalvelu toteutetaan Node.js:llä, kuten edellisessä vaihtoehdossa. Tässä vaihtoehdossa kuitenkin mikropalvelun luonti onnistuu mutkattomammin kuin vaihtoehdossa kaksi, sillä mikropalvelu ei vaadi React-kirjastoa eikä Babel-kääntäjää.

Ensimmäiseksi luodaan Node.js-projekti, joka tapahtuu yksinkertaisesti halutun kansiorakenteen luonnilla. Annetaan kansiolle kuvaava nimi, kuten *pdfGenerationService* ja luodaan sen alle kansiot *src*, *public, components* ja *templates.* Kansioon *src* tulee Node.js:än tarvitsemat lähdekoodit, *public*-kansioon tulee julkiset tiedostot, kuten kuvat ja tyylitiedostot, *components*-kansioon tulee PDF-raporttiin liittyvät komponentit ja *templates*-kansioon tulee komponenteista muodostetut PDF-raporttipohjat.

Tässä vaihtoehdossa komponentit ovat lähestulkoon samanlaisia kuin aiemmissa vaihtoehdoissa. React-komponenttien sijasta komponentit ovat Handlebars:in pieniä mallineita, joissa on tavallisen HTML:än lisäksi Handlebars:in aputoiminnallisuuksia. Luodaan samat komponentit, kuin aiemmissa vaihtoehdoissa.

Muodostetaan jälleen neljä raporttipohjaa *simple*, *comprehensive*, *form* ja *editable*. Raporttipohjat ovat nyt React-tiedostojen sijasta Handlebars-tiedostoja. Raporttipohjat tuottavat saman rakenteen, kuin aiemmissa vaihtoehdoissa. Kuvassa 7 on esimerkki Handlebars:illa tehdystä *editable*-raporttipohjasta.

```
{{#> document}}
    {{#each contents}}
        {{#if_eq this.type 'cover'}}
            {{> cover coverData=../coverData}}
        {{/if_eq}}
        {{#if_eq this.type 'textPage'}}
            {{> textPage textPageData=../textPageData}}
        {{/if_eq}}
        {{#if_eq this.type 'forms'}}
            {{#> page}}
                {{#> pageHeader}}{{this.title}}{{/pageHeader}}
                {{> forms forms=../forms}}
            {{/page}}
        {{/if_eq}}
        {{#if_eq this.type 'formList'}}
            {{#> page}}
                {{#> pageHeader}}{{this.title}}{{/pageHeader}}
                {{> formList formListData=../formListData}}
            {{/page}}
        {{/if_eq}}
        {{#if_eq this.type 'charts'}}
            {{#> page}}
                {{#> pageHeader}}{{this.title}}{{/pageHeader}}
                {{> charts charts=../charts}}
            {{/page}}
        {{/if_eq}}
        {{#if_eq this.type 'links'}}
            {{#> page}}
                {{#> pageHeader}}{{this.title}}{{/pageHeader}}
                {{> links links=../links}}
            {{/page}}
        {{/if_eq}}
    {{/each}}
{{/document}}
```

Kuva 7. *Handlebars:illa tehty editable-raporttipohja.*

Kuvan 7 *editable*-raporttipohja sisältää jälleen kaikki tässä työssä käytettävät komponentit. Raporttipohja muistuttaa React-raporttipohjaa, sillä se jakaantuu komponentteihin ja näyttää ne *contents*-listan perusteella. Raporttipohjalle syötetään sama data kuin React-raporttipohjalle. Data syötetään Handlebars:in tavoin syöttöobjektina.

Seuraavaksi luodaan *src*-kansion alle tiedosto nimeltä *server.js*, joka on samankaltainen edellisen vaihtoehdon *server.js*-tiedoston kanssa. *Server.js*-tiedostoon tehdään jälleen */printPDF*-päätepistettä kuunteleva käsittelijäfunktio. Erona edelliseen vaihtoehtoon on raporttipohjan muodostus. Raporttipohja valitaan käyttäjä datan perusteella ja se luetaan

templateHTML-muuttujaan. Tämä muuttuja käännetään Handlebars-mallineeksi Handlebars:in *handlebars.compile*-funktion avulla. Käännöksen jälkeen malline ajetaan syöttöobjektin kanssa ja paluuarvona saadaan HTML-tiedosto, joka annetaan eteenpäin Puppeteer:ille. [41]

Lopuksi lisätään mikropalveluun sen tarvitsemat tyylitiedostot. Lisätään mikropalveluun aiemmin luodut *preview.css*- ja *pdf.css*-tiedostot *public/css*-kansion alle.

Tämän vaihtoehdon etuna on, että mikropalvelun konfiguroiminen on yksinkertaista ja helppoa. Handlebars:in käyttöönotto tapahtuu nopeasti ja ainoastaan sen käyttäminen vaatii hieman opettelua. Handlebars kuitenkin tarjoaa verkkosivuillaan hyvän dokumentaation, joten tavallisiin pulmiin löytyy ratkaisu.

Komponentit ja komponenteista muodostetut raporttipohjat ovat hyvin samankaltaisia kuin edellisissä vaihtoehdoissa. Raporttipohjien ja komponenttien syntaksi ja apufunktioiden käyttäminen poikkeavat edellisistä vaihtoehdoista. Raporttipohjat ovat kuitenkin yksinkertaisia ja helppolukuisia. Raporttipohjia on helppo räätälöidä, mikäli muutokset ovat yksinkertaisia. Mikäli muutokset ovat sellaisia, jota Handlebars:in aputoiminnallisuudet eivät tue, on muutosten tekeminen vaikeampaa.

PDF-generointipalvelun suorituskyky on riippuvainen samoista asioista kuin vaihtoehdon kaksi. Eroja suorituskyvyssä voi olla Handlebars- ja React-kirjastojen välillä, mutta niiden suorituskyvyn vertaileminen ei kuulu työn tavoitteisiin. Generointipalvelu oli työn kirjoittajan testaamana myös hitaampi kuin vaihtoehto yksi.

Tietoturvan näkökulmasta tärkeän datan liikkumisen suhteen tämä vaihtoehto ei eroa juurikaan vaihtoehdosta kaksi. Mikäli tämä vaihtoehto otettaisiin kohdeyrityksessä käyttöön, asiakkaiden tietoja kulkisi kohdejärjestelmän palvelimen ja selainohjelmiston lisäksi myös mikropalveluun.

7. ARVIOINTI

Tässä luvussa käydään läpi kohdeyrityksessä tehty DCAR-arviointi, jonka kohteena oli PDF-generointipalvelun vaihtoehto kaksi. DCAR-arvioinnin lisäksi tämä luku käsittelee työn tuloksia, tuloksista johdettuja johtopäätöksiä ja jatkotutkimusmahdollisuuksia.

7.1 DCAR-arviointi

DCAR-arviointi suoritettiin vaihtoehdolle kaksi, joka oli käytännön kokeilujen ja kohdeyritykseltä saadun palautteen perusteella järkevä vaihtoehto ottaa tarkempaan tarkasteluun. Mallinekirjastoon pohjautuva arkkitehtuuri, kuten vaihtoehto kolme, on kohdeyritykselle jo tuttu, joten sen arvioiminen DCAR:illa koettiin vähemmän tärkeäksi kuin toisten. Vaihtoehdot yksi ja kaksi ovat taas selainohjelmistoltaan lähes samanlaisia, joten vaihtoehdon kaksi arvioiminen kattaa osin myös vaihtoehdon yksi.

7.1.1 Valmistelu

DCAR-arviointi toteutettiin perjantaina 17.07.2020. Tämän työn kirjoittaja toimi arvioinnin yleisenä järjestäjänä ja vetäjänä. Arviointiin kutsuttiin seitsemän henkilöä kohdeyrityksen teknologiatiimistä ja yksi henkilö asiakastiimistä.

Asiakastiimin henkilön oli määrä edustaa DCAR-arvioinnissa liiketoiminta- ja asiakasnäkökulmaa. Kun taas teknologiatiimin seitsemästä henkilöstä viiden oli määrä toimia arviointiryhmänä. Työn kirjoittajan ja yhden kohdeyrityksen frontend-kehittäjistä oli määrä edustaa järjestelmän pääarkkitehtia ja arkkitehtuurin suunnittelijoita.

Työn kirjoittaja valmisteli arkkitehtuuriesityksen DCAR-arviointia varten, joka jaettiin osallistujille saman viikon alussa, ennen arviointia. Valmisteltu arkkitehtuuriesitys noudatteli DCAR:in arkkitehtuuriesityspohjaa. Kirjoittaja myös perehdytti teknologiatiimin DCAR:iin ja jakoi siihen liittyvän aineiston kaikille osallistujille etukäteen. Asiakastiimiläinen sai aineiston mukana DCAR:in liiketoimintaesityspohjan omaa esitystään varten.

DCAR-arviointi sovittiin pidettävän etätapaamisena videoyhteydellä, sillä kohdeyrityksen Koronavirukseen liittyvä ohjeistus ei sallinut seitsemän ihmisen olevan samaan aikaan neuvottelutilassa. Etätapaamisesta huolimatta, arviointi suunniteltiin hyvin DCAR:in ohjeiden mukaiseksi. DCAR-materiaali, kuten aikataulu, ohjeet ja dokumentit päätösten ja päätöksentekovoimien kirjaamiseen jaettiin etukäteen verkossa, jotta ne olisivat kaikkien muokattavissa.

7.1.2 Eteneminen

Arviointi alkoi DCAR-menetelmän esittelyllä osallistujille. Arviointiin osallistui yksi asiakastiimin edustaja ja kuusi teknologiatiimin edustajaa mukaan lukien työn kirjoittaja. Lyhyen esittelyn jälkeen, siirryttiin liiketoimintaesitykseen, jossa asiakastiimin edustaja kertoi PDF-raportoinnista liiketoimintanäkökulmasta ja PDF-raportoinnin liiketoimintatavoitteista. Esityksen aikana syntyi keskustelua ja osallistujat esittivät täydentäviä kysymyksiä. Esitys kesti noin puoli tuntia, joka on hieman enemmän kuin DCAR:in ohjeissa on suositeltu. Päätöksentekovoimia kirjattiin tässä vaiheessa yhdessä yhteiseen tiedostoon. Kaikki osallistuivat päätöksentekovoimien tunnistamiseen mukaan lukien työn kirjoittaja eli pääarkkitehti. DCAR:issa tapana on, että arvioijat keräävät ja tunnistavat päätöksentekovoimia [3 s. 72], mutta vähäisen osallistujamäärän ja DCAR-kokemuksen takia, kaikki osallistuivat tähän. Päätöksentekovoimia tunnistettiin yhteensä 28 kappaletta. Yksi esimerkki tunnistetusta päätöksentekovoimasta on, että PDF-raportit ovat tärkeä osa kohdeyrityksen asiakkaalle tuottamaa arvoa.

Liiketoimintaesityksen jälkeen siirryttiin työn kirjoittajan arkkitehtuuriesitykseen. Esityksen aikana syntyi keskustelua arkkitehtuurista ja tehdyistä arkkitehtuuripäätöksistä. Esitys kesti noin 45 minuuttia, DCAR:in suosituksen mukaan. Muiden osallistujien tehtävänä oli esityksen aikana tunnistaa ja kirjata tehtyjä arkkitehtuuripäätöksiä yhteiseen tiedostoon. Arkkitehtuuripäätöksiä tunnistettiin yhteensä noin kymmenen kappaletta.

Arkkitehtuuriesityksen jälkeen pidettiin 45 minuutin tauko. Kun tauko oli ohi, siirryttiin arkkitehtuuripäätösten katselmointiin ja priorisointiin. Tunnistetut päätökset käytiin läpi, mutta niitä ei priorisoitu, niiden vähäisen määrän vuoksi. Kaikki päätökset otettiin mukaan seuraavien vaiheiden tarkasteluun.

Seuraava vaihe oli päätösten dokumentointi, jossa kaikki saivat noin kaksi päätöstä dokumentoitavaksi. Valinta tietyn päätöksen dokumentoijasta tehtiin päätöksen kontekstin perusteella, esimerkiksi frontend-kehittäjät dokumentoivat käyttöliittymään liittyvät päätökset. Päätökset dokumentoitiin jälleen yhteiseen tiedostoon.

Kun päätökset saatiin dokumentoitua, ne käytiin läpi osallistujien kesken. Jokaisesta päätöksestä keskusteltiin noin 10 minuuttia. Päätöksiä ja päätöksiin liittyviä riskejä arvioitiin sekä käytiin läpi päätösten hyviä ja huonoja puolia. Päätösten dokumentoituja tietoja päivitettiin yhdessä tämän vaiheen aikana. Tämä vaiheen aikana syntyi paljon keskustelua PDF-generointipalvelun toteuttamisesta.

Viimeisenä vaiheena oli retrospektiivi. Retrospektiivissä käytiin yleistä keskustelua tulevasta PDF-generointipalvelusta, sen vaatimuksista ja vaihtoehtoisista ratkaisuista. Keskustelu oli hyödyllistä ja siitä on varmasti apua kohdeyrityksen tulevassa projektissa. Keskustelua käytiin myös DCAR:ista ja arkkitehtuurien arvioinnista.

7.1.3 Tulokset

Tuloksena kohdeyrityksessä järjestetystä DCAR-arvioinnista syntyi lista tunnistetuista päätöksentekovoimista sekä arkkitehtuuripäätöksistä. Arkkitehtuuripäätöksistä tehtiin dokumentaatio, joissa oli dokumentoituna jokainen päätös, päätöksen ratkaisema ongelma, päätöksen vaihtoehtoiset ratkaisut, argumentit päätöksen puolesta ja vastaan. Tuloksena saatiin myös osallistujien yhteiset arviot tehdyille arkkitehtuuripäätöksille. Taulukossa 1 on tunnistetut päätökset ja osallistujien yhteinen arvio jokaiselle niistä. Arvio tehtiin äänestämällä, oliko päätös hyvä vai huono.

Taulukko 1. *Tunnistetut arkkitehtuuripäätökset*

Arkkitehtuuripäätös	Arvio
Järjestelmässä on valmiina erilaisia raporttipohjia.	Hyvä
Loppukäyttäjällä on mahdollisuus vaikuttaa PDF-raportissa näkyviin tietoihin.	Hyvä
Käyttöliittymässä näytetään sama data, kuin raportilla (esikatselutila).	Hyvä
Raporttipohjia voidaan käyttää, joko sellaisenaan tai noudattaa käyttäjän valintoja.	Hyvä
Raporttipohjien asettelu, muotoilu ja fontit tulee olla muokattavissa kohdeyrityksen toimesta.	Ei yhteistä arviota
Raporttipohjat valmiina mikropalvelulla.	Huono
Selain lähettää dataa mikropalvelulle.	Ei yhteistä arviota
Mikropalvelu ei ole yhteydessä tietokantaan.	Hyvä
Node.js:än, React:in ja Puppeteer:in käyttö.	Hyvä
Ympäristöä kohden vähintään yksi mikropalveluinstanssi	Hyvä

Osallistujat tunnistivat kymmenen taulukossa 1 nähtävää arkkitehtuuripäätöstä arvioinnin aikana. Tunnistettujen päätösten vähäinen määrä johtuu todennäköisesti esitetyn arkkitehtuurin pienestä koosta ja siitä, että kohdeyrityksellä ei ollut aiempaa kokemusta DCAR-arvioinnista. Päätöksentekovoimia sen sijaan tunnistettiin suhteellisen paljon, yhteensä 28 kappaletta. Osallistujat pitivät lähes kaikkia tunnistettuja arkkitehtuuripäätöksiä hyvinä. Huono arkkitehtuuripäätös osallistujien mukaan oli raporttipohjien sijainti mikropalvelulla.

Yhteistä arviota ei saatu arkkitehtuuripäätökselle, jossa raporttipohjien tyylien muokkaaminen on ainoastaan kohdeyrityksen vastuulla. Tämä aiheutti keskustelua osallistujissa. Toiset olivat sitä mieltä, että järjestelmässä pitää olla yhtenäiset ja kohdeyrityksen brändin mukaiset tyylit ja toiset olivat sitä mieltä, että asiakkaiden olisi hyvä saada päättää ne itse. Arviota ei saatu myös päätökselle, jossa selainohjelmisto lähettää dataa mikropalvelulle. Arviota tälle päätökselle ei saatu, sillä arkkitehtuuriesitys ei antanut riittävän hyvää kuvaa siitä mitä dataa lähetetään.

Arvioinnin tuloksena saatiin myös, että arvioitu arkkitehtuuri eli vaihtoehdon kaksi PDF-generointipalvelu ei ole sopiva suoraan sellaisenaan kohdeyrityksen PDF-generointipalveluksi. Muutoksia täytyisi tehdä raporttipohjien tallentamiseen ja selainohjelmistolta mikropalvelulle siirrettävään dataan. DCAR-arvioinnissa ilmeni, että raporttipohjien tulisi sijaita muualla, kuin mikropalvelulla, tietoturvasyistä. Raporttipohjat voivat sisältää asiakasdataa, jolloin eri asiakkaiden raporttipohjat eivät saisi sijaita samassa paikassa. Tähän ratkaisuna esitettiin, että raporttipohjat haettaisiin verkon yli jostain muualta. Siirrettävään dataan tulisi todennäköisesti vain toteutusyksityiskohtaisia muutoksia.

Yleisesti DCAR-arviointiin osallistuneet kohdeyrityksen työntekijät ja yrityksen teknologiajohtaja pitivät arvioinnista. DCAR-arviointi oli uusi menetelmä kohdeyritykselle, joten arvioinnissa DCAR-ohjeiden noudattamisessa hieman joustettiin. Esimerkiksi Koronaviruksesta johtuvaan etätyötilanteeseen nähden arviointi oli onnistunut.

7.2 Työn tulokset

Yleisesti työssä saadut tulokset olivat hyviä. PDF-generointipalvelulle asetetut toiminnalliset vaatimukset saatiin täytettyä kahdessa ensimmäisessä testatuista vaihtoehdoista. Kolmannessa vaihtoehdossa toiminnallisia vaatimuksia, koskien raportin esikatselutilaa, käyttäjän muistiinpanoja ja raportin lomakelistan järjestämistä ei saatu täytettyä.

Työssä tehdyt vaihtoehdot olivat keskenään hyvin samankaltaisia, eli samoja komponentteja ja PDF-generointipalvelun osia voitiin käyttää uudelleen. Vaihtoehto kolme erosi vaihtoehdoista eniten, sillä siinä käytetään PDF-raporttien muodostukseen eri kirjastoa kuin kahdessa ensimmäisessä vaihtoehdossa.

Työn tuloksena havaittiin, että vaihtoehdon kolme Handlebars:illa muodostetut PDF-raporttipohjat olivat yksinkertaisia ja helposti luettavia. Tämä oli yllättävä tulos, sillä kohdeyrityksessä Smarty-mallineista tehdyt raporttipohjat ovat monimutkaisia ja vaikealukuisia. Handlebars-raporttipohjien muodostaminen kuitenkin vaati opettelua. Handlebars:in aputoiminnallisuudet poikkeavat React-raporttipohjissa käytetyistä JavaScript-

funktioista, jotka olivat työn kirjoittajalle tutumpia. Raporttipohjien muodostaminen osoittautui haastavaksi tilanteissa, joissa jouduttiin käyttämään Handlebars:in tarjoamien aputoiminnallisuuksien sijasta itse tehtyjä funktioita. Itse tehdyistä funktioista aiheutuvat virheet ja niiden virheviestit olivat epäselviä, ja virheen syyn tunnistaminen kesti suhteellisen kauan.

Työssä tehtyjä PDF-generointipalveluita käyttämällä havaittiin, että parhaan suorituskyvyn tarjosi vaihtoehto yksi. Vaihtoehtoja testattiin käytännössä pienillä datamäärillä ja työn kirjoittajan tietokoneella. Generointipalveluja ajettiin kyseisellä koneella, joten testi ei anna täydellistä kuvaa, kuinka nopeita muut generointipalvelut olisivat, jos niiden mikropalveluja ajettaisiin palvelimilla. Vaihtoehdon yksi generointipalvelun suorituskykyä parantaa se, että sen ei tarvitse olla yhteydessä mikropalveluun eikä näin myöskään odottaa mikropalvelulta saatavaa PDF-raporttia. Generointipalvelun raportti on tulostettavissa heti, kun selain on siihen valmis. Tulostamisen suorituskyky on siis riippuvainen selaimesta eikä niinkään generointipalvelusta.

Vaihtoehdoista ensimmäinen, eli PDF-generointipalvelun muodostaminen selainohjelmistossa React-kirjaston avulla, oli yksinkertaisin ja helpoin toteuttaa. Kahdessa muussa vaihtoehdossa tehtiin selainohjelmiston lisäksi varsinainen PDF-generoinnin suorittava mikropalvelu. Mikropalveluiden muodostaminen ei ollut vaikeaa, sillä mikropalvelut käyttivät työn kirjoittajalle tuttua JavaScript:iä. Niiden muodostaminen aiheutti kuitenkin lisää työtä.

7.3 Johtopäätökset

Tässä työssä toteutetut PDF-generointipalvelut olivat pienimuotoisia eivätkä anna täyttä varmuutta siitä, mikä vaihtoehto toimisi parhaiten kohdejärjestelmässä. Mikään vaihtoehdoista sellaisenaan ei tule korvaamaan kohdeyrityksen vanhaa PDF-generointitapaa. Vaihtoehtojen toteuttaminen kuitenkin antaa tietoa teknologioiden soveltuvuudesta PDF-generointiin.

Kuudennessa luvussa tehdyistä kokeiluista ja työn tuloksista voidaan päätellä, että vaihtoehto yksi olisi paras ratkaisu korvaamaan kohdeyrityksen vanha tapa generoida PDF-raportteja. Vaihtoehto yksi käyttää kohdeyritykselle jo ennestään tuttuja teknologioita, joten generointipalvelun käyttöönotto olisi helpompaa kuin muissa vaihtoehdoissa. Sen toteuttaminen vie kohdeyritykseltä todennäköisesti vähemmän aikaa kuin muiden vaihtoehtojen. Vaihtoehto yksi toteutti myös kaikki toiminnalliset vaatimukset. Se on kuitenkin alustariippuvainen, joten kohdeyrityksen asiakkaat, jotka käyttävät esimerkiksi vanhoja selaimia, voisivat saada suunnitellusta poikkeavia raportteja.

Tietoturvan kannalta vaihtoehdon yksi käyttöönottaminen kohdejärjestelmässä ei pitäisi muuttaa kohdejärjestelmän tietoturvan tasoa juurikaan, sillä kohdejärjestelmän selainohjelmisto vain laajenisi. Tämän vuoksi vaihtoehto yksi olisi myös tietoturvan kannalta järkevä.

Työn tuloksista ei voida varmuudella päätellä, olisiko vaihtoehto yksi vaihtoehdoista nopein, jos se otettaisiin kohdeyrityksessä käyttöön. Vaihtoehdon suorituskyky riippuu paljon käsiteltävästä datamäärästä. On mahdollista, että kohdeyrityksen asiakkaat tulostavat PDF-raportin, joka sisältää dataa yli tuhannelta lomakkeelta. Tällaisen ison datamäärän käsittely selaimessa voi olla raskasta verrattuna palvelimella ajettavassa mikropalvelussa. Pienillä datamäärillä ja tavallisessa käytössä vaihtoehdon pitäisi kuitenkin olla hyvin nopea.

DCAR-arvioinnista voidaan tehdä johtopäätös, että vaihtoehto kaksi ei myöskään ole huono vaihtoehto korvaamaan kohdejärjestelmän vanhaa tapaa generoida PDF-raportteja. DCAR-arvioinnissa suurinta osaa arvioiduista arkkitehtuuripäätöksistä pidettiin hyvinä.

7.4 Jatkotutkimusmahdollisuudet

Yhtenä jatkotutkimusmahdollisuutena olisi toteuttaa DCAR-arviointi kaikille kolmelle vaihtoehdolle. DCAR-arviointi oli kohdeyritykselle mieleinen, ja sen käyttöä voisi jatkaa. DCAR-arviointimenetelmän lisäksi arkkitehtuureja voitaisiin arvioida myös toisella arviointimenetelmällä, esimerkiksi työssä läpi käydyllä ATAM:illa. Näin saataisiin kattavampi arviointi ja mahdollisesti erilaisia tuloksia kuin pelkästään DCAR:illa arvioitaessa.

Mikäli vaihtoehto kaksi tai kolme otettaisiin käyttöön kohdeyrityksessä, vaihtoehtojen mikropalvelut tulisivat todennäköisesti Docker-kontteihin. Kontit ovat standardoituja ohjelmistoyksikköjä, joiden avulla kehittäjät voivat eristää sovelluksensa sovelluksen ympäristöstä. Docker on käytännössä standardi konttien rakentamiseen ja jakamiseen. [50] Jatkotutkimuksena voitaisiin selvittää, kuinka mikropalvelut konfiguroidaan kontteihin ja olisiko tällöin jokin vaihtoehdoista helpompi toteuttaa kuin toiset.

Vaihtoehdon yksi tutkimusta voitaisiin jatkaa selvittämällä laitteet, käyttöjärjestelmät ja selaimet, joilla selainten tulostusominaisuus tuottaa PDF-raporteille määritellyt tyylit väärin. Kohdeyrityksessä voitaisiin käyttää tätä selvitystä asiakkaiden tiedottamisessa ja päivittää tuettujen alustojen listaa.

Toteutettujen PDF-generointipalvelujen suorituskyvyille voitaisiin tehdä jatkotutkimusta, jossa suorituskykyä mitattaisiin usealla käyttäjällä oikeassa ympäristössä. Vaihtoehdon yksi generointipalvelun suorituskykyyn käyttäjien määrän lisääminen ei suoraan vaikuta,

mutta tulostettavan datan lisääminen mahdollisesti. Kahta muuta vaihtoehtoa voitaisiin testata myös suurilla datamäärillä ja niiden mikropalveluja skaalaamalla.

PDF-generointipalveluiden tutkimusta voitaisiin jatkaa myös selvittämällä paras vaihtoehto tietoturvan kannalta. Jos PDF-generointipalvelun uudistamista harkittaisiin esimerkiksi vaihtoehtojen kaksi ja kolme välillä, olisi tarpeellista selvittää perusteellisesti tietoturvan taso kummassakin vaihtoehdossa.

8. YHTEENVETO

Tässä työssä toteutettiin kolme eri PDF-generointipalvelua kolmella eri vaihtoehdolla. Yksi niistä toteutettiin selainohjelmistolla, ja kahdessa muussa generointipalveluun kuului selainohjelmiston lisäksi myös itse toteutettu mikropalvelu. Työssä pyrittiin arvioimaan, mikä vaihtoehdoista olisi paras korvaamaan työn kohdejärjestelmän vanha PDF-generointitapa.

Aluksi työssä käytiin läpi työhön liittyviä käsitteitä, teknologioita ja muita taustatietoja. Näiden jälkeen tutustuttiin arkkitehtuurien arviointiin ja ATAM- ja DCAR-arviointimenetelmiin, joista jälkimmäinen valittiin käytettäväksi tässä työssä.

Taustatietojen jälkeen työssä tutustuttiin työn kohdejärjestelmään, sen PDF-generointiin ja generoinnin ongelmiin. Työssä saatiin selville, että kohdejärjestelmän tuottamat PDF-raportit ovat hieman vanhanaikaisia tyyleiltään ja tarvitsisivat modernisointia. Kohdeyritys kärsii kohdejärjestelmän vanhojen raporttipohjien työläästä ylläpidosta ja muokkaamisesta. Kohdejärjestelmässä kaikkien käyttäjien ei ole mahdollista valita tulostettavaa raporttipohjaa eikä kellään käyttäjällä mahdollisuutta muokata raporttipohjia. Muun muassa näihin ongelmiin työssä lähdettiin hakemaan ratkaisua.

Kohdejärjestelmän esittelyn jälkeen esitettiin vaatimukset ja vaihtoehdot uudelle PDF-generointipalvelulle. Toiminnalliset vaatimukset uudelle generointipalvelulle olivat, että generointipalvelun tulee tukea määrämuotoisia raportteja, eri raporttipohjien valintaa, muokattavaa raporttipohjaa, lomakelistojen järjestämistä ja eri sisältöelementtejä. Laadullisia vaatimuksia olivat suorituskyky, modernit tyylit ja tietoturva. Vaihtoehtojen kohdalla käytiin läpi, kuinka ne tultaisiin toteuttamaan tässä työssä.

Vaatimusten ja vaihtoehtojen jälkeen päästiin työn käytännön läheiseen osaan, jossa toteutettiin kolme PDF-generointipalvelun vaihtoehtoa. Käytännön kokeiluissa generointipalveluista tehtiin pienimuotoisia, joissa kuitenkin pyrittiin mallintamaan kohdejärjestelmässä käytettävää dataa. Generointipalvelut tuottivat esimerkiksi PDF-raportin, jossa oli mukana muun muassa lomakelistoja, kuvia ja kuvaajia.

Eri vaihtoehtojen toteuttamisesta siirryttiin työn arviointilukuun, jossa ensin käytiin läpi vaihtoehdolle kaksi tehty DCAR-arviointi kohdeyrityksessä, sen valmistelu, eteneminen ja tulokset. DCAR-arviointi osoittautui kohdeyrityksen osallistujien perusteella onnistuneeksi ja hyväksi menetelmäksi ottaa mahdollisesti jatkossakin käyttöön. DCAR-arvioinnista saatiin tuloksena, että vaihtoehto kaksi pienillä muutoksilla olisi varteenotettava

vaihtoehto kohdeyrityksen PDF-generointipalveluksi. DCAR-arvioinnin jälkeen käytiin läpi työn tulokset, johtopäätökset ja jatkotutkimusmahdollisuudet.

Työn tuloksena saatiin, että toiminnallisten ja laadullisten vaatimusten perusteella, vaihtoehto yksi olisi paras vaihtoehto korvaamaan kohdejärjestelmän vanhan PDF-generointitavan. Vaihtoehto yksi oli yksinkertainen ja vei vaihtoehdoista vähiten aikaa toteuttaa. Sen PDF-raporttipohjat ovat ymmärrettäviä ilman merkittävää teknistä koulutusta.

Työn tulokset ovat suuntaa antavia, mutta tuottavat kuitenkin arvokasta tietoa kohdeyrityksen projektille, jossa uudistetaan vanha tapa generoida PDF-raportteja. Työ antoi myös kokemusta työn kirjoittajalle ja kohdeyritykselle DCAR-arvioinnista.

LÄHTEET

[1] M. Hardy, L. Masinter, D. Markovic, D. Johnson, M. Bailey, The application/pdf Media Type, Internet Engineering Task Force (IETF), 2017. Saatavissa (viitattu 09.06.2020): https://tools.ietf.org/html/rfc8118

[2] G. Alonso, F. Casati, H. Kuno, V. Machiraju, Web Services: Concepts, Architectures and Applications, Springer, 2004. Saatavissa (viitattu 26.06.2020): https://link-springer-com.libproxy.tuni.fi/content/pdf/10.1007%2F978-3-662-10876-5.pdf

[3] UDDI Executive White Paper, Universal Description, Discovery and Integration (UDDI), 2001. Saatavissa (viitattu 26.06.2020): http://www.uddi.org/pubs/UDDI_Executive_White_Paper.pdf

[4] D. Austin, A. Barbir, C. Ferris, S. Garg, Web Services Architecture Requirements, The World Wide Web Consortium (W3C), 2004. Saatavissa (viitattu 26.06.2020): https://www.w3.org/TR/wsa-reqs/#id2604831

[5] D. Fensel, F. M. Facca, E. Simperl, I. Toma, Semantic Web Services, Springer, Berlin, Heidelberg, 2011. Saatavissa (viitattu 26.06.2020): https://andor.tuni.fi/permalink/358FIN_TAMPO/1j3mh4m/alma9910614488305973

[6] D. T. Sanders, J. A. Hamilton, R. A. MacDonald, Supporting A Service-Oriented Architecture, Society for Computer Simulation International, 2008, pp. 325–334. Saatavissa (viitattu 27.06.2020): https://dl-acm-org.libproxy.tuni.fi/doi/pdf/10.5555/1400549.1400595

[7] T. Datz, What You Need to Know About Service-Oriented Architecture, CIO, 2004. Saatavissa (viitattu 27.06.2020): https://www.cio.com/article/2439859/what-you-need-to-know-about-service-oriented-architecture.html

[8] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, Web Services Architecture, The World Wide Web Consortium (W3C), 2004. Saatavissa (viitattu 27.06.2020): https://www.w3.org/TR/ws-arch/#service_oriented_architecture

[9] M. Fowler, Microservices, martinFowler.com, 2014. Saatavissa (viitattu 26.06.2020): https://martinfowler.com/articles/microservices.html

[10] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, Microservices: Yesterday, Today, and Tomorrow, Springer, 2017, pp. 195–216. Saatavissa (viitattu 26.06.2020): https://link-springer-com.libproxy.tuni.fi/content/pdf/10.1007%2F978-3-319-67425-4.pdf

[11] S. Maffeis, Client-server Computing, Encyclopedia of Computer Science, John Wiley and Sons Ltd., GBR, 2003, pp. 215–218. Saatavissa (viitattu 26.06.2020): https://dl-acm-org.libproxy.tuni.fi/doi/pdf/10.5555/1074100.1074215

[12] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures, Dissertation, University of California, 2000. Saatavissa (viitattu

26.06.2020): https://www.ics.uci.edu/~fielding/pubs/dissertation/net_arch_styles.htm

[13] MDN web docs, HTML: Hypertext Markup Language, MDN contributors, 2020. Saatavissa (viitattu 02.08.2020): https://developer.mozilla.org/en-US/docs/Web/HTML

[14] T. A. Powell, HTML & CSS: The Complete Reference, Fifth Edition, McGraw-Hill/Osborne, 2010. Saatavissa (viitattu 31.07.2020): https://andor.tuni.fi/permalink/358FIN_TAMPO/176jdvt/cdi_safari_books_9780071496292

[15] B. Bos, What is CSS?, The World Wide Web Consortium (W3C), 2020. Saatavissa (viitattu 27.08.2020): https://www.w3.org/Style/CSS/

[16] I. Pouncey, R. York, Beginning CSS: Cascading Style Sheets for Web Design, Third Edition, Wrox Press, 2011. Saatavissa (viitattu 02.08.2020): https://andor.tuni.fi/permalink/358FIN_TAMPO/1j3mh4m/alma9910613618405973

[17] M. Baartse, Professional JavaScript, 2nd Edition, Apress, Berkeley, CA, USA, 2004. Saatavissa (viitattu 31.07.2020): https://andor.tuni.fi/permalink/358FIN_TAMPO/176jdvt/cdi_skillsoft_books24x7_bks00003609

[18] R. Ferguson, Beginning JavaScript: The Ultimate Guide to Modern JavaScript Development, Third Edition, Apress, 2019. Saatavissa (viitattu 31.07.2020): https://andor.tuni.fi/permalink/358FIN_TAMPO/176jdvt/cdi_askewsholts_vlebooks_9781484243954

[19] K. Koskimies, T. Mikkonen, Ohjelmistoarkkitehtuurit, Talentum, 2005. Saatavissa (viitattu 20.05.2020): http://www.cs.tut.fi/~ohar/OHAR2012_13-KIRJA-KoskimiesMikkonen.pdf

[20] V.-P. Eloranta, U. van Heesch, P. Avgeriou, N. Harrison, K. Koskimies, Chapter 6 - Lightweight Evaluation of Software Architecture Decisions, Morgan Kaufmann, Boston, 2014. Saatavissa (viitattu 08.07.2020): https://www.sciencedirect.com/topics/computer-science/architecture-evaluation

[21] L. Dobrica, E. Niemelä, A survey on software architecture analysis methods, IEEE Transactions on Software Engineering, vol. 28, no. 7, 2002, pp. 638–653. Saatavissa (viitattu 08.07.2020): https://andor.tuni.fi/permalink/358FIN_TAMPO/176jdvt/cdi_proquest_journals_195573177

[22] U. van Heesch, V.-P. Eloranta, P. Avgeriou, K. Koskimies and N. Harrison, Decision-Centric Architecture Reviews, IEEE Software, vol. 31, no. 1, 2014, pp. 69–76. Saatavissa (viitattu 08.07.2020): https://andor.tuni.fi/permalink/358FIN_TAMPO/176jdvt/cdi_gale_infotracacademiconefile_A356283867

[23] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, The architecture tradeoff analysis method, Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No.98EX193), Monterey, CA, USA, 1998, pp. 68–78. Saatavissa (viitattu 18.08.2020): https://ieeexplore.ieee.org/document/706657

[24] R. Kazman, M. H. Klein, P. C. Clements, ATAM: Method for Architecture Evaluation, Software Engineering Institute, 2000. Saatavissa (viitattu 08.07.2020):

https://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13706.pdf

[25] A. Butterfield, G. E. Ngondi, A. Kerr, SaaS (Software as a Service), A Dictionary of Computer Science, Oxford University Press, 2016. Saatavissa (viitattu 18.08.2020): https://www-oxfordreference-com.libproxy.tuni.fi/view/10.1093/acref/9780199688975.001.0001/acref-9780199688975-e-6709

[26] PHP, The PHP Group, 2020. Saatavissa (viitattu 26.08.2020): https://www.php.net/

[27] PostgreSQL: The World's Most Advanced Open Source Relational Database, The PostgreSQL Global Development Group, 2020. Saatavissa (viitattu 26.08.2020): https://www.postgresql.org/

[28] React, A JavaScript library for building user interfaces, Facebook Inc, 2020. Saatavissa (viitattu 04.08.2020): https://reactjs.org/

[29] Toimeksiantajayritys, 2020.

[30] J. Truelsen, A. Kulkarni, What is it?, WKHTMLTOPDF. Saatavissa (viitattu 26.08.2020): https://wkhtmltopdf.org/

[31] Smarty Template Engine, What is Smarty?, New Digital Group, Inc, 2020. Saatavissa (viitattu 26.08.2020): https://www.smarty.net/about_smarty

[32] A. Aurum, C. Wohlin, Engineering and Managing Software Requirements, Springer, Berlin, Heidelberg, 2005. Saatavissa (viitattu 28.06.2020): https://doi-org.libproxy.tuni.fi/10.1007/3-540-28244-0

[33] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, The Institute of Electrical and Electronics Engineers (IEEE), 1990, pp. 1–84. Saatavissa (viitattu 28.06.2020): https://andor.tuni.fi/permalink/358FIN_TAMPO/1j3mh4m/alma9910637289305973

[34] React, Tutorial: Intro to React, Facebook Inc, 2020. Saatavissa (viitattu 04.08.2020): https://reactjs.org/tutorial/tutorial.html

[35] MDN web docs, Window.print(), MDN contributors, 2020. Saatavissa (viitattu 15.08.2020): https://developer.mozilla.org/en-US/docs/Web/API/Window/print

[36] Node.js, About Node.js, OpenJS Foundation, 2020. Saatavissa (viitattu 1.7.2020): https://nodejs.org/en/about/#about-node-js

[37] F. Copes, The definitive Node.js handbook, freeCodeCamp, 2018. Saatavissa (viitattu 27.08.2020): https://www.freecodecamp.org/news/the-definitive-node-js-handbook-6912378afc6e/

[38] D. Shah, Node JS Guidebook, BPB Publications, 2018. Saatavissa (viitattu 27.08.2020): https://andor.tuni.fi/permalink/358FIN_TAMPO/1j3mh4m/alma9911114395305973

[39] Puppeteer, GitHub Inc, 2020. Saatavissa (viitattu 04.08.2020): https://github.com/puppeteer/puppeteer

[40] Chrome DevTools Protocol, 2020. Saatavissa (viitattu 04.08.2020): https://chromedevtools.github.io/devtools-protocol/

[41] Introduction, Handlebars, 2020. Saatavissa (viitattu 27.08.2020): https://handlebarsjs.com/guide/

[42] MDN web docs, Using media queries, MDN contributors, 2020. Saatavissa (viitattu 02.08.2020): https://developer.mozilla.org/en-US/docs/Web/CSS/Media_Queries/Using_media_queries

[43] React, Create a New React App, Facebook Inc, 2020. Saatavissa (viitattu 20.08.2020): https://reactjs.org/docs/create-a-new-react-app.html

[44] Introducing JSON, 2020. Saatavissa (viitattu 18.08.2020): https://www.json.org/json-en.html

[45] Express, Fast, unopinionated, minimalist web framework for Node.js, StrongLoop, IBM, and other expressjs.com contributors, 2017. Saatavissa (viitat-tu 27.07.2020): expressjs.com

[46] What is Babel?, Babel, 2020. Saatavissa (viitattu 27.07.2020): https://babeljs.io/docs/en/index.html

[47] F. Copes, Server Side Rendering with React, flaviocopes.com, 2018. Saatavissa (viitattu 29.08.2020): https://flaviocopes.com/react-server-side-rendering/

[48] Express, Basic Routing, StrongLoop, IBM, and other expressjs.com contributors, 2017. Saatavissa (viitattu 27.08.2020): http://expressjs.com/en/starter/basic-routing.html

[49] Express, Hello World example, StrongLoop, IBM, and other expressjs.com contributors, 2017. Saatavissa (viitattu 27.08.2020): http://expressjs.com/en/starter/hello-world.html

[50] Docker, Developers bring their ideas to life with Docker, Docker Inc, 2020. Saatavissa (viitattu 30.08.2020): https://www.docker.com/why-docker