Mikko Nousiainen

Verkkosovelluksen toteutus modernilla JavaScript-sovelluskehyksellä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

10.4.2017

Metropolia

Tekijä Otsikko Sivumäärä Aika	Mikko Nousiainen Verkkosovelluksen toteutus modernilla JavaScript-sovelluskehyksellä 33 sivua 10.4.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Yliopettaja Markku Karhu

Insinöörityön tarkoituksena oli tarkastella verkkosovelluksen toteuttamista modernilla JavaScript-sovelluskehyksellä ja hahmottaa modernien verkkosovellusteknologioiden etuja. Insinöörityöprosessi jakautui taustatutkimukseen, toteutetun prototyyppisovelluksen tarpeen ja toiminnollisuuksien määrittelemiseen ja toteutukseen.

Prototyyppisovellus toteutettiin itsenäisenä projektina. Sovelluksen toteuttamisen tarve määriteltiin tarkastelemalla tilaajaorganisaation päivittäistoimintaa. Prototyypin toiminnollisuudet valittiin vastaamaan erään tilaajaorganisaation prosessin kehittämistarpeisiin.

Taustatutkimuksessa perehdyttiin verkkosovellusteknologioiden historiaan ja nykyhetkeen. Teknologioiden kehitysvaiheisiin vertaamalla määriteltiin modernien verkkosovellusteknologioiden piirteitä, kuten tilan eriyttäminen näkymäkerroksesta.

Insinöörityössä toteutettiin prototyyppisovellus, jonka avulla voitiin todeta modernien verkkosovelluteknologioiden käytöstä koituvia etuja, kuten asynkronisuuden ja reaktiivisuuden toteuttamisen helppous. Prototyyppi koostuu selainpohjaisesta käyttöliittymästä ja palvelinsovelluksesta, jotka toteutettiin käyttäen Meteor- ja React-JavaScript-sovelluskehyksiä.

Insinöörityön konkreettinen tulos oli tilaajaorganisaation sisäiseen käyttöön toteutettu prototyyppisovellus. Prototyypissä ei ole mitään sellaisia sidonnaisuuksia mitkä estäisivät sen käytön myös muissa konteksteissa.

Insinöörityö antoi perspektiiviä siihen, miten verkkosovelluskehitys on muuttunut ja kehittynyt verkkosovellusten historian aikana. Käyttämällä moderneja verkkosovellusteknologioita voidaan saavuttaa pienemmillä kehityskustannuksilla nopeammin enemmän.

Avainsanat	Verkkosovellus, JavaScript, Meteor, React

Author	Mikko Nousiainen
Title	Developing a Web application using a modern JavaScript framework
Number of Pages	33 pages
Date	10 April 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Markku Karhu, Principal Lecturer

The aim of this final year project was to investigate using modern JavaScript frameworks in web application development, and to demonstrate the benefits of using modern web technologies. The process of this project consisted of research, defining the requirements of the implemented prototype, and the implementation of the prototype.

Research included an overview of the history and current state of web application technologies. Reflecting on these, features of modern web application technologies were defined, such as the separation of view and application state.

The requirements for the prototype were defined by observing the processes of the organization for which the prototype was to be implemented. The functionalities of the prototype were tailored to improve a certain process within the organization.

The prototype was developed as an individual effort. The implemented prototype demonstrates some of the benefits of using modern web application technologies, for example the ease of implementing asynchronicity and reactivity. The prototype consists of a web browser based front-end user interface and back-end server application, implemented using the Meteor and React JavaScript frameworks.

The concrete result of the final year project is the implemented prototype. The prototype demonstrated some of the benefits of using modern web application technologies, for example that comparatively more features can be implemented with less development effort.

Keywords	Web application, JavaScript, Meteor, React

Sisällys

1 Johdanto 1

2 Verkkosovellusteknologiat 2

2.1 Verkkosovellusteknologioiden kehitysvaiheet 2
2.2 JavaScript-ohjelmointikieli 6
2.3 JavaScript-sovelluskehykset 7
2.4 LAMP-pinosta MEAN-pinoon 8
2.5 Meteor-sovelluskehys 9
2.6 React-kirjasto 11
2.7 Moderni JavaScript-sovelluskehys 12

3 Tilaajaorganisaation viikkopalaveriprosessi 13

3.1 Yrityksen toimintamallit 13
3.2 Viikkopalaveriprosessi 15

4 Viikkopalaverisovellus 21

4.1 Viikkopalaverisovelluksen toiminnollinen ja tekninen määrittely 21
4.2 MVP-version ominaisuudet 24
4.3 Viikkopalaverisovelluksen toteutus 26

5 Yhteenveto ja pohdinta 30

Lähteet 32

1 Johdanto

Verkkosovellusten lyhyen historian aikana sovellusteknologiat ovat kehittyneet nopeasti. Etenkin viimeisen kymmenen vuoden aikana kehityksen tahti on kiihtynyt JavaScript-ohjelmointikielen ja sovelluskirjastojen kehityksen myötä. Nykyiset sovellusteknologiat mahdollistavat sellaisten verkkosovellusten toteutuksen, jotka olivat viime vuosikymmenen vastaavilla kehittämistyökaluilla erittäin työläitä tehdä tai jopa mahdotonta toteuttaa. Verkkosovellusten tekniset rajoitteet ovat tämän kehityksen myötä niin vähäisiä ja edut työpöytäsovelluksiin verrattuna merkittäviä, että monet sovellukset, jotka aiemmin saattoivat olla saatavilla vain perinteisinä työpöytäsovelluksina, ovat nykyään siirtyneet kokonaan verkkosovelluksiksi.

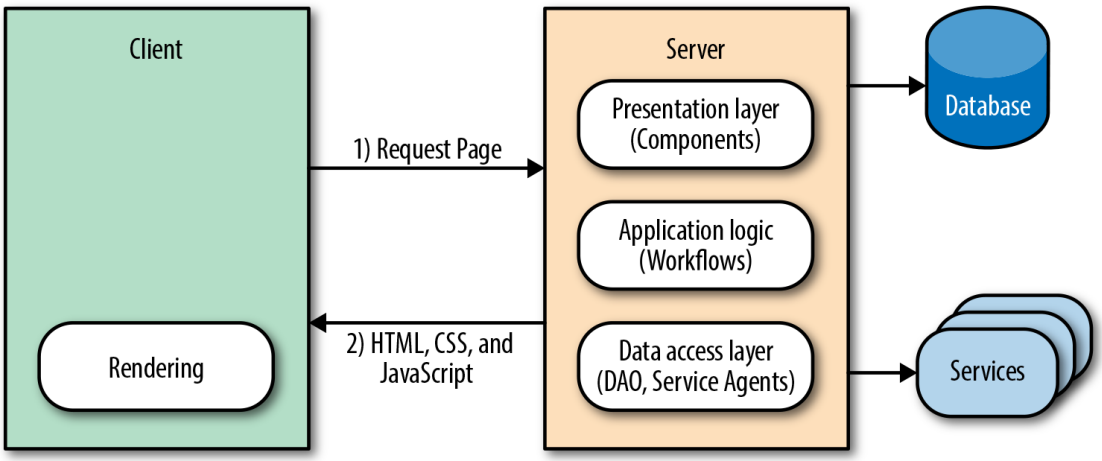
Insinöörityön tarkoituksena on määritellä moderni JavaScript-sovelluskehys ja kuvata verkkosovelluksen toteutusta käyttäen määritelmän mukaista sovelluskehystä. Kuvauksen pohjaksi otan verkkosovelluksen prototyypin, jonka suunnittelen ja toteutan työnantajalleni Fraktio Oy:lle. Verkkosovellusteknologioita käsitellään luvussa 2 ja tätä tietoa sovelletaan luvussa 4.

Fraktio on vuonna 2012 perustettu verkkopalveluihin erikoistunut ohjelmistoalan konsultointiyritys. Nykyään sen palveluksessa on 26 työntekijää. Yritys on yksi Suomen parhaista työpaikoista: se sijoittui vuosina 2016 ja 2017 Great Place To Work -tutkimuksessa pienten organisaatioiden sarjassa sijoille kolme ja neljä [1]. Fraktio on yrityksenä monella tavalla poikkeuksellinen verrattuna muihin alan yrityksiin, esimerkiksi organisaatiorakenteen ja toimintatapojen suhteen. Fraktion keskeisiin arvoihin kuuluu kokeilukulttuuri. Toimintamalleja ja prosesseja tarkkaillaan ja kyseenalaistetaan jatkuvasti, ja tavoitteena on kehittyä organisaationa jatkuvasti paremmaksi. Yksi keskeinen prosessi on viikkopalaveri. Tässä insinöörityössä toteutettava verkkosovelluksen prototyyppi pyrkii selvittämään, kehittäisikö viikkopalaveriprosessille räätälöity verkkosovellus prosessia paremmaksi.

2 Verkkosovellusteknologiat

2.1 Verkkosovellusteknologioiden kehitysvaiheet

Perinteinen verkkosovellus on rakenteeltaan kolmitasoinen asiakas-palvelinsovellus, jossa tasot ovat esitystaso, sovellustaso ja tietovarasto [2]. Esitystaso, eli sovelluksen käyttöliittymä, on käyttäjän selaimessa ja toteutettu HTML-, CSS- ja JavaScript-selainteknologioilla. Sovellustaso, eli sovelluslogiikka, on palvelinpäässä ja toteutettu esimerkiksi PHP-, Java- tai JavaScript-ohjelmointikielellä. Tietovarasto on myös palvelinpäässä, ja se on esimerkiksi relaatiotietokanta, kuten MariaDB, tai dokumenttitietokanta, kuten MongoDB. Kuvassa 1 on havainnollistettu kolmitasoisen verkkosovelluksen tasorakennetta.



Kuva 1. Kolmitasoisen asiakas-palvelinsovelluksen tasorakenne [3].

Esitystaso ottaa vastaan ja käsittelee käyttäjän syötteitä. Niiden perusteella sovelluksen esitystaso lähettää pyyntöjä sovelluksen sovellustasolle. Sovellustaso ottaa vastaan pyyntöjä ja pyynnön sisällöstä riippuen suorittaa jonkin osa-alueen sovelluslogiikasta. Tämän suorituksen aikana sovellustaso saattaa käsitellä tietovarastossa olevia tietoja. Suorituksen päätteeksi sovellustaso lähettää vastauksen esitystasolle. Esitystaso ottaa vastaan vastauksen ja päivittää käyttöliittymän tarvittavilta osin vastauksen sisällön perusteella.

Esitystaso ei ole suorassa yhteydessä tietovarastoon eikä sisällä sovelluslogiikkaa muutoin kuin käyttöliittymän toteutuksen kannalta välttämättömän logiikan. Sovellusta-

so saattaa laatia käyttöliittymän täydellisen kuvauksen, kuten kokonaisen HTML-dokumentin, mutta se ei sisällä tarvittavia ominaisuuksia tämän käyttöliittymän esittämiseen käyttäjälle. Tietovarasto on sovelluksesta riippumaton, eikä se toteuta mitään osa-aluetta sovelluslogiikasta.

Verkkosovelluksen tasojako ei ole aina näin yksiselitteistä. Moderneissa verkkosovelluksissa sovelluslogiikka saattaa olla lähes kokonaan käyttäjäpäässä ja palvelinpää on vain rajapinta tietovarastolle. Palvelinpään sovellustaso saattaa toisaalta olla monitasoinen. Sovelluslogiikka voi olla palvelukeskeisen arkkitehtuurin mukaisesti hajautettu useaan pienempään itsenäiseen sovelluskokonaisuuteen, jotka integroitumalla toisiinsa tarjoamiensa rajapintojen kautta toteuttavat käyttäjälle yhtenäisenä sovelluksena näyttäytyvän kokonaisuuden.

On lukuisia etuja, jos asiakas-palvelinarkkitehtuuriin perustuva sovellus toteutetaan verkkosovelluksena verrattuna sovelluksen toteuttamiseen käyttöjärjestelmä- tai suoritusalustakohtaisena natiivisovelluksena.

Verkkoselain suoritusympäristönä on alustariippumaton, eli sama sovellus toimii kaikissa päätelaitteissa. Mahdolliset verkkoselain- tai päätelaitekohtaiset sovellukseen toteutettavat eroavaisuudet liittyvät tyypillisesti sovelluksen käytettävyyden optimointiin riippuen päätelaitteen osoitinlaitetyypistä tai näytön koosta. Nämä eroavaisuudet ovat tyypillisesti huomattavasti yksinkertaisempia toteuttaa kuin alustakohtaisten natiivisovellusten toteuttaminen usealle alustalle.

Asiakas-palvelinarkkitehtuuria käytetään muun muassa sovelluksissa, joissa käyttäjän päätelaitteessa suoritettava sovellus käyttää palvelinpäässä olevaa keskitettyä tietovarastoa tai palvelin välittää dataa käyttäjältä toiselle. Yleistäen, käyttäjien ja palvelimen välillä kulkee tietoliikennettä. Monissa moderneissa sovelluksissa ylipäätänsä on graafinen käyttöliittymä. Kaikissa verkkoselaimissa on toteutettuna HTTP-prokolla, HTML, CSS ja JavaScript, joten verkkosovellusta toteuttaessa tietoliikenteen ja käyttöliittymän toteuttaminen on jokseenkin helppoa.

Verkkosovelluksen päivittäminen uuteen versioon on helppoa. Kun käyttäjä käynnistää verkkosovelluksen, eli toisin sanoen navigoi verkkoselaimellaan sovelluksen osoitteeseen lähettäen palvelimelle GET HTTP -pyynnön, palvelin toimittaa vastauksena käyttäjälle sovelluksen viimeisimmän version. Käyttäjän ei tarvitse tehdä itse mitään saa-

dakseen sovelluksen viimeisimmän version käyttöönsä. Natiivisovellusten kanssa on tyypillistä, että sovelluksen päivittyessä käyttäjä joutuu erikseen asentamaan sovelluksen uuden version.

Enenevissä määrin sovellukset, jotka saattoivat aiemmin olla saatavilla ainoastaan natiivi- tai työpöytäsovelluksina, muutetaan verkkosovelluksiksi edellä mainituista syistä. Useat työpöytäsovellukset käyttävät selainteknologioita käyttöliittymätoteutuksissaan.

Verkkosovellusten historia

Julkisen internetin alkuaikoina ensimmäiset verkkoselaimet tukivat vain staattisia verkkosivuja, koska dynaamisen sisällön mahdollistavia teknologioita ei ollut kehitetty. Käyttäjän syötteet käsiteltiin käyttäen HTML-lomakkeita. Jokaisen pyyntö-vastaus-syklin käsittely päättyi selaimen ikkunan uudelleenlataukseen.

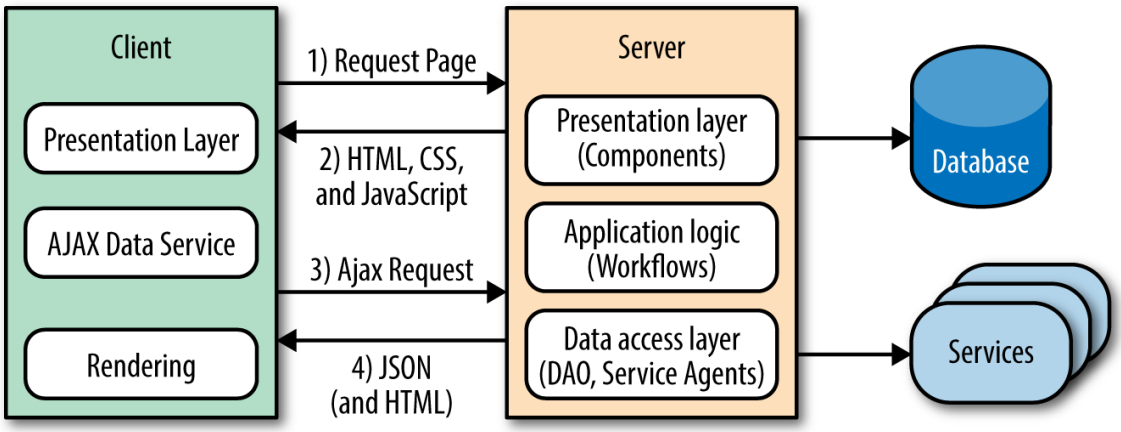
Vuonna 1995 Netscape rekrytoi Brendan Eichin integroimaan Scheme-ohjelmointikielen Netscape Navigator -verkkoselaimeen dynaamista sisältöä mahdollistavaksi skriptikieleksi. Erinäisten vaiheiden jälkeen Netscape tuli kuitenkin siihen tulokseen, että Navigatoriin integroitavan skriptikielen kieliopin tulisi muistuttaa Sun Microsystemsin Java-ohjelmointikieltä. Eich toteutti prototyypin kielestä kymmenessä päivässä. Kielen työnimi oli Mocha, mutta se nimettiin LiveScriptiksi ensimmäisissä beta-versioissa kunnes kielen lopulliseksi nimeksi valittiin Netscape Navigator 2.0 beta 3 -versiossa JavaScript. [4; 5.]

Netscape myös julkisti palvelinpään JavaScript-tuotteen, Netscape Enterprise Serverin. Palvelinpään JavaScript yleistyi kuitenkin vasta vuonna 2009 julkaistun palvelinpään Node.js JavaScript -suoritusympäristön myötä.

JavaScript ja vuonna 1996 julkaistu Macromedia Flash -selainlaajennus olivat ensimmäiset dynaamisen verkkosisällön mahdollistavat teknologiat. Samana vuonna Internet Explorer -selaimessa julkaistu Microsoftin kehittämä iframe HTML -elementti oli ensimmäinen askel selainpään asynkroniseen tietoliikennepyyntöjen käsittelyyn.

Vuonna 1999 Microsoft julkaisi Internet Explorer 5.0 -selaimessa XMLHTTP ActiveX -komponentin. Muut selainvalmistajat myöhemmin sisällyttivät tämän teknologian selaimiinsa XMLHttpRequest-JavaScript-oliona. [5.]

Tästä suorana jatkumona vuonna 2005 syntyi termi Ajax, lyhenne sanoista Asynchronous JavaScript and XML. Ajax ei ole yksittäinen teknologia, vaan tapa käyttää useaa teknologiaa toteuttamaan asynkronista HTTP-tietoliikennepyyntöjen ja -vastausten käsittelyä. XMLHttpRequest-olio toteuttaa pyyntöjen lähettämisen ja vastausten vastaanottamisen; pyyntöjen ja vastausten tietosisältö on koodattu XML- tai JSON-muotoon; sovelluskehittäjä toteuttaa pyyntöjen tietosisällön koostamisen ja vastausten tietosisällön käsittelyn JavaScript-lausekkein; tieto esitetään selaimessa käyttäen HTML-, CSS- ja DOM-teknologioita [6]. Kuvassa 2 on kuvattu Ajax-pyyntöjä käyttävän verkkosovelluksen rakennetta.



Kuva 2. Kolmitasoinen Ajax-pyyntöjä käyttävä verkkosovellus [3].

World Wide Web Consortium julkaisi 1. marraskuuta 2016 HTML5-standardin viimeisimmän version 5.1, eli siitä tuli WWW Consortiumin suositus. HTML5-standardi kattaa suuren joukon muutoksia ja uusia ominaisuuksia HTML-standardiin, joista tässä muutama esimerkki:

- uudet <audio>, <video> ja <canvas> -elementit, jotka mahdollistavat multimediasisältöjen esittämisen selaimessa ilman selainlisäosia
- semanttiset elementit, kuten <aside>, <footer> ja <section>, jotka helpottavat HTML-dokumentin semanttisen rakenteen koneellista tulkintaa
- Web Storage API, jota voi käyttää esimerkiksi tietojen tallentamiseen käyttäjän verkkoselaimeen verkkoyhteyden puuttuessa.

Multimediatoiston mahdollisuus ilman selainlisäosia on johtanut tähän tarkoitukseen aiemmin yleisesti käytettyjen selainlisäosien, kuten Adobe Flash, suosion hiipumiseen. Tämä on entisestään vahvistanut JavaScriptin asemaa dynaamisten sisältöjen tuotantotyökaluna [7].

2.2 JavaScript-ohjelmointikieli

JavaScript on korkean tason ohjelmointikieli. Aiemmin JavaScriptin suoritus verkkoselaimissa oli toteutettu tulkilla, eli JavaScript-lähdekoodi suoritettiin rivi kerrallaan ilman kääntämistä. Nykyään tulkin sijasta käytetään virtuaalikoneita, jotka kääntävät lähdekoodin joko välikieleksi (Mozilla SpiderMonkey) tai konekieleksi (Google V8, WebKit SquirrelFish Extreme). Muistinhallinta on toteutettu automaattisella roskienkeräyksellä.

Heikko-vahvatyypitys liittyy siihen, miten tyyppijärjestelmä toimii käännettäessä. Heikosti tyypitetyn kielen kääntäjä sallii tyyppien käsittelyn, kuin ne olisivat jotain muuta tyyppiä, esimerkiksi kokonaisluvun käsittely merkkijonona, mikä suorituksen aikana saattaa aiheuttaa poikkeuksen. Vahvasti tyypitetyn kielen kääntäjä ei salli tätä, vaan kääntäminen keskeytetään tuottamatta suoritettavaa sovellusta. Jos sivuutetaan se, että JavaScript ei ole yksiselitteisesti käännetty kieli, tällä akselilla JavaScript on heikosti tyypitetty tai vähintäänkin ei-vahvasti tyypitetty.

Dynaaminen-staattinentyypitys liittyy siihen, miten ohjelmointikieli käsittelee tyyppejä suorituksen aikaisesti. Dynaaminen tyypitys tarkoittaa sitä, että muuttujilla ei ole tyyppiä, mutta arvoilla, joihin ne viittaavat, on tyypit. Toisin sanoen muuttujaan, joka viittaa kokonaislukuun, voidaan sijoittaa merkkijono, mutta kokonaisluvun käsittely merkkijonona aiheuttaa poikkeuksen, ellei ohjelmointikieli tue implisiittisiä tyyppimuunnoksia. Staattisesti tyypitetyssä ohjelmointikielessä muuttujilla on tyypit, ja yleensä kun muuttuja esitellään, sille määritellään tyyppi, vaikka siihen ei esittelyn yhteydessä alustettaisi arvoa. JavaScript on dynaamisesti tyypitetty [8].

JavaScript on moniparadigmainen. Siinä on piirteitä strukturoidusta, imperatiivisesta, funktionaalisesta, tapahtumapohjaisesta ja olio-ohjelmoinnista. Strukturoitua ja imperatiivista ohjelmointia edustavat muun muassa ehtolauseet ja silmukat. Funktionaalista ohjelmointia edustsaa esimerkiksi se, että funktiot ovat ensimmäisen luokan olioita Ja-

vaScriptissä: uusia funktioita voidaan luoda ohjelman suorituksen aikana, funktio voi ottaa toisen funktion parametrina, funktioita voidaan sijoittaa muuttujiin ja funktion paluuarvo voi olla funktio. Tapahtumapohjaista ohjelmointia hyödynnetään käyttäjän syötteiden käsittelyssä [9].

JavaScript on prototyyppipohjainen olio-ohjelmointikieli. Uusia olioita luodaan joko kloonaamalla olemassa oleva olio tai kutsumalla rakentajafunktiota. Toisin kuin luokkapohjaisissa olio-ohjelmointikielissä, prototyyppipohjaisissa olio-ohjelmointikielissä, kuten JavaScriptissä, rakentajafunktiot eivät ole sidottuja luokkamääritelmään, koska luokkia ei määritellä. JavaScriptin varattuihin sanoihin lukeutuu class, johon aiemmin ei liittynyt mitään toiminnallisuutta. ECMAScript 2015 määrittää class-sanalle toiminnallisuutta, jonka avulla on näennäisesti mahdollista määritellä luokkia selkeyttämään olioiden ja periytymisen käsittelyä. Tämä toiminnallisuus on kuitenkin vain ”syntaktista sokeria” JavaScriptin prototyyppipohjaiselle periytymisjärjestelmälle [9].

2.3 JavaScript-sovelluskehykset

Prototype.js, MooTools, Dojo Toolkit, YUI Library ja etenkin jQuery olivat ensimmäisiä laajalti verkkosovelluskehityksessä käytettyjä JavaScript-kirjastoja [4; 10]. Niiden ensimmäiset versiot julkaistiin vuosina 2005–2007. Nämä kirjastot tarjoavat työkalupakin, joka etenkin ennen selainten JavaScript-toteutusten yhtenäistymistä helpotti muun muassa Ajaxin käyttöä ja dynaamisen käyttöliittymän toteuttamista. Tämän aikakauden selainpään JavaScript-sovelluslogiikalle oli tyypillistä, että dokumenttioliomallin tilaa muokattiin suoraan tapahtumien käsittelyn yhteydessä. Vaikka palvelinpään sovelluskehyksissä model-view-controller-arkkitehtuuri oli vakiinnuttanut asemansa, selainpään sovelluslogiikkaan suhtauduttiin pelkkänä näkymäkerroksena ja sovelluksen tilaa ei kapseloitu malleihin tai muutoin eriytetty esityksestä. Tilanhallinta oli manuaalista ja haurasta, ja sovelluksen ylläpidettävyys korreloi käänteisesti sovelluslogiikan monimutkaisuuden kanssa [11]. Nämä kirjastot olivat eräänlainen JavaScript-sovelluskehysten esiaste.

Ensimmäiset varsinaiset selainpään JavaScript-sovelluskehykset, kuten KnockoutJS, Angular, Ember.js ja Backbone.js, julkaistiin vuosina 2010–2011. Niissä käytetään model-view-viewmodel-, model-view-presenter- tai model-view-controller-arkkitehtuureja, jotka ovat hieman toisistaan eroavia lähestymistapoja tilan ja sovelluslogiikan kapse-

lointiin. Tietojen kaksisuuntainen sidonta, eli näkymän ja tietomallin tilan synkronointi sovelluskehyksen toimesta, oli tämän aikakauden voimakas trendi [12].

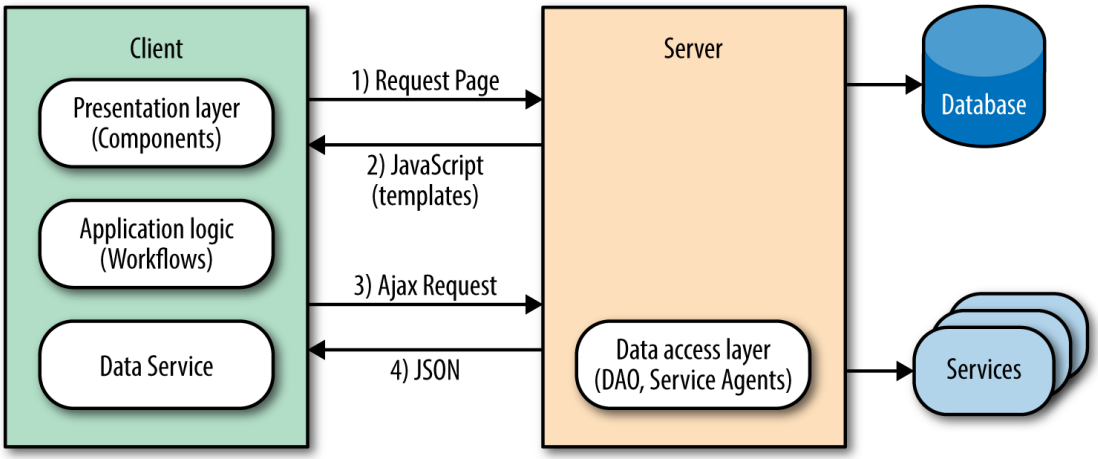
2.4 LAMP-pinosta MEAN-pinoon

Kun käytetään termiä LAMP-pino, tarkoitetaan Linux-käyttöjärjestelmän, Apache HTTP-palvelinsovelluksen, MySQL-relaatiotietokannan ja PHP-ohjelmointikielen käyttöä toteuttamaan verkkosovelluksen palvelinpään osiot. JavaScript-sovelluskehysten myötä verkkosovellusten ohjelmointi käyttäen ”full-stack”-JavaScriptiä yleistyi. Full-stack-JavaScript tarkoittaa sitä että myös verkkosovelluksen palvelinpään osioissa hyödynnetään JavaScriptiä. MongoDB-dokumenttitietokannan, palvelinpään Express.js-sovelluskehyksen, selainpään Angular-sovelluskehyksen ja Node.js JavaScript -suoritusympäristön yhteiskäytöstä ryhdyttiin käyttämään MEAN-pino-nimitystä. Esimerkiksi kun Ember.js-sovelluskehys korvaa Angularin selainpään sovelluskehyksenä, käytetään MEEN-pino-nimitystä.

Kuitenkin tärkeämpää kuin se, mitä spesifejä teknologioita käytetään, on se filosofinen siirtymä, jota MEAN-pinon esiintulo edustaa. Perinteiset LAMP-pinopohjaiset verkkosovellukset ovat palvelinpääkeskeisiä. Selainpään osio sovelluksesta on tiukasti esitystasoa. Käyttöliittymä saatetaan koostaa kokonaisuudessaan palvelinpäässä, eli palvelin lähettää vastauksena käyttäjän pyyntöihin kokonaisia HTML-dokumentteja. Reititys tapahtuu palvelinpäässä, eli kun käyttäjä siirtyy URL-osoitteesta toiseen esimerkiksi sovelluksessa olevan linkin kautta, palvelinpään sovellus päättelee pyydetystä reitistä, minkälaisen HTML-dokumentin se lähettää vastauksena. Sovelluksen palvelinpäässä käytetään JavaScriptiä, palvelinpäässä PHP:tä tai jotain muuta perinteistä palvelinpään ohjelmointikieltä ja tietokantakyselyt toteutetaan SQL-kielellä.

Sitä vastoin MEAN-pino-pohjaisissa verkkosovelluksissa painopiste on enemmän selainpäässä. Sovelluslogiikka voi olla jopa kokonaisuudessaan toteutettu selainpäässä. Palvelinpään osio tuottaa kokonaisten HTML-dokumenttien sijasta tarvittavan datan sisältäviä paluuviestejä, esimerkiksi JSON-dokumenttimuodossa. Käyttöliittymä muodostetaan ohjelmallisesti selainpäässä tämän datan pohjalta. Reititys voidaan myös toteuttaa selainpäässä: URL-osoitteeseen siirtyessä selainpää hakee asynkronisesti seuraavaa näkymää varten tarvittavan datan palvelimelta ja koostaa näkymän ohjelmallisesti. Tämä mahdollistaa modernit yhden sivun sovellukset, single page app

(SPA), eli käyttäjän käynnistettyä verkkosovelluksen selainikkunaa ei ole tarvetta käytön aikana uudelleenladata. Selainpäässä käytetään JavaScriptiä, palvelinpäässä käytetään JavaScriptiä ja jopa tietokantakyselyt voivat olla JavaScriptiä ja tietokanta voi käyttää tietomallinaan JSON-dokumentteja [11]. Kuvassa 3 esitetään yhden sivun verkkosovelluksen rakennetta.



Kuva 3. Yhden sivun arkkitehtuuria toteuttava verkkosovellus [3].

2.5 Meteor-sovelluskehys

Meteor-sovelluskehyksen ensimmäisen versio julkaistiin vuonna 2012. Meteor on siten poikkeuksellinen sovelluskehys, että se on integroitu full-stack-ratkaisu MEAN-pinon hengessä. Käytän tässä insinöörityössä termiä Meteor-sovellusalusta tarkoittamaan tätä full-stack-kokonaisuutta erotuksena siitä sovellusalustan osa-alueesta, joka muodostaa sovelluskehyksen sanan perinteisemmässä merkityksessä.

Kuten MEAN-pino, Meteor käyttää palvelinpään Node.js JavaScript -suoritusympäristöä ja MongoDB-tietokantaa. Express.js ja Angular on korvattu Meteor-sovelluskehyksellä, minkä mahdollistaa se, että Meteor on isomorfinen JavaScript-sovelluskehys. Isomorfisuus viittaa siihen, että samaa koodia voidaan suorittaa suoritusympäristöstä riippumattomasti, eli tässä tapauksessa sekä palvelinpään Node.js-ympäristössä että käyttäjän selaimessa [12]. Sovelluskehittäjän ei tarvitse osata useaa eri sovelluskehystä, ja sovelluksen lähdekoodin jäsentely tiedostorakenteen ja versiohallinnan kannalta siten, että sitä on kehittäjän helppo ymmärtää, on yksinkertaisempaa.

Reaktiivisten käyttöliittymien toteuttaminen Meteorilla on helppoa. Tietokantakursorit ovat reaktiivisia, eli esimerkiksi jos tietokannan tietueita muokataan, käyttöliittymä päivittyy tarvittavilta osin automaattisesti. Samoin käyttöliittymässä tapahtuvat tilamuutokset päivitetään olennaisin osin automaattisesti tietokantaan. On myös muun muassa mahdollista määritellä globaaleja reaktiivisia muuttujia, joihin voidaan kytkeä tilamuutoksen ansiosta suoritettavaa sovelluslogiikkaa [13].

Meteor-kehitysympäristön pystytys on helppoa: käytännössä sovelluskehittäjän tarvitsee suorittaa vain yksi komentorivikomento. Tyypillisesti verkkosovelluksia ohjelmoitaessa sovelluskehys on vain runko itse sovellukselle, mutta sovelluksen suorittamiseen tarvittavat palvelinohjelmat ja niiden asetukset kehittäjä joutuu asentamaan ja asettamaan itse. Meteor hoitaa tämän kaiken kehittäjän puolesta.

Meteor asentaa insecure-paketin oletusarvoisesti. Tämä paketti mahdollistaa tietokannan käsittelyn suoraan selainpään koodissa. Jos poistaa insecure-paketin, tämä ei ole enää mahdollista, vaan selainpään koodin täytyy kutsua palvelinpäässä suoritettavaa funktiota, joka toteuttaa kannan käsittelyn. Tässä yhteydessä voidaan varmistaa tietoturva esimerkiksi tarkistamalla palvelinpäässä, että käyttäjällä on käsittelyyn tarvittavat oikeudet. Insecure-paketti nopeuttaa MVP:n tai prototyypin toteutusta, koska sovelluskehittäjän tarvitsee kirjoittaa vähemmän koodia.

MongoDB-dokumenttitietokanta käyttää JSON-olioita tietomallinaan. MongoDB tukee dokumenttien rakenteen validointia eli sitä, että varmistetaan, että tietokantaan syötettävä tieto on etukäteen määritellyn tietokantaskeeman mukaista. Skeeman ja validaation käyttö on kuitenkin valinnaista, eli toisin kuin relaatiotietokannoissa, MongoDB-tietokantaan voi halutessaan tallentaa rakenteeltaan mielivaltaista dataa. Tämäkin on hyödyllistä prototyyppiä tuotettaessa, koska tietokannan skeemaa ei tarvitse päivittää aina sovelluksen tietomallien muuttuessa.

Meteor tukee kolmea eri kirjastoa näkymien toteuttamiseen: Meteorin oma Blaze, joka muistuttaa syntaksiltaan Handlebars-kirjastoa, Angular ja React.

2.6 React-kirjasto

React on Facebookin kirjasto reaktiivisten käyttöliittymien toteuttamiseen. React kirjastona on melko yksinkertainen, mutta etenkin laajemman React-ekosysteemin taustalla olevat ajatukset ovat mullistavia [14].

React on komponenttipohjainen. Tässä kontekstissa komponenttipohjaisuus tarkoittaa, että käyttöliittymä koostetaan komponenteista, jotka vuorollaan voivat koostua komponenteista. Komponentti on jokin käyttöliittymän elementti, kuten painike tai lista, tai tehtävälista, joka koostuu listasta ja painikkeista. React-komponentilla on kaksi osaa: tila ja render-funktio, joka palauttaa komponentin HTML-kuvauksen. Kun komponentin tila muuttuu, render-funktiota kutsutaan automaattisesti, jotta komponentti on visuaalisesti synkronoitu sen tilan kanssa. Esimerkkikoodissa 1 on kuvattuna yksinkertainen React-komponentti.

```
class HelloMessage extends React.Component {
  render() {
  return <div>Hello {this.props.name}</div>;
  }
}
```

Esimerkkikoodi 1. Yksinkertainen React-komponentti.

React-komponentteja määriteltäessä voi käyttää Reactin JSX JavaScript-syntaksilaajennusta. Koodiesimerkin 1 render-funktion paluuarvossa on käytetty JSX:ää. Kun käytetään JSX-laajennusta, voidaan kirjoittaa koodia, joka muistuttaa hyvin paljon HTML:ää, suoraan komponenttien JavaScript-koodin sisään. Perinteisemmissä näkymäkerroksen komponetisointiratkaisuissa, kuten Meteorin Blaze, ongelmaa yleensä lähestytään päinvastaisesta suunnasta: erillisessä template-tiedostossa HTML:n sekaan kirjoitetaan yksinkertaista, usein mahdollisuuksiltaan rajoittunutta template-spesifistä koodia. JSX:ää käyttäessä näkymämäärittelyn sekaan voi kirjoittaa mielivaltaista JavaScriptiä. JSX on myös siten kehittäjäystävällisempi, että esitystä ja logiikkaa ei ole pakko teknologiarajoitteiden takia pilkkoa useampaan tiedostoon, mikä vähentää kontekstivaihdosten aiheuttamaa kognitiivista kuormaa [15].

React itsessään on vain käyttöliittymäkirjasto. Se ei ota kantaa tai tarjoa työkaluja muutoin verkkosovelluksen rakenteeseen. Facebook kehitti Flux-arkkitehtuurin Reactin

kanssa käytettäväksi tilanhallintamalliksi. Flux ei ole kirjasto vaan kuvaus, miten jäsennellä ja toteuttaa React-sovelluksessa tilanhallintaa. Kolmannet osapuolet ovat toteuttaneet Flux-arkkitehtuuria toteuttavia kirjastoja Reactin kanssa käytettäväksi. Redux on ehkä suosituin näistä kirjastoista.

Reduxia käytettäessä sovelluksen tila mallinnetaan esimerkiksi puumaisena ja usein immutaabelina tietorakenteena. Tilan muutokset hallitaan yhdensuuntaisella syklillä. Yksinkertaistaen tämän syklin voisi kuvata seuraavasti:

- Käyttäjä klikkaa painiketta lisätäkseen uuden tehtävän tehtävälistaan.
- Klikkauksesta syntyvä tapahtuma käsitellään Reduxin sisällä.
- Redux luo uuden tilapuun jonka tehtävälistan tilaa kuvaavaan osioon on lisätty lisättävä tehtävä.
- Redux korvaa vanhan tilapuun tällä uudella tilapuulla.
- Muuttunut tila päivittyy käyttöliittymän komponentteihin.

Tämän tilanhallintamallin etuna on, että kehittäjän on huomattavasti helpompaa suunnitella ja toteuttaa sovelluslogiikka siten, että se on ihmisen helpommin tulkittavissa ja sisäistettävissä, jolloin muun muassa ohjelmointivirheiden määrä vähenee [15].

2.7 Moderni JavaScript-sovelluskehys

Luvuissa 2.3–2.6 käsiteltyihin kuvauksiin perustuen määrittelen seuraavaksi modernin JavaScript-sovelluskehyksen piirteitä. En väitä, että näiden kaikkien tulisi täsmätä, jotta sovelluskehystä voidaan pitää modernina, mutta näitä voidaan käyttää arvioinnin tukena. Piirteitä ovat

- tilan eriyttäminen esityksestä
- reaktiivisuus
- käyttöliittymän dynaamisuus
- asynkronisuus
- reititys selainpäässä
- käyttöliittymän ohjelmallinen laatiminen selainpäässä

- sovelluslogiikan painotus selainpäähän
- käyttöliittymän komponenttipohjaisuus.

Meteor-sovellusalusta React-kirjaston kanssa on näiden piirteiden mukainen.

3 Tilaajaorganisaation viikkopalaveriprosessi

3.1 Yrityksen toimintamallit

Fraktiolla ei ole omia kaupallisia ohjelmistotuotteita tai -hankkeita, vaan se on puhtaasti konsultointiyritys. Fraktion asiakkailleen tarjoamiin palveluihin lukeutuvat ohjelmistokehitys, käyttökokemussuunnittelu, liiketoiminnan kehittäminen, käyttöpalvelu ja koulutus. Näistä palveluista ohjelmistokehitys, käyttökokemussuunnittelu ja liiketoiminnan kehittäminen liittyvät läheisesti toisiinsa. Tyypillinen asiakkaan kanssa toteutettu hanke voi edetä siten, että hankkeen alussa painotus on käyttökokemussuunnittelussa ja liiketoiminnan kehittämisessä, tavoitteena kirkastaa, minkälainen verkkopalvelu tukee parhaalla mahdollisella tavalla asiakasorganisaation liiketoimintaa. Tämän vaiheen tavoitteen saavuttamiseksi voidaan käyttää työkaluina käyttäjähaastatteluita, asiakkaan kanssa toteutettavia suunnittelutyöpajoja ja sähköisiä tai paperisia prototyyppejä.

Alkuvaiheen jälkeen painopiste siirtyy ketterään ohjelmistokehitykseen. Tyypillisesti sovelletaan Kanban-menetelmää: ohjelmistokehitystehtäviä hahmotellaan pienehköiksi kokonaisuuksiksi, tehtävät priorisoidaan asiakkaan kanssa ja ohjelmistokehittäjät ottavat näitä tehtäviä työn alle prioriteettien mukaan. Hankkeen luonteesta riippuen saatetaan pyrkiä tuottamaan MVP (Minimum Viable Product) -versio verkkosovelluksesta, eli toiminnollisuuksiltaan suppein versio, joka tukee asiakkaan liiketoimintaa. Tämän version pohjalta tehtyihin havaintoihin perustuen suunnitellaan, priorisoidaan ja toteutetaan seuraavia tehtäviä, jotka voivat olla ohjelmistokehitystä, käyttökokemussuunnittelua tai liiketoiminnan kehittämistä. Jatkuvasti kehitetään verkkosovellusta paremmin asiakkaan liiketoimintaa tukevaksi. Hankkeen luonteesta riippuen tätä sykliä saatetaan toistaa useita kertoja tai kehitys voi olla jatkuvaa, vuosia kestävää.

Usein hankkeen alussa verkkosovelluksen toimintavarmuuden tai tietoturvan ei tarvitse olla ensiluokkaista, esimerkiksi kun sovellus on vasta asiakkaan sisäisessä testauskäytössä ennen julkistusta. Kuitenkin yleensä jossain vaiheessa saavutetaan sovelluksen

elinkaaressa piste, jolloin jo lainsäädännöllisistä syistä tietoturvaa ei voida täysin laiminlyödä ja sovelluksen toimintavarmuus on asiakkaan liiketoiminnan kannalta oleellista. Tässä vaiheessa käyttöpalvelu on ajankohtainen.

Käyttöpalvelulla tarkoitetaan verkkopalvelun teknistä ylläpitoa. Suoritusympäristöä monitoroidaan: esimerkiksi jos palvelun saavutettavuus heikkenee poikkeuksellisen suuren samanaikaisen kävijämäärän takia riittämättömistä palvelinresursseiden, kuten virtualisoidun palvelinympäristön käyttöön varatun muistin tai prosessoritehon, määrästä, voidaan tähän reagoida reaaliaikaisesti ja lisätä väliaikaisesti näitä resursseja. Palvelinsovellukset, kuten tietokanta ja HTTP-palvelin, päivitetään säännöllisesti viimeisimpiin versioihin tietoturvan ylläpitämiseksi. Lisäksi sovelluksesta varmuuskopioidaan tarvittavat osa-alueet, tyypillisesti tietokanta, turvaamaan muun muassa laitevioista mahdollisesti aiheutuvaa tietomenetystä.

Näiden palveluiden lisäksi Fraktio tarjoaa myös koulutusta. Se järjestää säännöllisesti kaksipäiväistä ”Suomen rankin React-valmennus” -kurssia. Fraktiolta on myös mahdollista tilata räätälöityjä koulutuksia muun muassa verkkosovellusten tietoturvasta, PHP-sovelluskehityksestä, käyttökokemussuunnittelusta tai yrityskulttuurin kehityksestä. Fraktion työntekijöiden on mahdollista vapaasti osallistua näihin koulutuksiin. Lisäksi Fraktio järjestää kaikille avoimia koulutustapahtumia, jotka Fraktio on brändännyt nimellä perjantaipresentaatio eli perjantaipresis. Perjantaipresis järjestetään lähes joka perjantai kello 16.00, ja se on noin tunnin kestävä tilaisuus, jossa joku pitää esitelmän jostain itselleen läheisestä aiheesta. Esitelmän pitäjä voi olla Fraktion työntekijä tai joku ulkopuolinen puhuja. Perjantaipresisten aiheina on ollut esimerkiksi sienestäminen, oluen paneminen, verkostoituminen ja työn tulevaisuus.

Yrityksen organisaatiorakenne on litteä. Työntekijöillä ei ole esimiehiä. Ainoastaan toimitusjohtaja Jesse Peurala on jokaisen työntekijän esimies, mutta lähinnä hallinnollisessa mielessä, muun muassa siinä merkityksessä, että hänellä on allekirjoitusoikeus sopimusasioissa. Koska päätöksenteko on lähes poikkeuksetta demokraattista, päätösvaltaisia johtajia ei tarvita päivittäisessä toiminnassa. Johtajuutta vaativissa tilanteissa valitaan joku vapaaehtoinen fasilitoimaan tilannetta tai muutoin ottamaan ohjat tilanteen kulussa.

Töitä tehdään itseorganisoituvissa tiimeissä. Yrityksen sisäisiä asioita edistämään tiimit usein muodostuvat spontaanisti sen mukaan, kuka haluaa edistää kyseistä asiaa, esi-

merkiksi neuvotteluhuoneen sisustuksen uusimista tai vuosittaisen henkilöstön retriitti-viikonlopun ohjelman suunnittelua. Asiakasprojekteissa tiimin ensimmäinen kokoonpano muodostetaan ottaen huomioon työntekijöiden työkuormat ja toiveet, minkälaisia asioita he haluaisivat olla mukana edistämässä. Hankkeen aikana kokoonpanoon saattaa tulla erinäisistä syistä muutoksia, ja tiimi pyrkii varmistamaan, että hanketta edistävästä tiimistä löytyy tarvittava osaaminen. Hankkeen tarpeet tämän osaamisen suhteen voivat vaihdella: esimerkiksi käyttöliittymäsuunnittelun tarve vaihtelee hankkeen eri vaiheissa, tai asiakasorganisaation rahoituspohja kokee muutoksia, minkä vuoksi tarvitaan enemmän tai vähemmän tekijöitä. Vastuu ei kuitenkaan ole ainoastaan tiimin harteilla, vaan käytännössä resursointia käydään avoimesti läpi koko organisaation tasolla vähintään viikoittain.

Asiakasprojektitiimi hoitaa itsenäisesti hanketta asiakkaan kanssa yhteistyössä: kommunikoi suoraan asiakkaan kanssa pikaviestimellä; sopii prosesseista; päättää käytettävistä teknologioista; järjestää tarvittavat palaverit; suunnittelee, priorisoi ja toteuttaa verkkosovelluksen kehitykseen liittyvät työtehtävät.

Tiimeillä on paljon vapautta sen suhteen, miten ne järjestävät työtehtävät. Yksi Fraktion keskeisistä periaatteista on miettiä, edistääkö jokin päätös tai asia työntekijöiden tyytyväisyyttä; edistääkö se asiakkaan tyytyväisyyttä; ja onko se taloudellisesti järkevää. Näiden kolmen ehdon puitteissa on tiimeillä ja yksilöillä mahdollisuus tehdä päätöksiä ja valintoja miltei täysin vapaasti työntekoon liittyen.

Byrokratia ja prosessit pyritään pitämään mahdollisimman vähäisinä ja kevyinä. Uusia prosesseja lähdetään herkästi kokeilemaan, jos ne vaikuttavat potentiaalisesti hyödyllisiltä, mutta niistä myös luovutaan tai kokeillaan jotain muuta, jos niistä saavutettavaa hyötyä ei koeta tarpeeksi suureksi, soveltaen kokeilukulttuurin aatetta.

3.2 Viikkopalaveriprosessi

Fraktion vakiintuneisiin prosesseihin kuuluu viikkopalaveri. Se on kerran viikossa järjestettävä palaveri, johon osallistuu koko yhtiön henkilöstö, ja se on ainoa tällä laajuudella säännöllisesti pidettävä Fraktion sisäinen palaveri. Viikkopalaverissa käsitellään asioita, jotka ovat relevantteja yritystasolla. Asiat, jotka koskevat vain osaa yrityksen henki-

löstöstä, esimerkiksi asiakashankkeiden päivittäistoiminnan yksityiskohtiin liittyvät asiat, pyritään käsittelemään jonain muuna hetkenä.

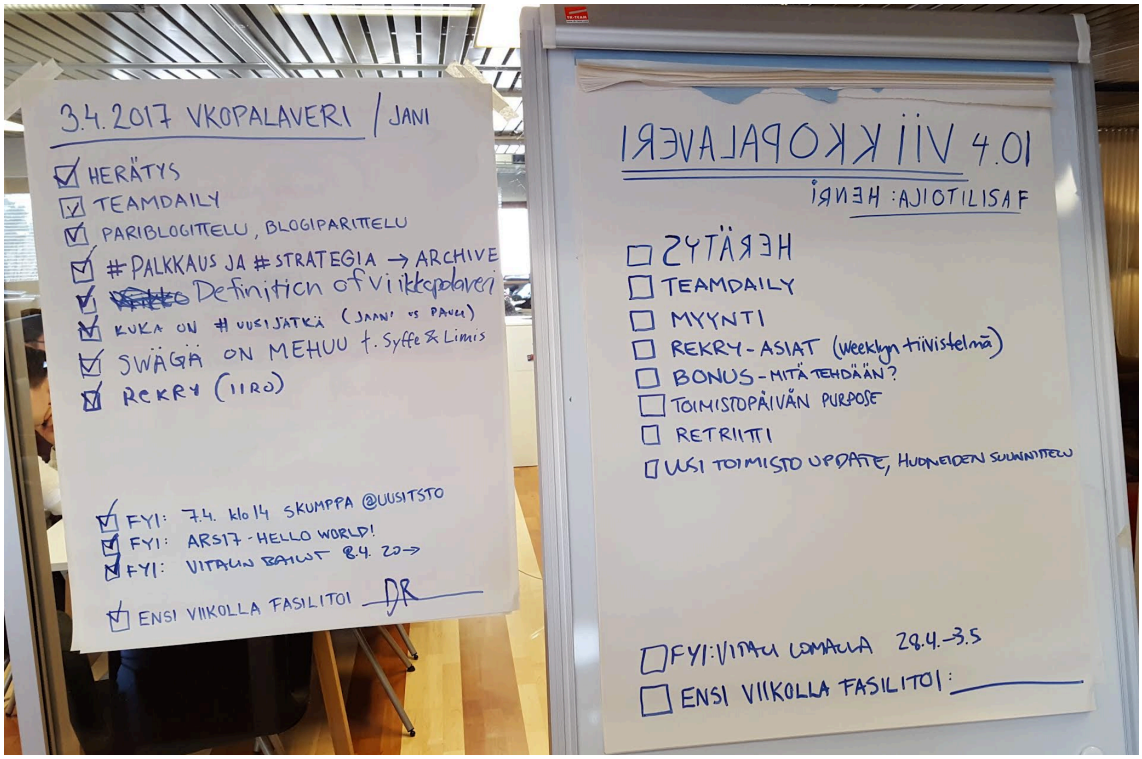
Viikkopalaverin historia

Kuten muiden prosessien, myös viikkopalaverin suhteen on kokeiltu erilaisia variaatioita. Kun menin töihin Fraktiolle marraskuussa 2014, silloisessa viikkopalaverin versiossa keskeinen sisältö oli käydä läpi työntekijöiden työtilanteet. Kukin kertoi vuorollaan työkuormastaan ja edistämiensä hankkeiden kuulumisista. Kertomuksen perusteella merkittiin Google Sheets -laskentataulukkoon henkilökohtaiset muistiinpanot työkuormasta. Työntekijöillä oli oma rivi, ja yksi sarake vastasi viikkoa. Kuluvan viikon solun väri asetettiin vihreäksi, jos työkuorma oli sopiva, punaiseksi, jos töitä oli liikaa, ja lohenpunaiseksi, jos töitä oli liian vähän. Tätä tietoa kerättiin lähinnä ohjaamaan myyntiä: onko tarve saada nopeasti lisää asiakashankkeita, vai pitääkö myynnin suhteen jarrutella. Työkuormien lisäksi käytiin läpi muita ajankohtaisia asioita, esimerkiksi potentiaalisia tulevia hankkeita. Viikkopalaverit pidettiin Länsi-Pasilan Huolintatalossa sijaitsevan Fraktion toimiston neuvotteluhuoneessa maanantaiaamuisin kello 8.00 alkaen, eikä niiden kestoa ollut rajattu. Palaverilla ei ollut fasilitoijaa, työntekijät esittivät keskustelunaiheita spontaanisti ja puhuivat niistä ilman jaettuja puheenvuoroja. Palaverin eteneminen oli melko kaoottista.

Tämä formaatti koettiin turhan jäykäksi, palavereiden tunnelma oli usein painostava. Jotkut myös kokivat alkamisajankohdan haastavaksi: viikonlopun jälkeen oli raskasta tulla kello 8.00 alkavaan palaveriin etenkin aamuruuhkien vuoksi. Lisäksi henkilöstön kasvaessa neuvotteluhuone kävi ahtaaksi, ja palaverin kesto koettiin ongelmalliseksi, kun jokainen uusi työntekijä merkitsi palaverin kestoajan pidentymistä. Viikkopalaverit siirrettiin Huolintatalon Sodexon tiloihin lisätilan vuoksi. Vuokrattavaa neuvottelutilaa kokeiltiin, mutta sitä ei koettu tarpeelliseksi, kun avoimessa ruokalatilassa ei aamuisin käytännössä ollut muita asiakkaita, joten palaverit saattoi hyvin pitää siellä. Alkamisajankohta vaihdettiin kello 9.00:ksi. Henkilökohtaisten työtilanteiden läpikäymisestä luovuttiin palaverin keston järkeistämiseksi. Miljöö oli myös rennompi, ja Sodexon aamiaistarjoilua saattoi nauttia yrityksen laskuun, mikä edesauttoi rentoutta ja toisaalta lisäsi alkamisajankohdan tunnilla eteenpäin siirron hyötyä. Palaverilla ei edelleenkään ollut fasilitoijaa, ja aiheet, puheenvuorot ja palaverin kesto olivat edelleen suunnittelemattomia.

Palaverin kesto koettiin edelleen ongelmalliseksi. Usein viikkopalaverissa nousi käsittelyyn aiheita, joista keskusteltiin pitkään ilman varsinaista lopullista päätöstä tai ratkaisua, ja samasta aiheesta saatettiin käydä miltei samat puheenvuorot läpi seuraavassa viikkopalaverissa. Lisäksi pieni joukko henkilöstöstä oli suurimman osan ajasta äänessä ja muiden oli vaikeaa saada omia ajatuksiaan ilmaistuksi, koska kukaan ei ollut jakamassa puheenvuoroja. Työtilanteiden läpikäynnistä luopuminen aiheutti myynnille haittaa, kun sen selkeys, kuinka paljon työntekijöillä on laskutettavaa työtä, hämärtyi.

Fraktion henkilöstön keskuudessa on käytäntönä lähteä lounastamaan johonkin paikalliseen lounasravintolaan kello 11.15. Viikkopalaverin alkamisajankohta siirrettiin 10.30:ksi, jotta lounastauko loisi palaverille luonnollisen päättymisajan ja keston. Samalla viikkopalaveri siirrettiin Fraktion toimiston taukotilaan, koska se koettiin palaverin luonteen vuoksi miellyttävämmäksi ja rennommaksi kuin Sodexon tilat, ja siellä on toistaiseksi edelleen tarpeeksi tilaa koko henkilöstölle. Työtilanteiden seurantaa varten kehitettiin sisäisesti verkkosovellus, johon kukin voisi itse kirjata työkuormansa ja edistämiensä hankkeiden senhetkisen tilan. Toimiston seinälle laitetaan viikkopalaverin jälkeen lehtiötauluarkki, johon kuka tahansa voi käydä kirjaamassa käsiteltäviä aiheita seuraavaa viikkopalaveria varten, jotta käsiteltävien aiheiden määrä ja siten kullekin aiheelle liikenevä aika on paremmin arvioitavissa. Kuvassa 4 ovat kahden viikkopalaverin arkit.



Kuva 4. Fraktion kahden viikkopalaverin asialistat.

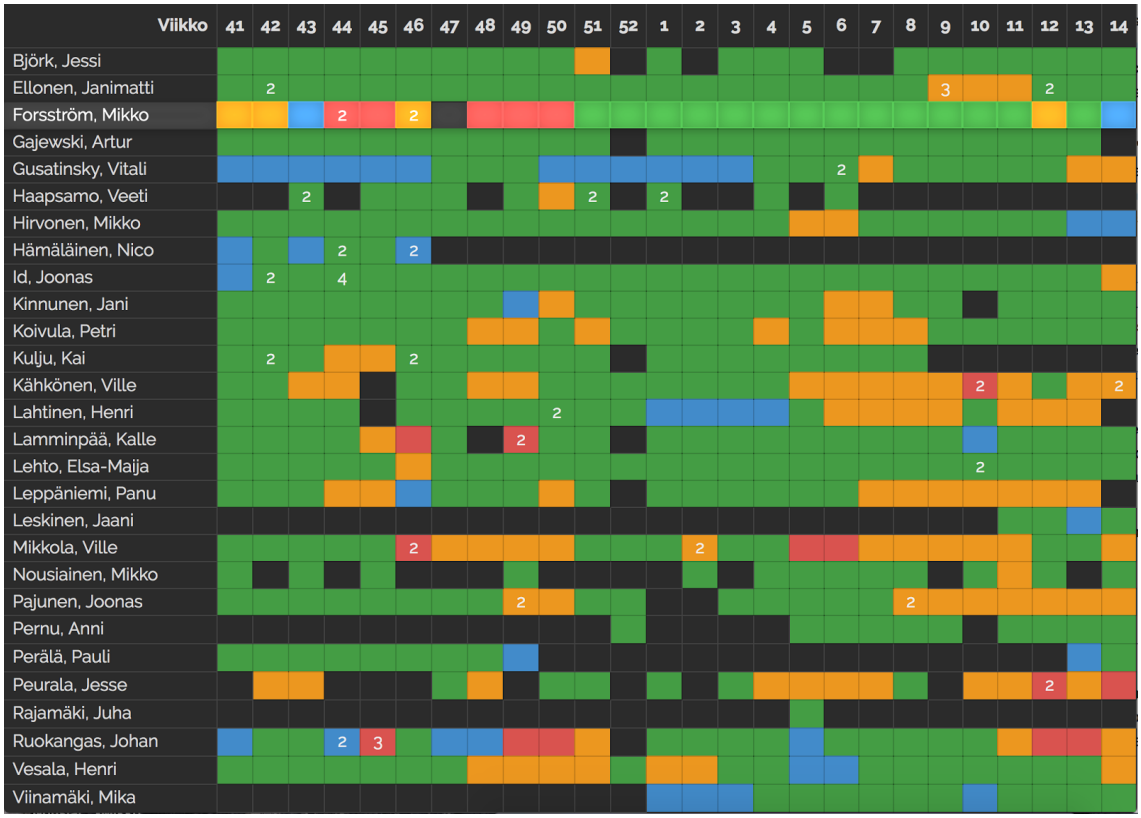
Palaveria johtamaan otettiin viikoittain vaihtuva fasilitoija, joka valitaan palaverin viimeisenä käsiteltävänä asiana seuraavaa viikkopalaveria varten ja kirjataan seuraavan palaverin aihearkkiin. Jokainen fasilitoija johtaa viikkopalaveria vähän omalla tyylillään, mutta fasilitoijan keskeisin tehtävä on jakaa puheenvuorot. Puheenvuoroja pyydetään viittaamalla, ja fasilitoija myöntää puheenvuorot siinä järjestyksessä kuin niitä on pyydetty. Harkintansa mukaan hän priorisoi henkilöitä, joilla on jotain aiheen kannalta tärkeää informaatiota tai jotka ovat esimerkiksi puhuneet vähemmän palaverin aikana. Fasilitoija esittelee aiheet, yleensä lukemalla aihearkilta seuraavaksi käsiteltävän aiheen ääneen ja kysymällä, kuka aiheen on lisännyt, jotta aiheen lisääjä voisi pohjustaa, mistä on kyse. Fasilitoija yleensä yliviivaa tai muutoin merkitsee aihearkista käsitellyt aiheet käsitellyiksi selkeyttämään, mitkä asiat ovat vielä käsittelemättä.

Fasilitoijan vastuulla on pyrkiä varmistamaan, että kaikki aiheet saadaan käsiteltyä palaverin aikana ilman, että palaveri jatkuu yliajalle, mikä saattaa olla haastavaa, koska joistain aiheista saattaa syntyä yllättävän paljon keskustelua ja tavoitteena on, että kaikki voivat halutessaan ilmaista mielipiteensä ja näkemyksensä. Tämän varmistamiseksi fasilitoija saattaa pyytää, että siirrytään seuraavaan aiheeseen, tai todeta, että jostain aiheesta otetaan enää rajallinen määrä uusia puheenvuoroja, jos aiheesta kes-

kusteluun on jo käytetty runsaasti aikaa. Jos yksittäisen aiheen käsittely jää selkeästi kesken ja aihe on aikakriittinen, saatetaan järjestää erillinen palaveri tämän aiheen loppuun käsittelemiseksi viikkopalaverin jälkeen.

Suurin osa Fraktion henkilöstöstä työskentelee Fraktion toimistolla, mutta osa työskentelee asiakkaan toimitiloissa. Koska tavoitteena on, että kaikki osallistuvat viikkopalaveriin, ryhdyttiin viikkopalavereissa käyttämään Google Hangouts -videopuhelua, jotta myös asiakkaan tiloissa työskentelevät voivat osallistua sitä kautta viikkopalaveriin.

Kuvassa 5 on kuvakaappaus TeamDaily-sovelluksen matrisiinäkymästä. Matriisinäkymästä saa nopealla vilkaisulla käsityksen asiakastyön määrästä.



Kuva 5. TeamDaily-sovelluksen matriisinäkymä asiakastyön määrästä.

Nykymuotoisessa viikkopalaverissa käydään palaverin aluksi nopeasti läpi työtilanne vilkaisemalla TeamDailyn matriisinäkymää ja keskustelemalla siitä, onko kenelläkään liian vähän tai liikaa töitä. Aktiivisessa myyntivaiheessa olevat potentiaaliset tulevat hankkeet käydään läpi. Muut aiheet vaihtelevat sen mukaan, mikä on ajankohtaista, muun muassa tulevat tapahtumat, kuten sisäiset koulutukset; rekrytointi: onko tarvetta

rekrytoida uusia työntekijöitä, ketkä haastattelevat haastatteluun tulossa olevat hakijat, hakijoita haastatelleet kertovat havaintojansa ja mielipiteitänsä hakijoista, tehdään haastatelluista hakijoista päätös, palkataanko vai ei; ilmoitusluontoiset asiat, kuten lomat tai poissaolot ja niin edelleen.

Viikkopalaverin haasteet

Vaikka viikkopalaverin nykyinen versio on huomattavasti parempi kuin aiemmat, on siinäkin vielä parannettavaa. Fraktion toimiston taukotilassa on keskellä pilari, joka estää näkyvyyttä katsojasta katsottuna tilan vastakkaiselle puolelle. Tämän vuoksi fasilitoija ei välttämättä huomaa, jos pilarin toisella puolella joku viittaa. Jos useampi henkilö viittaa, fasilitoija jakaa puheenvuorot siinä järjestyksessä, jossa viittaajat ovat viitanneet, ja koska fasilitoija ei välttämättä muista, ketkä halusivat puheenvuoron, viittaajat joutuvat viittaamaan useamman kerran. Jos viittaajalla on jotain tärkeää sanottavaa, käytännössä hän joutuu puhumaan puheenvuorot ohittaen ilmoittaakseen tästä. Jotkut saattavat ottaa oma-aloitteisesti puheenvuoron, jos eivät näe jonkun muun viittaavan pilarin toisella puolella, olettaen, että voivat olla tehokkaampia ajankäytön suhteen, kun eivät jää odottamaan fasilitoijan vuoronjakoa.

Hangouts-videopuhelun kautta osallistuttaessa viikkopalaveriin puheenvuoron pyytäminen tehdään käytännössä siten, että odotetaan hiljaista hetkeä, jotta voi suullisesti pyytää puheenvuoroa. Koska aiheet on kirjattu paperille, etäosallistujia varten otetaan paperista kuva, joka jaetaan sisäisessä käytössä olevan Slack-pikaviestimen kautta. Välillä tämä kuva unohdetaan ottaa. Viikkopalaverin aikana joku tekee muistiinpanoja käsitellyistä aiheista ja niistä käydystä keskustelusta, ja nämä muistiinpanot jaetaan Slackin kautta, mutta vanhoihin muistiinpanoihin palaaminen on työlästä, ja välillä nämä muistiinpanot jäävät tekemättä.

Kokeilukulttuurin hengen mukaisesti ryhdyin insinöörityönä selvittämään, voisiko viikkopalaveriprosessia parantaa sen tarpeisiin räätälöidyllä verkkosovelluksella.

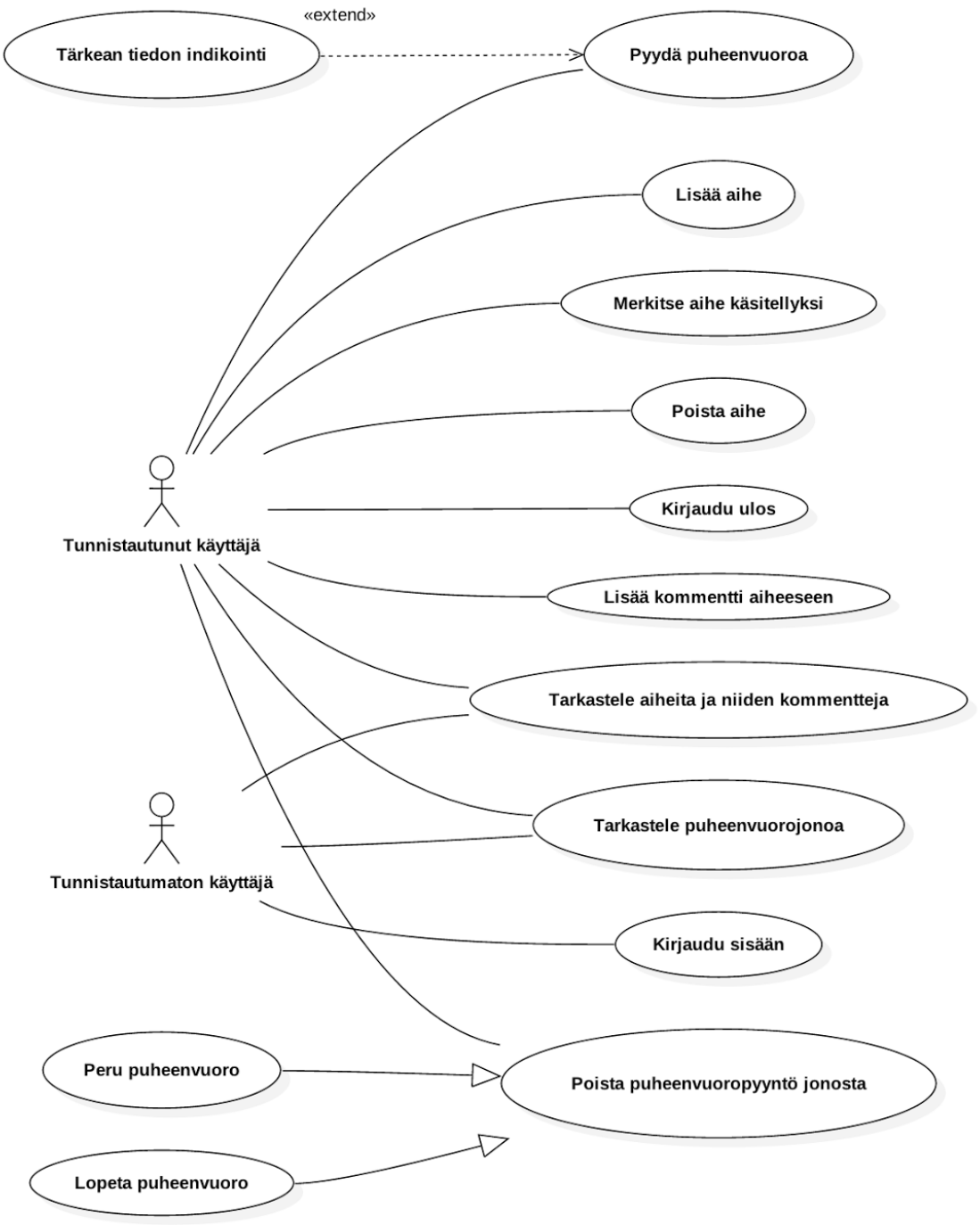
4 Viikkopalaverisovellus

4.1 Viikkopalaverisovelluksen toiminnollinen ja tekninen määrittely

Viikkopalaverin haasteita pohdittuani ryhdyin suunnittelemaan viikkopalaverisovelluksen mahdollisia toiminnollisuuksia ja näkymiä. Olen osallistunut kymmeniin viikkopalavereihin, joten minulla on kattava käsitys viikkopalaverista prosessina.

Suunniteltuja ominaisuuksia

Kuvassa 6 ovat kuvattuna viikkopalaverisovelluksen käyttötapaukset.



Kuva 6. Viikkopalaverisovelluksen käyttötapaukset

Viikkopalaverisovelluksessa tulisi olla mahdollista pyytää puheenvuoroa siten, että

- puheenvuoroa pyytävän ei tarvitse toistuvasti esittää pyyntöä saadakseen puheenvuoron
- puheenvuorot jakautuvat oikeassa järjestyksessä ilman inhimillisen virheen riskiä

- fasilitoijalla ei tarvitse olla näköyhteyttä puheenvuoron pyytäjään
- voi muita keskeyttämättä indikoida, että tietää keskustelun kannalta jotain tärkeää minkä takia pyytäjän tulisi saada seuraava puheenvuoro muiden jonossa olevien pyyntöjen edelle
- etäosallistujat voivat pyytää puheenvuoroa samalla tavalla kuin läsnäolevat
- puheenvuoron pyytäminen on päätelaitteesta riippumatonta.

Pyydettyjen puheenvuorojen tulisi olla sovelluksessa kaikkien käyttäjien nähtävissä, esitettynä siten, että on ilmeistä, missä järjestyksessä niitä on pyydetty, jotta puheenvuoroa pyytäneet tietävät, milloin on heidän puheenvuoronsa. Käytetyn puheenvuoron tulisi poistua näkymästä tai muutoin selvästi erottua jonossa olevista puheenvuoroista. Meneillään oleva puheenvuoro voisi olla visuaalisesti korostettu. Puheenvuoro tulisi voida päättää, jotta aktiivinen puheenvuoro siirtyy jonossa seuraavalle. Puheenvuoropyyntö olisi hyvä voida perua, jos esimerkiksi joku sanoo jo aiemmassa puheenvuorossa saman, mitä puheenvuoron pyytäjä aikoi sanoa.

Puheenvuoroa pyytäessä tulisi olla mahdollista indikoida, että tietää jotain aiheen kannalta tärkeää ja siksi haluaa jonosta riippumatta seuraavan puheenvuoron. Sovellus voisi pitää kirjaa siitä, kuinka paljon kukin on ollut äänessä, mitattuna esimerkiksi puheenvuorojen määränä, ja priorisoida henkilöitä, jotka ovat puhuneet selkeästi vähemmän sellaisten edelle, jotka ovat puhuneet keskiarvoa enemmän. Puheenvuoropyynnöt voisivat olla sovelluksen ensimmäisessä versiossa yhdessä jonossa, joka ei ole sidottu aiheisiin, mutta myöhemmin voisi kokeilla versiota, jossa pyynnöt voisivat olla sidottuna aiheisiin. Jokaisella aiheella olisi oma jono, ja puheenvuoropyyntöjä voisi etukäteen liittää aihekohtaisesti, jos tietää, että haluaa ilmaista mielipiteensä tai muuta vastaavaa jostain kyseisestä aiheesta. Tässä versiossa pyyntöjonon aktiivisuus olisi sidottuna aiheen aktiivisuuteen. Tämä malli voisi edesauttaa aihekohtaista aika-allokointia ja priorisointia, jos käyttäjät aktiivisesti lisäisivät puheenvuoropyyntöjä ennakkoon.

Viikkopalaverissa käsiteltävien aiheiden tulisi olla digitaalisina sovelluksessa. Asiakkaiden toimitiloissa työskentelevien tai muusta syystä etäosallistuvien tulisi olla mahdollista katsella viikkopalaveriaiheita sovelluksesta. Käsiteltävän aiheen tulisi olla visuaalisesti korostettu. Aiheiden lisäämisen tulisi olla ajasta ja paikasta riippumatonta. Aiheen lisääjä tulisi olla liitettynä aiheeseen. Aiheet tulisi voida merkitä käsitellyiksi. Käsittele-

mättömät aiheet olisi hyvä voida siirtää seuraavan viikkopalaverin agendaan, jos esimerkiksi aika ei riitä niiden käsittelyyn.

Aihekohtaisia muistiinpanoja olisi hyvä voida kirjata suoraan sovellukseen, jotta aiheet ja muistiinpanot olisivat keskitetysti yhdessä paikassa. Edellisten viikkojen aiheisiin ja muistiinpanoihin tulisi olla mahdollista palata.

Sovelluksen tulisi tunnistaa eri käyttäjät, jotta puheenvuoropyynnöt, aiheet, muistiinpanot ja mahdolliset tilastoinnit yksilöityvät oikeille käyttäjille. Käytännössä tämä tarkoittaa, että sovelluksessa on henkilökohtaiset käyttäjätilit tai että käytetään vastaavaa ratkaisua kuin TeamDaily-sovelluksessa, jossa käyttäjä valitsee nimensä alasvetovalikosta.

Jotta muun muassa puheenvuorojen järjestyksen suhteen ei olisi epäselvyyksiä, sovelluksen tilan tulisi synkronoitua eri käyttäjien päätelaitteiden välillä automaattisesti.

Sovelluksessa voisi olla ajanotto-ominaisuuksia. Palaverin kokonaiskestolle voisi olla ajastin edesauttamaan palaverin keston hallintaa. Aiheilla voisi olla kello, jotta konkreettisesti nähtäisiin, kuinka paljon mistäkin aiheesta keskusteluun on käytetty aikaa, mahdollisesti jopa puheenvuoroilla. Toisaalta tämäntyyppiset ajastinominaisuudet saatettaisiin kokea häiritsevinä, kun viikkopalaveria on aiemmin haluttu viedä tunnelmaltaan rennompaan suuntaan, joten näiden ominaisuuksien hyötyä tulisi punnita mahdollista haittaa vasten.

4.2 MVP-version ominaisuudet

Luvussa 4.1 esitellyt vaatimukset huomioiden suunnittelin viikkopalaverisovelluksen MVP-versiota, jonka pohjalta voisi todeta, parantaako se viikkopalaveriprosessia ja onko sen mahdollinen jatkokehitys kannattavaa. Keskeisimmät ominaisuudet sovelluksen kannalta ovat puheenvuoropyynnön lisääminen ja poistaminen ja pyyntöjen järjestyminen ajan mukaan; aiheiden lisääminen, poistaminen ja käsitellyksi merkitseminen; näihin liittyvät yksinkertaiset näkymät ja näkymien tilan automaattinen synkronointi käyttäjien välillä. Nämä ominaisuudet kattavat enemmistön viikkopalaveriprosessin nykyisistä haasteista.

Aktiivisen puheenvuoron tai aiheen visuaaliset korostukset ovat toisarvoisia, koska keskustelusta on yleensä pääteltävissä, kenen vuoro on puhua ja mistä puhutaan. Puheenvuorojen järjestyminen muussa kuin aikajärjestyksessä, eli ”tärkeätä informaatiota”- tai ”vähemmän puhuneiden priorisointi”-tapauksissa, on tarvittaessa toteutettavissa sosiaalisin mekanismein hyväksyttävän suuruisella haittavaikutuksella palaverin kulkuun. Samoin sovelluksen sisällä tapahtuva muistiinpanojen kirjaaminen ja ajanotto eivät varsinaisesti ratkaise mitään niin suurta ongelmaa, että niiden toteuttamisesta syntyvä hyöty olisi välttämättä vaivan arvoista, mutta hyödyn suuruutta voisi arvioida MVP-toteutuksen jälkeen.

Niin pitkään kuin sovelluksessa ei ole muistiinpanomahdollisuutta, ei edellisten palaverien asialistan tarkastelusta ole juuri hyötyä, koska aiheet kirjataan myös viikkopalaverin jälkeen jaettaviin muistiinpanoihin. Jos sovelluksessa ei ole toteutettuna edellisten palaverien tallentamista, ei ole myöskään järkevää toteuttaa tulevien palaverien asialistaan etukäteen aiheiden lisäämistä, eli käytännössä sovelluksessa on vain yksi asialista, jota muokataan. Jos jokin aihe jää käsittelemättä palaverissa, se voidaan jättää listalle odottamaan seuraavaa palaveria.

Sovelluksessa on yksi päänäkymä, joka koostuu kahdesta listasta: aiheet ja puheenvuorot. Aiheiden lisäämiseen on tekstikenttä. Aihelistauksen rivissä on valintaruutu, josta aihe voidaan asettaa käsitellyksi, aiheteksti, aiheen lisääjän nimi, sekä nappi, josta rivi voidaan poistaa. Aihelistan rivit järjestyvät nousevassa lisäysaikajärjestyksessä, ja niiden järjestys muuttuu, kun aihe asetetaan käsitellyksi, siten että listassa ovat ensin käsittelemättömät aiheet ja listan pohjalla käsitellyt. Puheenvuorot ovat listana nousevassa aikajärjestyksessä. Listan yhteydessä on painike, jota painamalla kirjautuneen käyttäjän puheenvuoropyyntö lisätään listaan. Samaa painiketta uudelleen painamalla pyyntö poistuu listasta.

Sovelluksen toiminnollisuudet, esimerkiksi puheenvuoropyyntöihin liittyvät, edellyttävät, että sovellus tukee useaa rinnakkaista käyttäjää. Sovelluksen tila tulee olla synkronoitu käyttäjien välillä, joten luonnollinen valinta sovelluksen korkean tason arkkitehtuuriksi on asiakas-palvelinmalli. Päätelaiteriippumattomuus on hyödyllistä, jotta sovellusta voi käyttää sekä kannettavalla tietokoneella että älypuhelimella. Nämä seikat puoltavat sitä, että sovellus on järkevää toteuttaa verkkosovelluksena.

Sovellus on MVP-vaiheessa kokonaisuudessaan hyvin käyttöliittymävetoinen ja on myös hyvin todennäköisesti sitä mahdollisen jatkokehityksen myötä. Luvussa 2 esitellyistä syistä moderni JavaScript-sovelluskehys on luonnollinen valinta sovelluksen toteutusta varten.

4.3 Viikkopalaverisovelluksen toteutus

Koska tämän insinöörityön viikkopalaverisovellusprojektin keskeinen tavoite oli selvittää, voisiko viikkopalaveriprosessia parantaa räätälöidyllä verkkosovelluksella, keskeinen tekijä sovelluskehystä valitessani oli mahdollisimman suuri kehitysnopeus. Mikäli tultaisiin siihen tulokseen, että sovelluksesta ei ole tarpeeksi hyötyä, jotta sen käyttöä haluttaisiin jatkaa, mitä vähemmän aikaa prototyypin toteuttamiseen on käytetty, sitä parempi. Lisäksi tilan synkronointi eri käyttäjien välillä on sovelluksen toiminnan kannalta hyvin keskeistä, joten tämän mahdollisimman helppo toteutus oli valinnan kannalta myös tärkeä kriteeri. Luvussa 2 esitellyistä syistä valitsin Meteor-sovellusalustan projektin pohjaksi.

Valitsin näkymäkerrosteknologiaksi Reactin. Vaivattoman full-stack-reaktiivisuuden toteuttamiseksi Meteorin tilanhallinta on tiukasti integroitu sovellusalustaan, joten sen korvaaminen Redux-pohjaisella tilanhallinnalla olisi haastavaa. Kuitenkin Reactin komponenttipohjaisuus on hyödynnettävissä. Näiden valintojen myötä ryhdyin toteuttamaan sovellusta.

Asensin Meteor-sovellusalustan tietokoneelleni ja perustin uuden Meteor-sovellusprojektin esimerkkikoodin 2 ensimmäisellä kahdella komennolla. Suorittamalla meteor-komento sovellusprojektin juurikansiossa sovellusalusta luo sovelluksen tarvitsemat kansiot ja tiedostot ja asentaa tarvittavat kirjastoriippuvuudet.

```
curl https://install.meteor.com/ | sh
meteor create viikkopalaveri
cd viikkopalaveri
meteor
```

Esimerkkikoodi 2. Meteor-projektin perustamiseen tarvittavia komentorivikomentoja.

Meteor luo esimerkkikoodin 3 mukaisen tiedostorakenteen sovelluksen pohjaksi. Client-nimisissä kansioissa oleva koodi ladataan ja suoritetaan selainpäässä, ja sitä ei

ladata tai suoriteta palvelinpäässä. Server-nimisissä kansioissa oleva koodi ladataan ja suoritetaan palvelinpäässä, ja sitä ei ladata tai suoriteta selainpäässä. Tiedostossa package.json on npm-paketinhallintajärjestelmästä asennettavien pakettien tiedot, ja npm asentaa paketit node_modules-kansioon. Kansio .meteor on Meteor-sovellusalustan projektikohtaisia asetuksia ja tietoja varten.

```
client/main.js
client/main.html
client/main.css
server/main.js
node_modules
package.json
.meteor
.gitignore
```

Esimerkkikoodi 3. Yksinkertaisen Meteor-sovelluksen tiedostorakenne.

Tiedostossa .gitignore määritetään, mitkä tiedostot ja kansiot Git-versiohallintajärjestelmän tulisi jättää huomiotta. Tämä on hyödyllistä esimerkiksi node_modules-kansion kohdalla, sillä asennettujen pakettien lisääminen versiohallintaan on turhaa, koska ne ovat aina asennettavissa package.json-tiedostoa käyttäen. Git-versiohallintajärjestelmän käyttö Meteor-sovelluskehityksessä, ja yleisesti verkkosovelluskehityksessä, on yleistä mutta ei pakollista. Käytin Git-versiohallintajärjestelmää kehittäessäni viikkopalaverisovellusta.

Esimerkkikoodissa 4 on komento, jolla asensin React- ja ReactDOM-kirjastot käyttäen Meteor sovellusalustaan paketoitua npm paketinhallintaa.

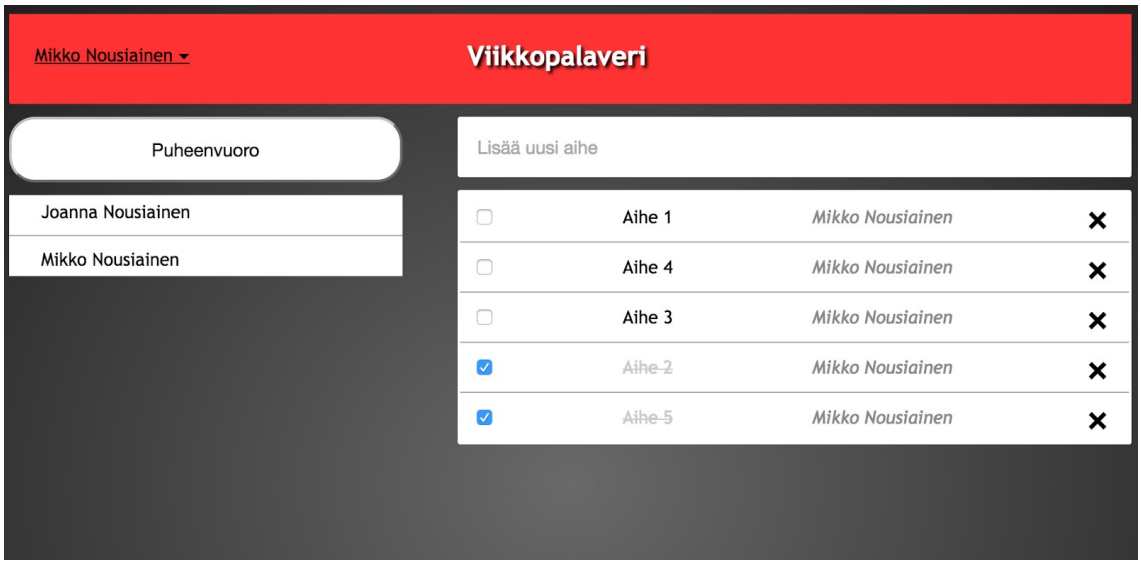
```
meteor npm install --save react react-dom
```

Esimerkkikoodi 4. React- ja ReactDOM-kirjastojen asentaminen npm-paketinhallinnasta.

Kun käyttää vipua --save, sen lisäksi, että npm asentaa paketit, se myös tallentaa asennettavien pakettien tiedot package.json-tiedostoon. Kun kaikki sovelluksen npm-paketinhallinnasta asennettavat kirjastoriippuvuudet tallennetaan package.json-tiedostoon, ovat sovelluksen riippuvuudet asennettavissa yhdellä komennolla sovelluksen lokaaliin rekisteriin. Jos kaikki riippuvuudet eivät ole määriteltyinä package.json-tiedostoon, sovellus saattaa kuitenkin toimia, jos kehittäjä on asentanut tarvittavat riippuvuudet tietokoneelleen paketinhallinnan globaaliin rekisteriin. Jos globaalisti asenne-

tut kirjastot ovat eri versioita, kuin mitä vasten sovellus on toteutettu, sovellus ei välttämättä toimi odotetulla tavalla, jos ollenkaan. Jokin toinen sovellus saattaa taas olla riippuvainen kyseisestä versiosta. Tästä syystä riippuvuudet on järkevää asentaa lokaalisti, ja siksi riippuvuuksien tietojen tulisi olla package.json-tiedostossa.

Kuvassa 7 on kuvakaappaus viikkopalaverisovelluksen MVP-version käyttöliittymästä. Sovelluksessa on yksi päänäkymä, jossa on sovelluksen kaikki toiminnallisuudet. Näkymän yläreunassa on palkki, jonka vasemmassa reunassa on linkki, jonka takaa avautuu kirjautumisvalikko. Kuvassa käyttäjä Mikko Nousiainen on sisäänkirjautuneena.



Kuva 7. Viikkopalaverisovelluksen MVP-version käyttöliittymä.

Näkymässä vasemmalla on puheenvuorojono. Joanna Nousiainen on ensimmäisenä ja Mikko Nousiainen on toisena jonossa. Puheenvuoro-painiketta painaessa, jos käyttäjällä ei ole vuoroa jonossa, lisätään käyttäjä puheenvuorojonoon viimeiseksi. Jos käyttäjällä on vuoro jonossa, vuoro poistetaan jonosta.

Näkymässä oikealla on aihelista. Ylhäällä olevaan tekstikenttään voi syöttää uuden aiheen ja painamalla Enteriä aihe lisätään listaan. Aiheen lisääminen lisää listaan uuden rivin, jossa vasemmalta oikealle on valintaruutu, josta aiheen voi asettaa käsitellyksi, itse aihe, aiheen lisääjän nimi, ja painike, joka poistaa aiheen. Listan yläpäässä ovat käsittelemättömät aiheet, alhaalla käsitellyt. Käsitellyksi asettaminen siirtää aiheen listan alaosaan.

Loin projektin juureen kansion imports ja sen sisälle alikansiot api, startup ja ui. Kansioiden, jotka on nimetty imports, sisältämiä tiedostoja ei ladata missään, vaan ne täytyy sisällyttää import-komennolla johonkin toiseen tiedostoon käyttöä varten, mikä tekee sovelluksen rakenteesta helpommin ymmärrettävää.

Kansiossa api on kaksi tiedostoa: topics.js ja turns.js, joissa määritellään aiheille ja puheenvuoroille tietokantarajapinnat. Esimerkkikoodissa 5 on tiedoston topics.js sisältämä lähdekoodi. Koodissa tuodaan mongo-kirjastosta Mongo-niminen jäsen, jonka jäseniä ovat MongoDB:n rajapintametodit, luodaan uusi kokoelma MongoDB-tietokantaan nimellä ”topics” ja viedään tämän kokoelman rajapinta muualla sovelluksessa käytettäväksi vakiossa Topics. Tiedosto turns.js on identtinen pois lukien vientiä varten luodun vakion ja tietokantaan perustettavan kokoelman nimet.

```
import { Mongo } from 'meteor/mongo';
export const Topics = new Mongo.Collection('topics');
```

Esimerkkikoodi 5. topics.js-tiedostossa oleva lähdekoodi.

Meteor tarjoaa käyttäjätiliominaisuuksia varten valmiit kirjastot. Käyttämällä kirjastoja accounts ja accounts-google sovellukseen on mahdollista rekisteröityä sovelluskohtaisella käyttäjätunnuksella, mutta sisäänkirjautuminen onnistuu myös Googlen tarjoaman OAuth-kirjautumisen kautta. Koska Fraktio käyttää muun muassa Googlen sähköpostipalvelua, kaikilla Fraktion työntekijöillä on Google-tili, joten heidän on mahdollista kirjautua sisään ilman erillistä rekisteröitymistä käyttäen Google-tiliään.

Kansiossa startup on tiedosto accounts-config.js, jossa on tiliominaisuuksien asetuksia. Meteorissa on valmiina Blaze-käyttöliittymäkirjaston kanssa käytettävä sisäänkirjautumislomakekomponentti. Kansiossa ui on tiedosto AccountsUIWrapper.jsx, joka mukauttaa tämän lomakkeen React-komponentiksi, jotta sitä olisi mahdollista käyttää React-pohjaisessa käyttöliittymätoteutuksessa.

Kansiossa ui on lisäksi muut sovelluksen React-käyttöliittymäkomponentit: App.jsx, Topic.jsx ja Turn.jsx. Topic- ja Turn-komponentit määrittävät aihelistan ja puheenvuorolistan yhden rivin ulkoasun ja toiminnallisuudet. App-komponentti on sovelluksen koostava ylimmän tason komponentti. App-komponentti ylläpitää sovelluksen tilaa reaktiivisesti eli synkronoi näkymän ja tietokannan tilan, piirtää sovelluksen käyttöliittymän käyttäen aiemmin mainittuja komponentteja ja käsittelee käyttäjän syötteet.

5 Yhteenveto ja pohdinta

Insinöörityössä tarkasteltiin moderneja JavaScript-sovelluskehyksiä: määriteltiin, mikä on moderni JavaScript-sovelluskehys, ja toteutettiin verkkosovellus sovelluskehyksellä, joka täytti tämän määritelmän.

Taustoittaakseeni tilaajaorganisaation tarpeen toteutetulle verkkosovellukselle, kuvailin tilaajaorganisaation rakennetta, toimintatapoja ja prosesseja. Viikkopalaveri on yksi tilaajaorganisaation keskeisimmistä prosesseista. Toteutetun sovelluksen tavoite oli selvittää, voiko viikkopalaveriprosessia parantaa räätälöidyllä verkkosovelluksella. Sovelluksen määrittelyn tueksi tarkastelin viikkopalveriprosessin piirteitä ja historiaa.

Määrittelin ensin, mitä kaikkia ominaisuuksia sovelluksessa voisi olla. Tämän jälkeen rajasin näistä ominaisuuksista sen joukon, joka määrittelee toiminnallisuuksiltaan suppeimman version sovelluksesta, jonka pohjalta tilaajaorganisaatio voisi päätellä, onko sovellus hyödyllinen.

Jotta voisin määritellä, mikä on moderni JavaScript-sovelluskehys, tarkastelin verkkosovellusten ja sovellusteknologioiden historiaa. Koska JavaScript oli tämän insinöörityön keskiössä, käsittelin tarkemmin sen ominaisuuksia ohjelmointikielenä.

Tarkastelin ensimmäisiä laajasti verkkosovelluskehityksessä käytettyjä JavaScript-kirjastoja ja niitä myöhemmin seuranneita varsinaisia sovelluskehyksiä. JavaScript-sovelluskehysten myötä MEAN-pino haastoi perinteisen LAMP-pinon, ja näiden erot heijastavat perinteisistä palvelinpääkeskeisistä verkkosovelluksista siirtymistä modernimpiin selainpääkeskeisiin sovelluksiin.

Meteor on eräänlainen MEAN-pinon sovellutus. Se on ennemmin sovellusalusta kuin sovelluskehys, mutta sen käyttötarkoitus on analoginen sovelluskehysten kanssa. Toin esille Meteorin ominaisuuksia, jotka vaikuttivat valintaani käyttää sitä sovelluksen prototyypin toteuttamisessa. Käsittelin myös React-kirjastoa, koska ajatukset sen taustalla edustavat modernin JavaScriptin keihäänkärkeä. Tästä syystä valitsin sen sovelluksen näkymäkerrosteknologiaksi.

Määrittelin, mitä asioita pidän modernin JavaScript-sovelluskehyksen piirteinä perustuen edeltäviin selvityksiin. Meteor-sovellusalusta React-kirjaston kanssa on teknolo-

giapinona näiden piirteiden mukainen. Kuvailin, miten toteutin sovelluksen prototyypin käyttäen valittuja teknologioita käymällä läpi prototyypin lähdekoodia.

JavaScript on nykyaikaisen internetin edellytys. JavaScriptin ansiosta on mahdollista toteuttaa enenevissä määrin toiminnallisuuksiltaan monimutkaisia ja monipuolisia verkkosovelluksia. Samanaikaisesti JavaScript-ohjelmointikielen ja JavaScript-kirjastojen ja -sovelluskehysten kehityksen myötä verkkosovellusten toteutus on myös jatkuvasti yksinkertaisempaa.

Viikkopalaverisovelluksen prototyypin toteutus vaati verrattain vähän ohjelmointityötä, koska Meteor-sovellusalustasta ja sen kanssa yhteensopivissa kirjastoissa on niin paljon valmiita toiminnallisuuksia. Esimerkiksi reaktiivisuuden toteuttaminen, käyttäjätilit käyttäen Google OAuth -tunnistautumismekanismia tai tietokannan käsittely ilman sovellusalustaa ja kirjastoja olisi huomattavasti työläämpää.

Toteutettu prototyyppi on valmis käyttöönotettavaksi. Käytön myötä tulevat ilmi sovelluksen jatkokehitystarpeet ja konkreettinen hyöty tilaajaorganisaatiolle. Prototyypistä on myös hyötyä esimerkkinä Meteor-sovelluskehyksen ja React-kirjaston mahdollisuuksista.

Lähteet

1 Suomen parhaat työpaikat: Pienet organisaatiot. 2017. Verkkodokumentti. Great Place To Work Finland. <http://www.greatplacetowork.fi/best-companies/suomen-parhaat-tyoepaikat-pienet-organisaatiot> Luettu 7.4.2017.

2 What is Three-Tier Architecture? Verkkodokumentti. Techopedia. <https:/www.techopedia.com/definition/24649/three-tier-architecture> Luettu 5.4.2017.

3 Najim, Maxime & Strimpel, Jason. 2016. Building Isomorphic JavaScript Apps: Chapter 1. Why Isomorphic JavaScript? Verkkodokumentti. <https://www.safaribooksonline.com/library/view/building-isomorphic-javascript/9781491932926/ch01.html> Luettu 5.4.2017.

4 A Short History of JavaScript. Verkkodokumentti. W3C. <https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript> Luettu 5.4.2017.

5 JavaScript: Designing a Language in 10 Days. 2012. Verkkodokumentti. IEEE Computer Society. <https://www.computer.org/csdl/mags/co/2012/02/mco2012020007.html> Luettu 6.4.2017.

6 Garret, Jesse James. 2005. Ajax: A New Approach to Web Applications. Verkkodokumentti. <http://adaptivepath.org/ideas/ajax-new-approach-web-applications/> Luettu 5.4.2017.

7 Jobs, Steve. 2010. Thoughts on Flash. Verkkodokumentti. <https://www.apple.com/hotnews/thoughts-on-flash/> Luettu 6.4.2017.

8 Is JavaScript an untyped language? 2009. Stack Overflow. Verkkodokumentti. <http://stackoverflow.com/questions/964910/is-javascript-an-untyped-language> Luettu 7.4.2017.

9 JavaScript. 2016. Verkkodokumentti. Mozilla Developers Network. <https://developer.mozilla.org/bm/docs/Web/JavaScript> Luettu 7.4.2017.

10 Chalkley, Andrew. 2014. Why jQuery is the Most Popular JavaScript Library. Verkkodokumentti. <http://blog.teamtreehouse.com/jquery-popular-javascript-library> Luettu 7.4.2017.

11 Tunru, Vincent. 2016. A short history of JavaScript frameworks: a comparison of JQuery, AngularJS and React. Verkkodokumentti. <https://vincenttunru.com/A-short-history-of-Javascript-frameworks-a-comparison-of-JQuery-AngularJS-and-React/> Luettu 6.4.2017.

12 Wayner, Peter. 2015. MEAN vs. LAMP for the future of programming. Verkkodokumentti. <http://www.infoworld.com/article/2937159/application-development/mean-vs-lamp-your-next-programming-project.html> Luettu 7.4.2017.

13 Greif, Sacha. 2014. Reactivity Basics: Meteor's Magic Demystified. Verkkodokumentti. <https://www.discovermeteor.com/blog/reactivity-basics-meteors-magic-demystified/> Luettu 7.4.2017.

14 Holas, Tomas. 2015. Navigating the React.JS Ecosystem. Verkkodokumentti. <https://www.toptal.com/react/navigating-the-react-ecosystem> Luettu 7.4.2017.

15 Introducing JSX. 2017. Verkkodokumentti. Facebook <https://facebook.github.io/react/docs/introducing-jsx.html> Luettu 7.4.2017.

16 Data Flow. Verkkodokumentti. Redux. <http://redux.js.org/docs/basics/DataFlow.html> Luettu 7.4.2017.