Tommi Pulkka

YKSISIVUISEN WEB-SOVELLUKSEN KÄYTTÖLIITTYMÄSOVELLUSKEHYKSEN VALINTAAN VAIKUTTAVAT TEKIJÄT

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2022

TIIVISTELMÄ

Pulkka, Tommi
Yksisivuisen web-sovelluksen käyttöliittymäsovelluskehyksen valintaan vaikuttavat tekijät
Jyväskylä: Jyväskylän yliopisto, 2022, 65 s.
Tietojärjestelmätiede, pro gradu tutkielma
Ohjaaja(t): Seppänen, Ville

Nykyaikaiset yksisivuiset web-sovellukset ovat nykyään tyypillinen ohjelmiston muoto, jolla pystytään korvaamaan aikaisemmin erikseen tehdyt työpöytä- ja mobiilisovellukset kustannusten minimoimiseksi. Moderneille yksisivuisille web-sovelluksille on tyypillistä, että niiden pohjalla on *sovelluskehys,* eli kirjasto, joka tarjoaa kehittäjälle lukuisia työkaluja ja valmista koodia, jonka ympärille sovellusta voi alkaa kehittämään. Nykyisin on tyypillistä, että sovelluskehykseksi valitaan suosittu ratkaisu, jotta ongelmien noustessa kehittäjän on mahdollista hakea apua sovelluskehyksen ympärille rakentuneelta yhteisöltä. On kuitenkin olemassa lukuisia suosittuja käyttöliittymän kehittämistä varten tehtyjä sovelluskehyksiä, joten kehitystiimien on hankalaa valita juuri heidän projektiinsa sopiva käyttöliittymäsovelluskehys. Tämän tutkimuksen tarkoituksena on ottaa selvää teemahaastattelujen avulla, mikä johtaa tyypillisesti käyttöliittymäsovelluskehyksen valintaan ja mitkä kriteerit koetaan tärkeimmiksi sovelluskehystä valittaessa.

Asiasanat: sovelluskehykset, front-end, Vue, React, Angular

ABSTRACT

Pulkka, Tommi
Factors leading to selection of a front-end framework of a single-page web application
Jyväskylä: University of Jyväskylä, 2022, 65 pp.
Information Systems Science, Master's Thesis
Supervisor(s): Seppänen, Ville

Modern web-applications are nowadays a typical form of software, which can be used to replace previously used separate mobile- and desktop-applications to minimize costs. It is typical for modern web applications, that they have what is called a software framework, which is a library that offers a set of tools and pre-written code, that can be used as a skeleton for the software to be developed around. Nowadays it is typical that a software framework is selected based on its' popularity among developers, so that it's easy for developers to seek help from the community that is built around the framework whenever problems arise. There are, however, several of popular front-end frameworks, so it is difficult for development teams to select the right front-end framework for their purpose. The goal of this study is to figure out, using semi-structured interviews, which are the typical reasons behind the selection of a front-end software framework and which criteria are perceived as the most important factors when choosing a front-end framework.

Keywords: front-end frameworks, front-end, Vue, React, Angular

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Web-teknologioiden innovaatioiden ansiosta sovelluskehittäjät pystyvät kehittämään nopeasti responsiivisia, mobiiliystävällisiä sovelluksia (Shahzad, 2017). Innovatiiviset web-sovelluskehykset ja kirjastot mahdollistavat sen, että samaa ohjelmakoodia voidaan käyttää perustana sekä työpöytä- että mobiilisovelluksille (Shahzad, 2017). Usein on myös tyypillistä, ettei erillisten työpöytä- tai mobiilisovellusten kehittämistä nähdä järkeväksi, vaan nojaudutaan pelkästään selainpohjaiseen, responsiiviseen web-sovellukseen kustannusten minimoimiseksi.

Web-sovellukset rakentuvat usein kolmen web-teknologian päälle, joita ovat HTML, CSS ja JavaScript (Mozilla, 2021), ja joista JavaScript on varsinainen ohjelmointikieli. JavaScript onkin kaikista suosituin selaimessa ajettava ohjelmointikieli (Graziotin & Abrahamsson, 2013; Yue & Wang, 2009). Vaikka websovellusten kehittäminen puhtaalla JavaScript-koodilla on mahdollista, se pyritään välttämään hyödyntämällä JavaScript-sovelluskehyksiä (*JavaScript Framework, JSF*). Web-sovellusta kehittäessä sovelluskehys kannattaa valita mahdollisimman aikaisessa vaiheessa, sillä sovelluskehys tuo mukanaan tiettyjä rajoitteita. Ohjelmistokehittäjät törmäävät usein ongelmiin JavaScript sovelluskehyksiä arvioidessa ja suositut verkkosivut kuten StackOverflow.com ovat täynnä aloittelijoiden kysymyksiä aiheesta. (Graziotin & Abrahamsson, 2013)

Käyttöliittymäsovelluskehykset ovat suosittuja ammatinharjoittajien keskuudessa ja ne ovat mielenkiintoinen keskustelunaihe monien ohjelmistotalojen kahvipöydissä, sekä ohjelmistokehitystä koskevat keskustelupalstat ovat pullollaan keskustelua käyttöliittymäsovelluskehyksistä ja niiden vertailuja. Käyttöliittymäsovelluskehysten suosittuudesta huolimatta ilmiöstä löytyy varsin rajallinen määrä tieteellistä tutkimusta, mikä sinänsä antaa motivaation tälle tutkimukselle.

Tämän tutkielman tarkoituksena on tutkia, mitkä tekijät vaikuttavat käyttöliittymäsovelluskehyksen valintaan. Tutkielman kirjallisuuskatsausosiossa keskitytään tutkimaan lähinnä sitä, mitkä tekijät vaikuttavat sovelluskehyksen valintaan yleisellä tasolla, sillä **käyttöliittymä**sovelluskehysten valintaa vaikuttavista tekijöistä ei löydy kovinkaan paljoa olemassa olevaa tutkimusta. Lähdekirjallisuudesta voidaan löytää malleja sovelluskehyksen valintaa vaikuttavista tekijöistä yleisellä tasolla, mutta kyseisiä malleja ei ole rajattu pelkästään

käyttöliittymäsovelluskehyksiin. Havaittavissa on näin ollen tarve päivittää kirjallisuudessa esiintyvää mallia siten, että se soveltuu nykyaikaisten käyttöliittymäsovelluskehysten kontekstiin; mitä olemassa olevista malleista voidaan pudottaa pois epäolennaisina seikkoina käyttöliittymäsovelluskehyksille ja voidaanko havaita joitain uusia, käyttöliittymäsovelluskehyksillä tärkeitä osa-alueita, joita sovelluskehyksiä arvioitaessa ei yleensä oteta huomioon. Tästä tullaan tekemään havaintoja tämän tutkimuksen empiirisessä osiossa.

Tutkielman kirjallisuuskatsausosion lähdeaineisto on kerätty pääosin Google Scholarista käyttäen hakutermejä kuten *"front-end framework"*, *"software framework"*, *"vue react angular comparison"* sekä *"JavaScript framework"*

Tämän tutkielman tutkimuskysymykset ovat:

- Mitkä eri seikat vaikuttavat ohjelmistokehitysprojektin käyttöliittymäsovelluskehyksen valintaan?
- Mitä eri mittareita käyttöliittymäsovelluskehysten vertailussa voidaan käyttää?
- Mitkä kriteerit koetaan tärkeimmiksi käyttöliittymän sovelluskehystä valittaessa?
- Miten projektin luonne vaikuttaa käyttöliittymäsovelluskehyksen valintaan?

Luvussa kaksi käydään alustus aiheeseen, jossa esitellään keskeisimpiä käsitteitä sekä yksisivuisiin web-sovelluksiin ja JavaScript käyttöliittymäsovelluskehyksiin liittyvää tietoa ja statistiikkaa. Lisäksi käydään läpi suosituimmat käyttöliittymäsovelluskehykset ja niiden tyypillisimmät erot ja yhteneväisyydet.

Luvussa kolme käydään läpi kirjallisuutta sovelluskehysten valintaa vaikuttaviin tekijöihin liittyvää kirjallisuutta, jotka kytkeytyvät toisiinsa ja Panon, Graziotinin ja Abrahamssonin (2018) malli valikoituu tämän tutkimuksen pääasialliseksi teoreettiseksi viitekehykseksi.

Luvussa neljä esitellään yhteenveto tutkielman kirjallisuuskatsausosiosta ja käydään läpi kirjallisuudesta havaittuja keskeisimpiä nostoja sekä alustetaan tutkielman empiiristä osuutta.

Tutkimuksen empiirinen osuus alkaa kappaleesta numero viisi, missä käydään läpi empiirisessä osuudessa käytetty tutkimusmenetelmä, perusteet sen valinnalle sekä esitellään teemahaastatteluun osallistuneiden haastateltavien taustat.

Kuudes luku koostuu haastattelujen läpikäynnistä, ja kuudennen kappaleen alakohdat on rakennettu vastaamaan teoreettisen viitekehyksen mallia, jota myös haastattelurunko hyvin pitkälti noudattaa. Kuudennessa luvussa esitellään myös tämän tutkimuksen teoreettisen viitekehyksen sekä empiirisen tutkimusdatan tulosten pohjalta malli käyttöliittymäsovelluskehysten valintaa vaikuttavista tekijöistä ja sovelletaan uutta mallia käyttäen tutkimukseen kerättyä empiiristä dataa.

Viimeisessä, seitsemännessä luvussa tehdään yhteenveto keskeisimmistä haastattelututkimuksen tuloksista ja pohditaan jatkotutkimuskysymyksiä, sekä esitellään teoreettiset ja käytännön implikaatiot sekä tutkimukseen liittyvät

rajoitteet. Luvussa käydään läpi uudelleen tutkielman tutkimuskysymykset ja pohditaan, miten niihin onnistuttiin vastaamaan.

2 YKSISIVUISET JAVASCRIPT WEB-SOVELLUKSET

Toisin kuin vanhanaikaisissa, monisivuisissa web-sovelluksissa, moderneissa web-sovelluksissa käytetään lähes poikkeuksetta yksisivuista lähestymistapaa. Useimmiten kehittäjät hyödyntävät yksisivuisen sovelluksen kehittämiseen erilaisia kehittäjäkirjastoja, jotka tarjoavat käyttäjälleen monipuolisia työkaluja ja valmista ohjelmakoodia. Suosituimmissa kirjastoissa on etuna myös se, että ne tarjoavat laajamittaisen yhteisön ympärilleen, tarjoten ongelmien ilmetessä kehittäjille tietolähteen mahdollisten ongelmien varalle.

2.1 Yksisivuiset ja monisivuiset web-sovellukset

Monisivuiset sovellukset (*Multi-page application, MPA*) on verkkopalvelun perinteinen muoto, jossa reitit (*routes*) on rekisteröity palvelimelle ja jokainen HTTP-kutsu palauttaa palvelimelta uuden HTML-sivun. Suurin osa ohjelmalogiikasta on tällöin palvelimella, jossa koodi ajetaan ja asiakaspää on ainoastaan haetun sivun vastaanottaja. Monisivuinen verkkopalvelu voi olla sopiva valinta, kun kehityksen kohteena on yksinkertainen sovellus, kuten esimerkiksi verkkosivusto. (Kaluža & Vukelić, 2018)

2000-luvun alussa monisivuisia sovelluksia kehitettiin esittelemällä AJAX (*Asynchronous JavaScript and XML*), joka mahdollisti vain tiettyjen sivun elementtien päivittämisen koko sivun sijaan. (Kaluža & Vukelić, 2018). Tämä oli siis ikään kuin tietynlainen modernien yksisivuisten web-sovellusten esiaste, sillä tämä mahdollisti sivun osittaisen päivittämisen sen sijaan, että koko sivun resurssit ladataan joka kerta uudelleen.

Yksisivuinen web-sovellus (S*ingle-page application, SPA*) on yksisivuinen, kokonainen web-sovellus, jonka ainoa sivu toimii runkona kaikille käyttöliittymän osille. Suurin osa resursseista (JavaScript, HTML, CSS) ladataan vain kerran, ja data siirtyy edellisestä tilasta seuraavaan. Jokainen osa ladataan dynaamisesti ja itsenäisesti lataamatta koko sivua uudestaan, antaen käyttäjälleen illuusion, että sivu olisi vaihtunut. Runko sisältää yleensä minimaalisen struktuurin ja se

on yleensä tyhjä HTML-merkki (esim.), johon loppu sovelluksen sisältö piirretään. (Kaluža & Vukelić, 2018)

Sekä Jadhav, Sawant & Deshmukh (2015) että Kaluza & Vukelić (2018) mainitsevat yksisivuisten sovellusten pääasiallisen edun verrattuna monisivuiseen sovellukseen olevan sivun latausnopeuden. SPA:ssa käyttäjän ei tarvitse ladata kaikkia ulkoisia tiedostoja kuten tyylitiedostoja ja kuvia toisin kuin MPA-mallissa. Vaikka monisivuisten sovellusten resurssien latausnopeutta on pyritty taklaamaan erilaisilla välimuistituksilla, ei latausnopeudet yllä yksisivuisten sovellusten tasolle.

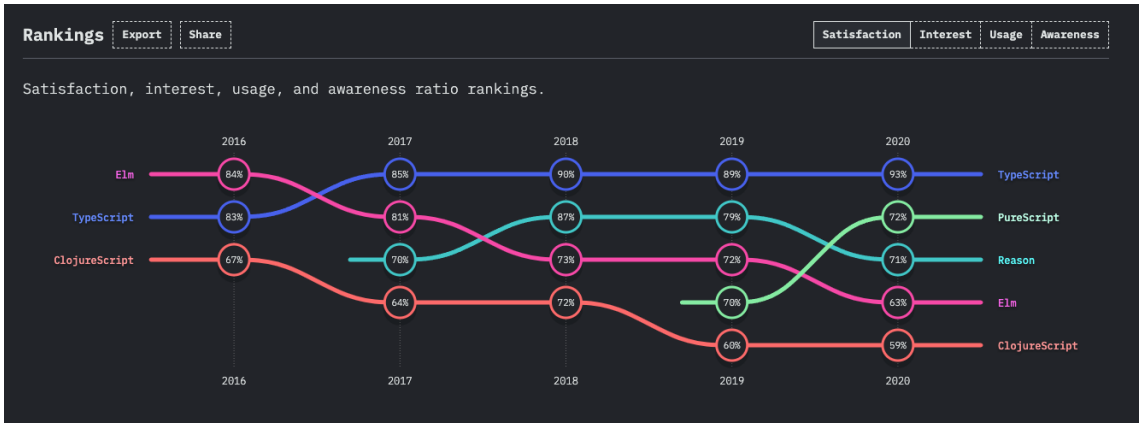
Kaluza & Vukelić (2018) ovat listanneet tutkimuksessaan yksisivuisten sovellusten tarjoamat edut verrattuna monisivuisiin sovelluksiin. Heidän havainnoistaan on koostettu yhteenveto alla olevaan taulukkoon.

TAULUKKO 1 Yksisivuisen sovelluksen tarjoamat edut käännettynä suomeksi (mukaillen Kaluza & Vukelić (2018))

Nopea vasteaika	Koska suurin osa sivusta ladataan kerralla, asiakaspäästä ei tarvitse tehdä jatkuvasti pyyntöjä, jotta uutta tietoa saadaan ladattua palvelimelta. Kun käyttäjä klikkaa aluetta sivulla, muutokset tehdään välittömästi, mikä antaa käyttäjälle interaktiivisen kuvan.
Esittämisen erottaminen toimintalogiikasta	Käyttöliittymäkoodi voidaan pitää selkeästi erillään asiakkaan puolella (*client-side*). Kehittäjä voi keskittyä käyttöliittymäkoodiin erikseen ja pitää sovelluksen bisneslogiikan palvelinpäässä. (*server-side*)
Nopeampi ja helpompi tiedonsiirto	Tapahtumat palvelimen kanssa ovat helpompia ja nopeampia koska alustavan latauksen jälkeen pelkästään dataa lähetetään ja vastaanotetaan palvelimelta
Mahdollisuus offline-tukeen	Koska yksisivuisissa sovelluksissa JavaScript ajetaan asiakkaan päässä, jatkuvaa yhteyttä palvelimeen ei tarvita. Palvelu jotakuinkin toimii, tai ainakin käyttäjälle voidaan antaa tällainen illuusio, mutta uutta dataa ei haeta palvelimelta

2.2 TypeScript

Biermanin, Abadin ja Torgersenin mukaan (2014) JavaScript on suosiostaan huolimatta huono kieli laajamittaiseen sovelluskehitykseen. JavaScriptiin on olemassa laajennus, TypeScript, joka tuo JavaScript-sovellukselle muista ohjelmointikielistä tuttuja ominaisuuksia kuten luokat, rajapintaluokat sekä staattisen tyypityksen (Bierman ym., 2014). Edellä mainitut TypeScriptin tarjoamat lukuisat edut huomioon ottaen, on käyttöliittymäsovelluskehyksiä arvioidessa syytä ottaa huomioon sovelluskehyksen TypeScript -yhteensopivuus.



KUVIO 1 TypeScriptin tyytyväisyys kehittäjien keskuudessa (State of JavaScript, 2021)

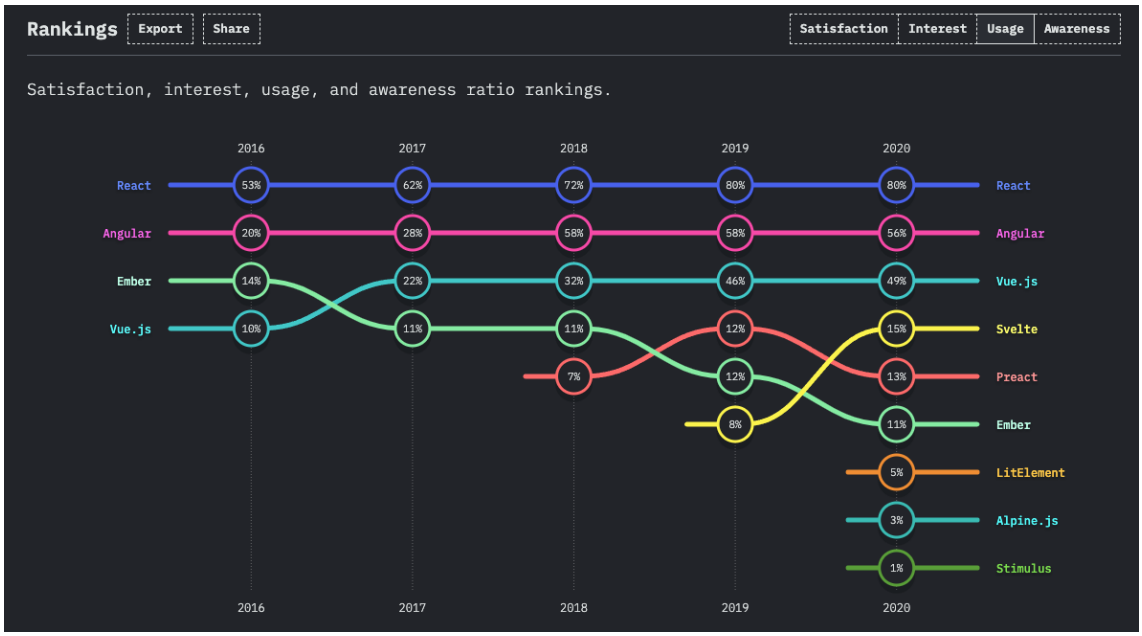
2.3 JavaScript-sovelluskehykset

Sovelluskehys on sovellus, jossa olemassa olevan koodin on tarkoitus tarjota yleisimpiä toiminnallisuuksia. Sovelluskehykset voivat sisältää muun muassa työkaluja, apuohjelmia, koodikirjastoja, rajapintoja tai kooditulkkeja, jotka on testattu eri alustoilla ja selaimilla (C. M. Novac ym., 2021; Pano ym., 2018). Tässä tutkielmassa käytettävää termiä ”käyttöliittymäsovelluskehys” (engl. *front-end framework*) tarkoitetaan käyttöliittymän kehittämistä varten kehitettyä sovelluskehystä.

Graziotinin ja Abrahamssonin (2013) mukaan JavaScript -sovelluskehykset tarjoavat monia hyötyjä kuten monimutkaisten ohjelmointivaiheiden välistä jättämisen sekä useimpien selainten välisten yhteensopivuusongelmien karsiutumisen. Lukuun ottamatta suorituskykyjen mittauksia ja asiantuntijoiden suosituksia, on olemassa vain vähän tutkimustietoa, joka auttaa kehittäjiä valitsemaan kaikista sopivimman sovelluskehyksen kuhunkin sopivaan tilanteeseen (Graziotin & Abrahamsson, 2013).

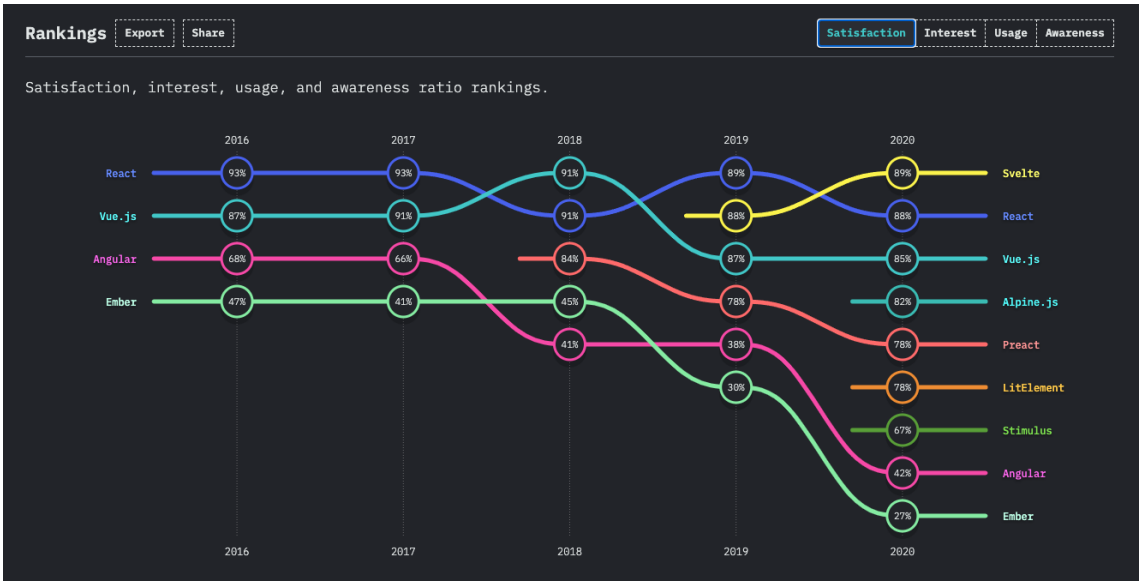
2.4 Suosituimmat JavaScript käyttöliittymäsovelluskehykset

State of JavaScript (2020) -kyselyt ovat JavaScript -ohjelmoijille teetettyjä kyselyitä, joilla on pyritty selvittämään JavaScriptin ympärille muodostuneen kehittäjäyhteisön tilaa sisältäen muun muassa kehittäjien mielipiteet eri JavaScriptin ominaisuuksiin, viitekehyksiin ja laajennuksiin. Kysely sisältää myös dataa sovelluskehyksistä, mikä on kiinnostavaa tämän tutkimuksen osalta. Kyselyssä eri sovelluskehyksiä vertailtiin neljällä eri kriteerillä, tyytyväisyys (*satisfaction*), kiinnostavuus (*interest),* käyttö (usage) sekä tietoisuus (*awareness*).



KUVIO 2 JavaScript -sovelluskehysten suosittuus käytön mukaan (State of JavaScript, 2020)

Kuvio 2 on kuvakaappaus State of JavaScript 2020 -sivustolta, josta käy ilmi, että React on ollut ylivoimaisesti käytetyin sovelluskehys vuodesta 2016 aina vuoteen 2020 saakka. Muita suosituimpia sovelluskehyksiä ovat Angular, Vue.js sekä uusi tulokas, Svelte. Alun perin suosittu ja ensimmäisten joukossa julkaistu Ember on menettänyt suosiotaan, kun taas muilla suosituilla sovelluskehyksillä trendi on ollut päinvastainen, tai vähintäänkin suosio on pysynyt jotakuinkin samana. Myös Angularin suosio on alkanut heikentymään viime vuosina.



KUVIO 3 JavaScript-sovelluskehysten vertailu kehittäjien tyytyväisyyden mukaan (State of JavaScript, 2020)

Näiden kahden sovelluskehyksen suosittuuden heikkenemistä voidaan selittää kehittäjien alhaisella tyytyväisyydellä sovelluskehykseen, jota havainnollistetaan kuviossa 3. Svelten räjähdysmäistä suosioita puolestaan voidaan selittää katsomalla kuvion 3 tyytyväisyysmittaria, jossa Svelte on kärkisijalla vuoden 2020 tuloksissa.

Tässä tutkimuksessa käsitellään lähinnä kolmea tällä hetkellä suosituinta käyttöliittymäsovelluskehystä, joita ovat React, Angular sekä Vue.js. Reactista erityisen kiinnostavan tekee se, kuinka se on pystynyt säilyttämään valta-asemansa niin kauan. Angularin kiinnostavuus kumpuaa puolestaan siitä, mitä on tehty väärin tai mitä ominaisuuksia on jätetty sovelluskehyksestä pois johtaen alhaiseen tyytyväisyyteen. Vue.js on selvästi Reactin ja Angularin rinnalla yksi suurimmista sovelluskehyksistä tuoreudestaan huolimatta

2.4.1 React

React kehitettiin alun perin Facebookia ja Instagramia varten paremman käyttäjäkokemuksen saavuttamiseksi. Reactin tehokkaiden ominaisuuksien vuoksi, Facebook (nyk. Meta) päätti julkaista Reactin avoimen lähdekoodin kirjastona. (Xing, Huang & Lai, 2019).

Molemmissa aikaisemmin esitellyissä State of JavaScript (2020) -sivustoilta kuvakaapatuista kuvista, React on sijoittautunut, ja sijoittuu yhä edelleen, yläsijoille. Vuonna 2020 80 % State of JavaScript -kyselyyn vastanneista kehittäjistä olivat käyttäneet Reactia. React on Facebookin (nyk. Meta) kehittämä JavaScript kirjasto käyttöliittymien rakentamista varten, jota kuvataan sen kehittäjäsivuillaan seuraavasti:

React on:

- D*eklaratiivinen*: React tekee interaktiivisten käyttöliittymien rakentamisesta vaivatonta. Jokaiselle sovelluksen tilalle voidaan suunnitella yksinkertaiset näkymät ja React päivittää ja piirtää tehokkaasti juuri oikeat komponentit riippuen siitä, mitä dataa päivitetään. Deklaratiiviset näkymät tekevät koodista ennalta-arvattavampaa ja helpommin ymmärrettävää, sekä tekee virheenetsinnästä helpompaa.
- *Komponenttipohjainen:* React mahdollistaa kapseloitujen komponenttien rakentamisen, jotka hallinnoivat oman tilansa, joita voidaan sijoittaa mihin vain sovelluksessa monimutkaisten käyttöliittymiä rakentaessa.
- *Riippumattomuus:* React ei ota kantaa käyttöliittymän ulkopuolisesta teknologiapinoon, joten on mahdollista kehittää uusia toiminnallisuuksia Reactissa kirjoittamatta uudelleen olemassa olevaa koodia. React tarjoaa myös palvelinpuolen renderöinnin Noden avulla sekä tuen mobiilisovellusten kehittämiselle React Nativen avulla.

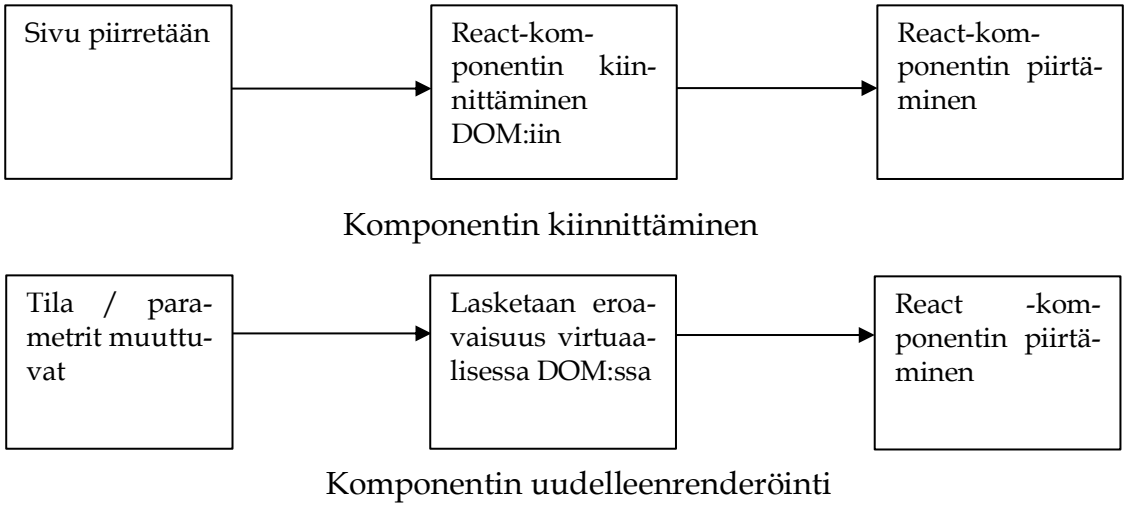
(Meta, 2021)

Aggarwal (2018) listaa artikkelissaan Reactin keskeisimpiä ominaisuuksia:

- *Kevyt DOM (Document Object Model, dokumenttiobjektimalli) paremman suorituskyvyn saavuttamiseksi*: React tarjoaa tehokkaan ja kevyen dokumenttiobjektimallin. React ei ole vuorovaikutuksessa selaimen generoiman DOM:n kanssa, vaan reagoi muistiin tallennetun DOM:n kanssa. Tätä kutsutaan virtuaaliseksi DOM:ksi, jota React vertailee varsinaiseen DOMiin diff()-algoritmin avulla, mahdollistaen sen, että vain muuttuneet osat päivitetään dokumenttiobjektimalliin.
- *Helppo oppimiskäyrä*: React on luonnoltaan helppo ja monimutkaton, tehden siitä nopeasti opittavan.
- *JSX*: JSX on kieli, joka on hyvin samanlainen kuin suosittu XML. JSX:n käyttäminen ei ole välttämätöntä React-sovellusten kehittämisessä, mutta sen käyttäminen on suosittua, sillä sen hyödyntäminen tekee kehittämisestä helpompaa ja nopeampaa.
- *Suorituskyky ja virtuaalinen DOM*: React on tunnetusti tehokas sovelluskehys ja se on yksi avaintekijöistä Reactin suosion takana. Suorituskykykyyden takana on pääasiallisena syynä virtuaalinen DOM.

Aggarwal (2018) listaa myös muutamia Reactiin liittyviä rajoitteita:

- React käsittää vain MVC-arkkitehtuurimallin näkymä (*view*)-osion, jolloin kokonaista sovellusta ei pysty tekemällä ainoastaan Reactilla, vaan projektiin täytyy sisällyttää myös muita työkaluja.
- JSX-sapluunoiden käyttö voi olla hankalaa ja monimutkaista joillekin kehittäjille.
- Virheet ohjelmakoodissa tapahtuvat koodia kääntäessä eikä ajettaessa

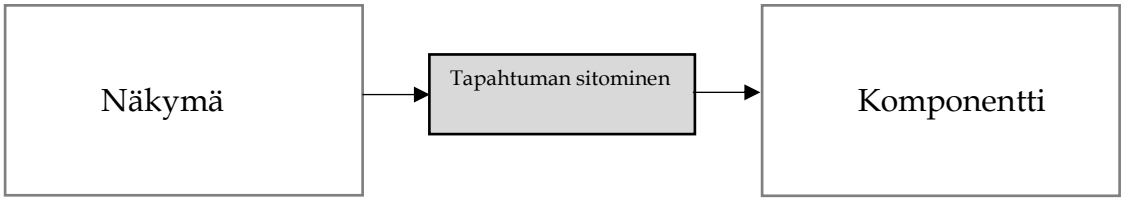


KUVIO 4 Komponenttien kiinnittyminen (*mount*) ja uudelleenrenderöinti, käännetty suomeksi (Xing, ym. 2019))

Oheinen Xingin ym. (2019) kuvaaja (KUVIO 4) havainnollistaa, kuinka React piirtää käyttöliittymäsivun. React kiinnittää verkkosivun sisällön erilaisina komponentteina DOM:iin. Selain piirtää komponentit käyttäjälle näkyväksi JavaScriptin avulla. Chrome V8 teknologian ansiosta JavaScriptin renderöintinopeus on perinteisiä, dynaamisia verkkosivuja nopeampi. (Xing ym., 2019)

2.4.2 Angular

Angular on suosittu Googlen vuonna 2010 kehittämä JavaScript ES5-pohjainen avoimen lähdekoodin käyttöliittymäsovelluskehys. Alkuperäisellä Angularilla on paljon rajoitteita sen alkuperäisen suunnittelumallin vuoksi verrattuna muihin käyttöliittymäsovelluskehyksiin. Angular tehtiin käytännössä kokonaan uusiksi vuonna 2016 Google kehitystiimin toimesta ja se nimettiin Angular 2:ksi. (Xing ym., 2019). Nykyään puhuttaessa Angularista viitataankin yleensä Angularin moderniin versioon. Myös tässä tutkielmassa puhuttaessa Angularista, viitataan Angularin moderniin versioon. Angularin uusin versio on versio 13 (Angular, 2021), eli sovelluskehykseen on tehty suuri määrä päivityksiä, ja sen elinkaaresta pidetään huolta.



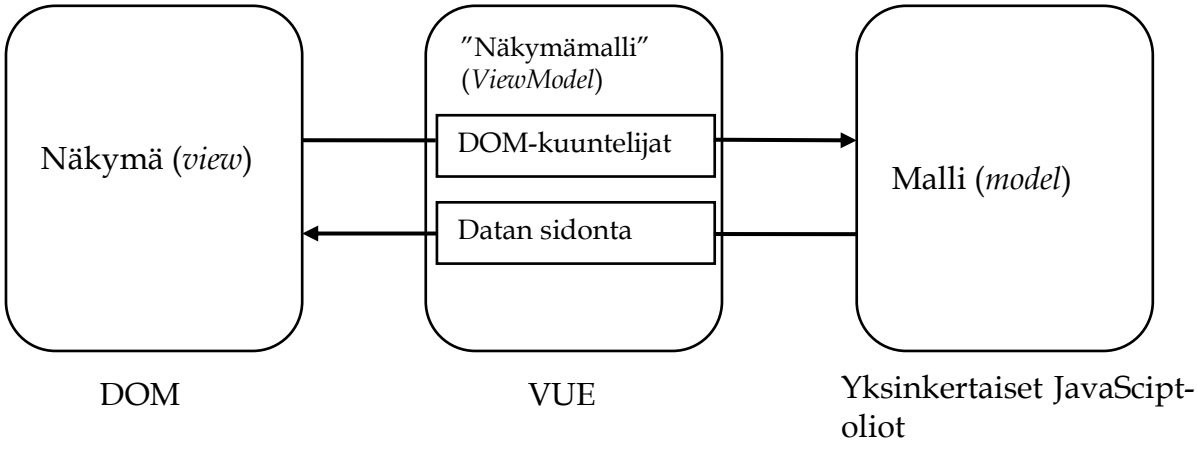
KUVIO 5 Angularin yksisuuntainen sitominen (Xing ym., 2019) (käännetty suomeksi)

Alkuperäinen Angular 1 käyttää sapluunaohjeistusta (*template directive*) ja kontrolleria (*controller*), jotka Angularin moderni versio korvaa uudella moduulilla, jota kutsutaan komponentiksi, joka yhdistää molemmat aikaisemmin mainitut osat. Toisena erona Angularin alkuperäiseen version on se, että uudemmissa versioissa Angular pohjautuu TypeScriptiin JavaScriptin sijaan. (Xing, ym. 2019). TypeScript-natiiviuuden voidaan katsoa olevan merkittävä etu verrattuna ei-TypeScript-natiiviin käyttöliittymäsovelluskehykseen.

Satrom (2018) kuvaa Angularia ”all-in-one”-tyyppisenä ratkaisuna, joka sisältää kaiken tarvittavan monimutkaisen sovelluksen kehittämiseksi. Myös tämä voidaan nähdä joissakin tapauksissa, riippuen projektin luonteesta, negatiivisena asiana, sillä liika valmis koodi voi tehdä sovelluskehyksestä monimutkaisen. Tätä asiaa sivutaan lisää tutkielman empiirisessä osuudessa.

2.4.3 Vue

Vue.js on suosittu EcmaScript6 -pohjainen avoimen lähdekoodin JavaScript käyttöliittymäsovelluskehys, jonka on kehittänyt alun perin Evan You vuonna 2014. Vuen alkuperäinen tarkoitus oli saavuttaa responsiivinen datan sidonta sekä käyttöliittymäkomponentit helpolla ohjelmointirajapinnalla. (Xing, ym. 2019)
Xing ym. (2019) havainnollistavat kuvion 6 avulla kuinka Vue.js sisältää kolme osiota datavetoiseen prosessointiin: Näkymän (View), näkymämallin (View Model) sekä mallin (Model). Näkymä-alue käsittää DOM:n, joka näyttää verkkosivun sisällön.



Näkymämalliosio (*View Model*) sisältää DOM-kuuntelijat sekä datan sidonnan. Kun käyttäjä datan prosessoinnin näkymässä, Vue käyttää DOM-kuuntelijoita tarkkailemaan ja päivittämään malliosion datan. Kun malliosion data on päivittynyt, Vue käyttää DOM-sidontaa verkkopalvelun sisällön päivittämiseen. Vue käyttää siis yhdensuuntaista sidontaa DOM-kuuntelijoilla saavuttaakseen kaksisuuntaisen sidonnan (Xing ym., 2019).

Tässä tutkielmassa käsiteltävistä sovelluskehyksistä Vue on ainut, jonka taustalla ei ole tunnettua organisaatiota, toisin kuin Reactin ja Angularin tapauksissa taustalla on Facebook ja Google. Tutkielman empiirisessä osuudessa tullaan sivuamaan tätä ja nähdään, koetaanko taustaorganisaation tuoma turva tärkeäksi

KUVIO 6 Vue.js datavetoinen konsepti (Xing, ym. 2019) (käännetty suomeksi)

2.5 Käyttöliittymäsovelluskehysten erityispiirteet, yhteneväisyydet ja pääasialliset erot

Kaikki tässä tutkielmassa esitellyt käyttöliittymäsovelluskehykset ovat erittäin suosittuja ja arvostettuja. Vaikka käyttöliittymäsovelluskehykset ovat tehty samaa tarkoitusta varten ja niiden avulla voidaan tehdä käytännössä täysin identtisiä käyttöliittymiä riippumatta siitä, mikä sovelluskehys on käytössä, on käyttöliittymäsovelluskehyksien välillä huimia eroja kehittäjänäkökulmasta.

TAULUKKO 2 Datan sidonta ja ohjelmointikielet eri käyttöliittymäsovelluskehyksissä, käännetty suomeksi (mukaillen Xing ym., 2019)

	Angular	**React**	**Vue**
Datan sidonta (*Data Binding*)	yksi- ja kaksisuuntainen	Yksisuuntainen	Yksi / kaksisuuntainen
Pohjautuu ohjelmointikieleen	TypeScript	JavaScript ES6	JavaScript ES6

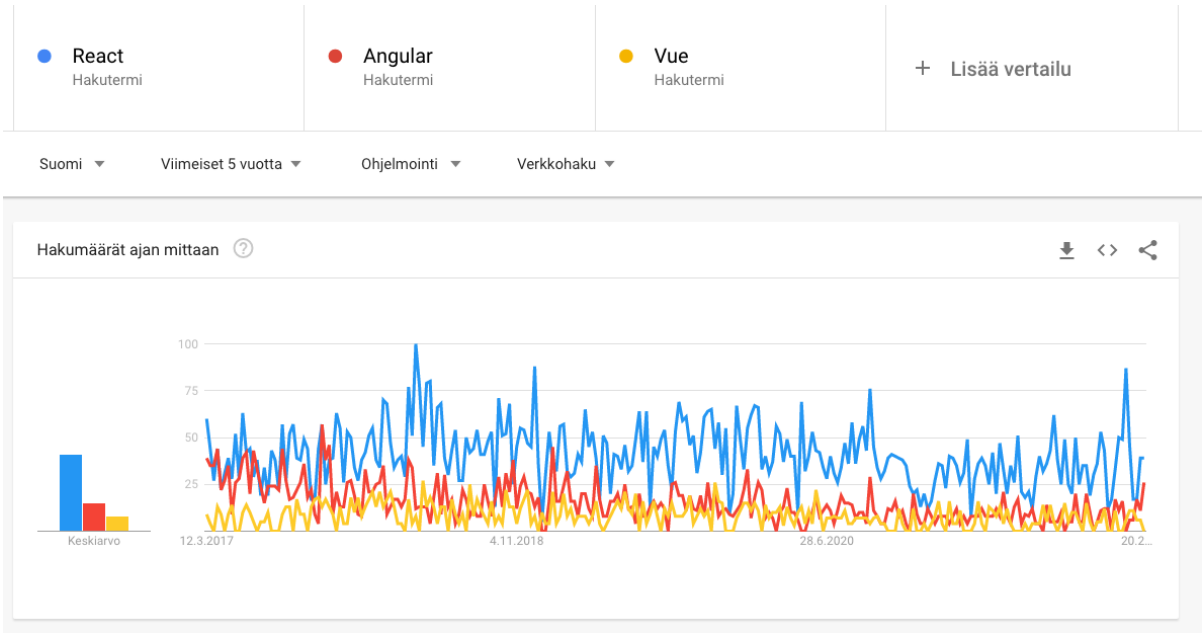
2.5.1 Suosittuus

React on C. M. Novacin ym. (2021) mukaan Vue'ta suositumpi sovelluskehys. Facebook (nyk. Meta) hallinnoi ja kehittää Reactia, kun taas Vue ylläpidetään avoimen lähdekoodin ja joukkoistamisen voimin. (C. M. Novac ym., 2021; O. C. Novac ym., 2021) Suosittuutta ja sen vaikutusta sovelluskehyksen valintaan käydään lisää tämän tutkimuksen empiirisessä osuudessa kappaleessa kuusi. Reactin ja Vuen suosittuutta on käytännössä mahdotonta verrata, sillä Vue erittäin suosittu Kiinassa, ja useimmat suosittuusvertailut pohjautuvat Google-hakuihin, minkä käyttö on kielletty Kiinassa. Tätä asiaa sivutaan lisää alaluvussa 2.5.3.

2.5.2 Käyttöönotto

Koska Vue tarjoaa graafisen käyttöliittymän käyttöönottoa varten, on sen käyttöönotto Reactiin verrattuna helpompaa. (C. M. Novac ym., 2021). Vuen helppokäyttöisyys ja käyttöönoton helppous onkin sen yksi vahvimmista puolista ja tyypillisimpiä sen valintaan johtavista tekijöistä, josta kerrotaan lisää luvussa 3.3. sekä tämän tutkimuksen empiirisessä osiossa. Satromin (2018) mukaan Vue on käyttöliittymäsovelluskehyksistä helpoin ottaa käyttöön ja Angular on puolestaan vaikein.

2.5.3 Maantieteelliset erot



KUVIO 7 React vs Angular vs Vue Google-hakujen vertailu (Google Trends, 2022)

C. M. Novacin ym. (2021) mukaan useat lähteet osoittavat, että React on Vue'ta suositumpi käyttöliittymäsovelluskehys, mutta hän kuitenkin toteaa, että koska kiinalaiset kehittäjät eivät juurikaan käytä Googlea, ovat tilastot hämääviä. C. M Novac ym. (2021) toteavat, että kiinalainen kehittäjä valitsee tyypillisesti Vuen Reactin sijaan. Sovelluskehysten todellista suosittuutta on siis mahdoton verrata maantieteellisten erojen vuoksi. Google Trendsin (2022) (KUVIO 7) mukaan React vaikuttaisi olevan kaikista suosituin käyttöliittymäsovelluskehys, mutta Gitstar-ranking-palvelun, (2022) joka arvottaa Github-arkistot (*repository*) käyttäjien antamien tähtien (*stars*) perusteella, puolestaan Vue on käyttöliittymäsovelluskehysten osalta kärjessä.

Syitä Vuen suosittuudelle Kiinassa on pohdittu useilla eri internetin keskustelupalstoilla ja blogeissa, ja usein syiksi osoittautuvat Vuen kehittäjä Evan Youn Amerikan-kiinalaistausta ja alusta asti kiinaksi kirjoitettu dokumentaatio. Lisäksi Reactin ja Angularin taustalla olevat palvelut ja yritykset Googlen ja

Facebookin käyttö on estetty Kiinassa, joten tämäkin luonnollisesti vaikuttaa sovelluskehyksen valintaan. (Quora, 2022, Vue-View, 2022, Williams, 2022)

2.5.4 Yhteenveto käyttöliittymäsovelluskehysten välisistä eroista

Taulukossa 3 on listattu Vuen, Angularin ja Reactin välisiä pääasiallisia eroja, jotka ovat pääosin peräisin O.C Novacin ym. (2021) tutkimuksesta.

TAULUKKO 3 Suosituimpien sovelluskehysten pääasialliset erot soveltaen O.C. Novacin ym. (2021) taulukkoa, React-osio täydennetty muista lähteistä

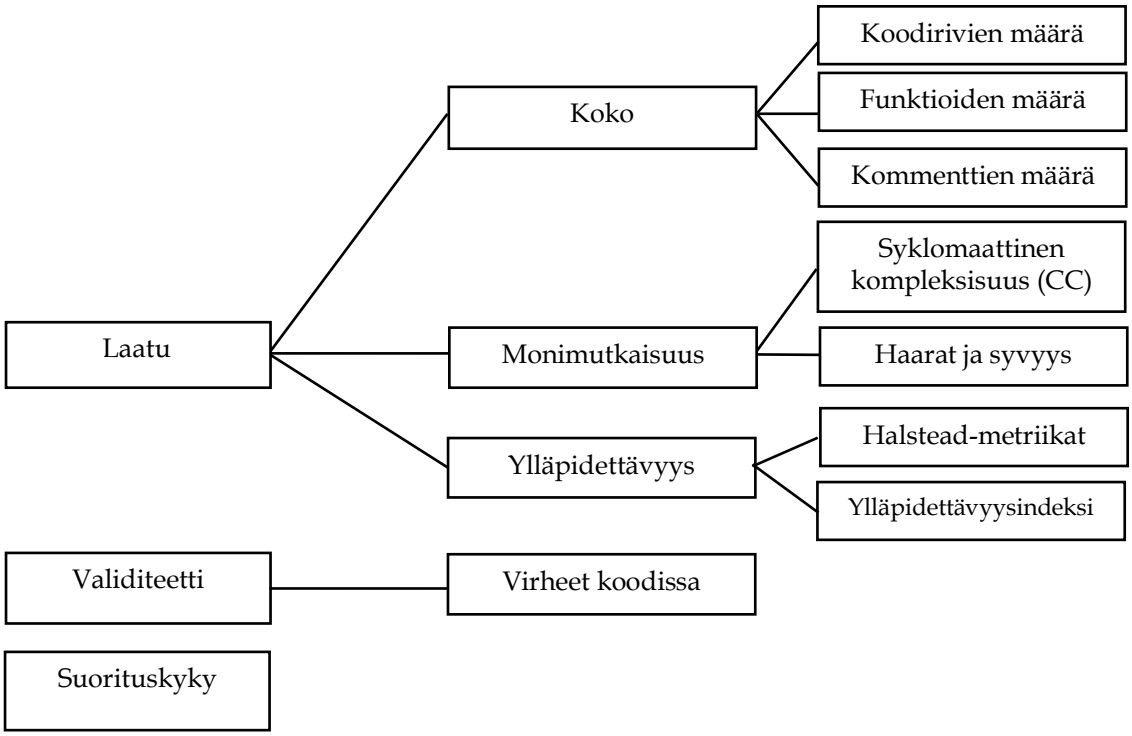
	Vue.js	Angular	React
Julkaistu	Helmikuu 2014	Syyskuu 2016	2013 (Xing ym., 2019)
Sovelluskehystä käyttävät yritykset/palvelut	GitLab, AliBaba, Baidu, ym.	Forbes, weather.com, Wix, Google, ym.	Netflix, Facebook, Instagram (C. M. Novac ym., 2021)
Perustajat	Evan You	Google	Facebook (nyk. Meta)
Sovellustyypit	Kehittyneet yksisivuiset sovellukset, natiivisovellukset	Natiivisovellusten kehittäminen, hybridisovellukset, web-sovellukset	Alustasta riippumattomat web-sovellukset (Novac ym., 2021)
Soveltuu täydellisesti	Yksisivuisten- ja web-sovellusten kehittämiseen	Monipuoliset sovellukset	Mobiilisovellukset ja monipuoliset sovellukset
Yhteisön tuki	Avoimen lähdekoodin projekti	Suuri kehittäjäyhteisö	Suuri kehittäjäyhteisö
GitHub-tähdet	191 000 (GitHub, 2021a)	78 100 (GitHub, 2021b)	179 000 (GitHub, 2021c)
Kieli-preferenssi	HTML-sapluunat ja JavaScript	TypeScript	JSX-sapluunat ja JavaScript (Aggarwal, 2018)
Kirjoitettu kielellä	JavaScript	TypeScript	JavaScript (C. M. Novac ym., 2021)

3 KÄYTTÖLIITTYMÄSOVELLUSKEHYKSEN VALINTAAN VAIKUTTAVAT TEKIJÄT

Käyttöliittymäsovelluskehystä valittaessa, on tärkeää valita laadukas ja suorituskyvyltään tehokas sovelluskehys. Sovelluskehystä valittaessa tulee ottaa huomioon ennen kaikkea sen parissa työskentelevät kehittäjät, sillä heidän arjessaan valinta tulee näkymään kaikista eniten. Sovelluskehyksen vaihtaminen jälkikäteen on vaativaa ja aikaa vievää, joten sen valintaan kannattaa nähdä vaivaa.

3.1 Sovelluskehysten arvioinnin mittarit

Gizas, Chrostodoulou ja Papatheodorou (2012) käyttävät tutkimuksessaan JavaScript-sovelluskehysten arviointiin mittareina laatua (*quality*), validiutta (*validity*) ja suorituskykyä (*performance*). Kaikki Gizasin ym. (2012) tutkimuksessa tarkastellut sovelluskehykset ovat JavaScript-käyttöliittymäsovelluskehyksiä, mutta ne ovat nykytilanteeseen nähden vanhentuneita teknologioita, eikä näin ollen enää nykyisellään niin laajasti käytettyjä, joten itse tutkimuksen tulokset eivät ole tämän tutkimuksen osalta kovinkaan relevantteja. Gizas ym. (2012) eivät pyrkineet tutkimuksessaan varsinaisesti löytämään parasta sovelluskehystä, vaan ennemminkin löytämään kehityskohtia tarkasteltuihin sovelluskehyksiin. Tämän tutkimuksen kannalta meitä kiinnostaa lähinnä se, *miten* Gizas ym. (2012) arvioivat ja mittasivat eri sovelluskehyksiä.



KUVIO 8 Sovelluskehysten arviointiin käytetyt mittarit Gizasin ym. (2012) tutkimuksessa

Kuten aiemmin mainittiin, Gizas ym. (2012) käyttävät sovelluskehysten arviointiin mittareina laatua, validiutta ja suorituskykyä. Tutkijat pilkkovat laadun kolmeen eri mittariin: kokoon (size), monimutkaisuuteen (complexity) sekä ylläpidettävyyteen (maintainability).

Gizas ym. (2012) jakavat koon kolmeen eri mitattavaan alakategoriaan: koodirivien, funktioiden sekä kommenttien määrään sekä niiden väliseen suhteeseen. Monimutkaisuuden mittareina he puolestaan käyttävät McCaben (1976) syklomaattista kompleksisuutta (*cyclomatic complexity, CC*), joka on matemaattinen malli monimutkaisuuden mittaamiseen. Ylläpidettävyyden mittaamiseen tutkijat käyttivät ylläpidettävyysindeksiä, joka saadaan sijoittamalla Halstead-metriikka (Halstead, 1977), syklomaattinen kompleksisuus, koodirivien määrä sekä kommenttien määrä laskukaavaan (Welker, 2001; Gizas ym., 2012). Laadun mittarit ovat siis Gizasin ym. (2012) mallissa tiukasti nidottu toisiinsa, eli koko ja monimutkaisuus vaikuttavat ylläpidettävyyteen. Validiutta he mittaavat koodivirheiden määrällä käyttämällä apuna Lint- sekä Yasca-työkaluja. Suorituskykyä he puolestaan mittaavat suoritusajalla, joka saadaan käyttäen apuna SlickSpeed Selectors -testisovelluskehystä.

Vaikka Gizas ym. (2012) sisällyttävät laajalti eri sovelluskehysten osa-alueet arviontiin, he eivät ota kantaa sovelluskehyksen ympärillä olevaa ekosysteemiä. Tähän Graziotin ja Abrahamsson (2013) ehdottavat laajennusta, joka käsittäisi validoinnin, laadun ja suorituskyvyn lisäksi myös dokumentaation, yhteisön sekä käytännöllisyyden (*pragmatics*). Graziotin ja Abrahamsson (2013) olivat ensimmäisiä, jotka tutkivat sovelluskehysten sosiaalista puolta sovelluskehyksen valintaan vaikuttavana tekijänä, mikä on mielestäni varsin mullistava löydös.

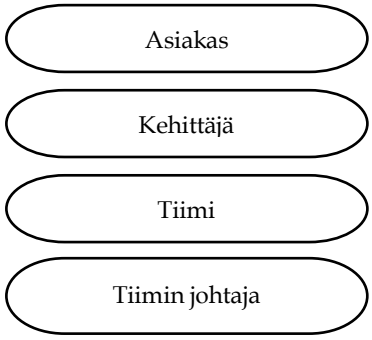
Graziotinin ja Abrahamssonin (2013) lisäys sovelluskehyksen ympärillä olevasta ekosysteemistä on varsin perusteltu ja onkin nykyisin yksi merkittävimmistä tekijöistä sovelluskehystä valittaessa. Graziotin ja Abrahamsson tarkentavat ja laajentavat tätä teoriaa yhdessä Panon kanssa, jota käydään tarkemmin läpi seuraavassa kappaleessa. (Pano, Graziotin & Abrahamsson, 2018)

3.2 Sovelluskehyksen valintaan vaikuttavat tekijät

Pano, Graziotin ja Abrahamsson (2018) tutkivat tutkimuksessaan JavaScript -sovelluskehyksen valintaan vaikuttavia tekijöitä ja he kehittivät UTAUT:ia (*Unified Theory of Acceptance and Use of Technology*) (Venkatesh, Morris, Davis, G. B. ja Davis, F. D, 2003) soveltaen mallin JavaScript-sovelluskehysten omaksumiseen johtavista tekijöistä. Venkateshin ym. (2003) malli teknologian hyväksymistä ja käyttämisestä on kaikista laajimmille levinnyt teknologian hyväksymistä käsittelevämalli. Tämän tutkielman kirjoitushetkellä Venkateshin ym. (2003) artikkeliin on viitattu jo yli 37 000 kertaa.

Panon ym. (2018) malli sovelluskehysten valintaan vaikuttavista tekijöistä lienee laajimmin käytetty malli sovelluskehysten hyväksymiselle, sillä kyseiseen tutkimukseen viitataan suuressa osassa aihealuetta käsittelevissä artikkeleissa, jotka ovat määrältään suhteellisen vähäisiä. Vaikka ilmiötä ei ole tutkittu paljoakaan, tämä malli tarjoaa hyvän pohjan tämän tutkimuksen empiiriselle osuudelle sekä muulle tulevaisuuden tutkimukselle mikä käsittelee sovelluskehysten valintaan vaikuttavia tekijöitä.

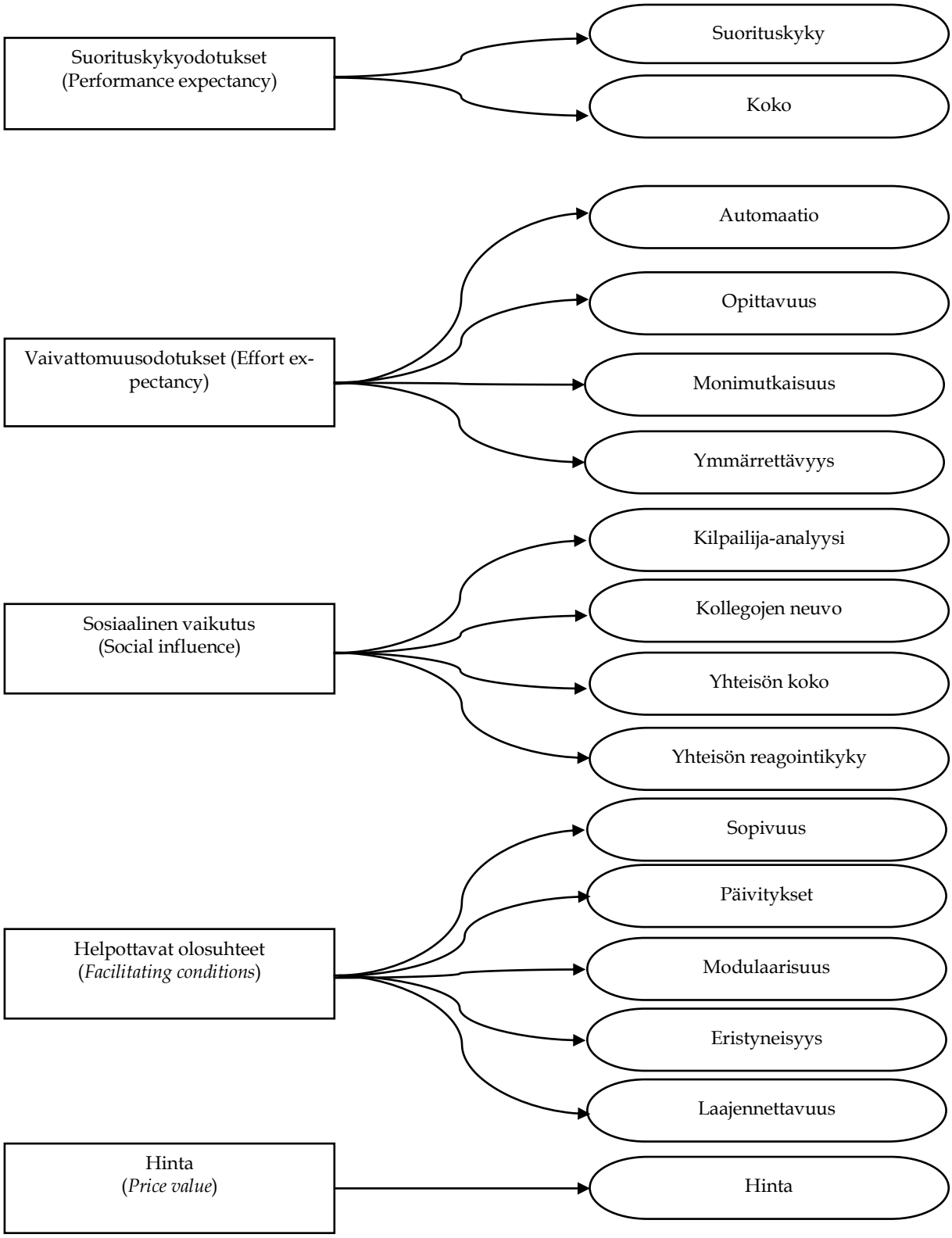
Vaikka malli on varsin perusteltu ja tarjoaa erinomaisen pohjan tutkimukselle, se on tietyissä määrin vanhentunut, sillä tutkimuksen empiirinen data kerättiin jo vuonna 2014 ja nykyisin pinnalla on aivan eri sovelluskehykset kuin tuolloin. Vain muutama tutkimukseen osallistuja mainitsi modernin teknologian (tuolloin React ja AngularJS). Mallia on siis syytä testata nykyaikaisten teknologioiden kontekstissa. Lisäksi on kiinnostavaa nähdä, onko kaikki mallin osa-alueet olennaisia, kun tarkastelun kohteena on pelkästään käyttöliittymäsovelluskehykset; tarvitseeko mallia päivittää, voidaanko siitä pudottaa joitakin osa-alueita pois, tai tuovatko modernit käyttöliittymäsovelluskehykset malliin joitakin lisäkohtia, jotka kaipaavat huomiointia?



KUVIO 6 Panon, Graziotinin ja Abrahamssonin (2018) malli JavaScript sovelluskehyksen hyväksymiseen vaikuttavista päätöksentekijöistä

Pano ym. (2018) toteavat, että sovelluskehyksen valinnassa on neljä muuttujaa, jotka tulee ottaa huomioon (KUVIO 6), jotka ovat asiakas, kehittäjä(t), tiimi sekä tiimin johtaja. Käyttöliittymäsovelluskehysten osalta on mielenkiintoista nähdä, onko mukana sama määrä toimijoita valitsemassa sovelluskehystä, vai päättääkö valinnasta suppeampi ryhmä päätöksentekijöitä.

Käytännön ammatinharjoittajat voivat hyötyä Panon ym. (2018) mallista, sillä malli helpottaa tunnistamaan osa-alueet mitä valinnassa kannattaa ottaa huomioon. Panon ym. (2018) mukaan heidän mallinsa onkin hyödyllinen sovelluskehyksen valinnasta päättäville henkilöille.



KUVIO 9 Panon, Graziotinin ja Abrahamssonin (2018) UTAUT:iin pohjautuva malli JavaScript sovelluskehysten hyväksymiselle (käännetty suomeksi)

Kuviossa 9 on kuvattu Panon ym. (2018) malli JavaScript sovelluskehyksen hyväksymiselle. Aikaisemmin esitellyt sovelluskehyksen hyväksymiseen vaikuttavat päätöksentekijät tarjoavat kontekstin ja laajuuden sovelluskehyksen hyväksynnän mallia käytettäessä. Panon ym. (2018) mallissa sovelluskehyksen hyväksyminen koostuu suorituskykyodotuksista, vaivattomuusodotuksista, sosiaalisesta vaikutuksesta, helpottavista olosuhteista sekä hinta-arviosta.

3.2.1 Suorituskykyodotukset

Suorituskykyodotukset (*performance expectancy*) koostuvat Panon ym. (2018) mallissa suorituskyvystä (*performance*) ja koosta (*size*). Suorituskyvyllä tarkoitetaan sovelluksen optimointia muistin ja kaistanleveydensuhteen. Koolla puolestaan tarkoitetaan yksinkertaisuudessaan sitä, että sovelluskehyksellä toteutetun sovelluksen tulisi sisältää mahdollisimman vähän koodirivejä. (Pano ym., 2018) Tämän tutkimuksen empiirisessä osuudessa kokoon on sisällytetty myös itse sovelluskehyksen paketin koko.

3.2.2 Vaivattomuusodotukset

Panon ym. (2018) vaivattomuusodotukset (*effort expectancy*) koostuvat neljästä eri alakohdasta:

- Automaatio (*Automatization*)
- Opittavuus (*Learnability*)
- Monimutkaisuus (*Complexity*)
- Ymmärrettävyys (*Understandability*)

Pano ym. (2018) eivät juurikaan listaamisesta huolimatta avaa automaation termiä tutkimuksessaan. Sen sijaan Pano ym. (2018) mainitsevat automaation *helpottavien olosuhteiden* alakohdassa *sopivuus (suitability)* olettaman automaatiosta: ”yksinkertaiset toiminnot kuten tapahtumien hallina, DOM manipulaatio ja reaaliaikaiset komponenttien päivitykset pitäisi olla automatisoituja”. Tämä on myös ikään kuin olettama käyttöliittymäsovelluskehyksissä, joten tämän tutkielman osalta siihen ei kiinnitetä juurikaan huomiota.

Opittavuudella (*learnability*) tarkoitetaan sitä, kuinka paljon vaivaa kehittäjä joutuu näkemään, että voi alkaa käyttämään sovelluskehystä. Tähän vaikuttaa olennaisesti myös kehittäjän tekniset taidot (Pano ym., 2018). Myös kehittäjätiimin aikaisempi kokemus vastaavista sovelluskehyksistä lienee varmasti myös merkittävä tekijä sovelluskehyksen opittavuuden kannalta.

Monimutkaisuudella (*complexity*) tarkoitetaan Panon ym. (2018) mukaan yksinkertaisesti sitä, että vähennetty havaittava monimutkaisuus nähdään tärkeänä sovelluskehyksen valintaan vaikuttavana tekijänä. Kehittäjät haluavat tyypillisesti välttää kompleksisia järjestelmiä.

Pano ym. (2018) määrittelee ymmärrettävyyden niiksi ominaisuuksiksi, jotka mahdollistavat käyttäjän tunnistamaan, kuinka sovelluskehystä

sovelletaan ja käytetään. He yhdistävät ymmärrettävyyden tiukasti sovelluskehyksen dokumentaatioon, sekä itse sovelluskehyksen avulla kirjoitettavan koodin helposti ymmärrettävyyteen. Tavallaanhan kyseessä on yksi sovelluskehysten perusajatuksista – piilottaa monimutkaisuudet ”konepellin alle” tehden koodista helpommin ymmärrettävää.

3.2.3 Sosiaalinen vaikutus

Sosiaalinen vaikutus on näkökulma, mitä ei olla juurikaan otettu huomioon sovelluskehyksiä tutkittaessa ennen Panon ym. (2018) tutkimusta. Esimerkiksi Gizas ym. (2012) eivät mainitse sosiaalista näkökulmaa tutkimuksessaan lainkaan, vaan he keskittyvät arvioimaan sovelluskehysten teknisiä ominaisuuksia. Sosiaalinen vaikutus nousee kuitenkin esille yhtenä tärkeimmistä tekijöistä sovelluskehystä valittaessa: Ferreiran, Borgersin ja Valenten (2021) tutkimuksessa *suosittuus* on kaikista eniten esille noussut syy sovelluskehyksen valinnalle. Tämä sama trendi toistuu myös tämän tutkimuksen empiirisessä osuudessa.

Pano ym. (2018) jakavat sosiaalisen vaikutuksen neljään eri alakohtaan:

- Kilpailija-analyysi (*Competitor Analysis*)
- Kollegojen neuvo (*Collegial Advice*)
- Yhteisön koko (*Community size*)
- Yhteisön reagointikyky (*Community responsiveness*)

Panon ym. (2018) mukaan sovelluskehyksen luotettavuuteen vaikuttaa olennaisesti se, jos samankaltaiset yritykset käyttävät kyseistä sovelluskehystä. Pano ym. (2018) käyttävät tästä termiä *competitor analysis,* eli suomennettuna kilpailija-analyysi.

Pano ym. (2018) nostavat yhtenä sovelluskehyksen valintaan johtavana tekijänä *kollegojen neuvon*. Joskus voi olla tyypillistä, että jotakin tiettyä sovelluskehystä suositellaan kollegojen tai muiden luotettavien verkostojen toimesta. Kollegojen neuvo rakentuu sovelluskehyksen havaitusta suosittuudesta. Ammatinharjoittajien mielestä päätökseen sovelluskehyksen valintaan vaikutti olennaisesti lukuisat maininnat sovelluskehyksestä, esimerkiksi esimerkkien ja ohjeiden muodossa. (Pano ym., 2018) *Kollegojen neuvo* on sinänsä siis vähän harhaanjohtava termi tässä yhteydessä, sillä tähän alakohtaan sisällytetään yleisesti koettu *suosittuus.* Myöhemmissä luvuissa ehdotetaan tiettyjen alakohtien yhdistämistä ja uudelleennimeämistä. Toki, termi *kollega* kuvastaa tässä yhteydessä *kehittäjiä yleensä,* sen sijaan, että havaintoja kerättäisiin vain omista verkostoista.

Yhteisön koko nousee yhtenä tekijänä JavaScript sovelluskehysten houkuttelevuutta tarkastellessa. (Pano ym., 2018) Pano ym. (2018) yhdistävät yhteisön koon vahvasti luotettaviin päivityksiin ja sovelluskehyksen elinkaaren luotettavuuteen. Sama kuvio toistuu myös tämän tutkimuksen empiirisessä osuudessa. Yhteisön kokoon sisällytetään sekä itse sovelluskehyksen kehitykseen osallistujat että sovelluskehyksen ympärillä olevien lisäosien kehittäjät.

Panon ym. (2018) mukaan yhteisön reagointikyky menee käsi kädessä yhteisön koon kanssa. Heidän mukaansa kehittäjille on tärkeää se, että mahdollisesti esille nouseviin ongelmiin löytyy nopeasti ratkaisu. Yhteisön

reagointikyvyn tarkastelu on olennaista myös siinä vaiheessa, kun tarkastellaan raportoituja ongelmia ja niiden ratkaisuja. Kehittäjät etsivät monesti ratkaisuja ongelmiin aikaisemmin nousseista keskusteluista foorumeilla kuten esimerkiksi StackOverFlow'ssa. Jos sovelluskehys on laajalti käytetty, voitaneen olettaa, että sovelluskehyksestä ja tavoista ratkaista eri ongelmia sen avulla löytyy paljon keskustelua kyseisillä keskustelualustoilla.

Lähes kaikki Panon ym. (2018) sosiaalisen vaikutuksen osa-alueista on suoraan yhteydessä *ekosysteemin* kokoon, jos sovelluskehyksellä on paljon käyttäjiä, eli se on *suosittu,* johon Pano ym. (2018) viittaa *kollegojen neuvolla,* voidaan olettaa, että myös monet kilpailijat käyttävät kyseistä sovelluskehystä (*kilpailija-analyysi*). Samaten voidaan olettaa, että suositulla sovelluskehyksellä on myös korkea *yhteisön reagointikyky* verrattuna vähemmän tunnettuun sovelluskehykseen.

3.2.4 Helpottavat olosuhteet

Helpottavilla olosuhteilla tarkoitetaan yksilön ymmärrystä sovelluskehyksen organisatorisista ja teknisistä infrastruktuurivalmiuksista, jotka tukevat sovelluskehystä. Se pitää sisällään ne sovelluskehyksen ominaisuudet, jotka kuvaavat vaatimusten sopivuutta ja integraatiopotentiaalia. (Pano ym., 2018)

Pano ym. (2018) jakaa helpottavat olosuhteet viiteen eri alakategoriaan:

- Sopivuus (*Suitability*)
- Päivitykset (*Updates)*
- Modulaarisuus (*Modularity*)
- Eristyneisyys (*Isolation*)
- Laajennettavuus (Extensibility)

Sopivuudella (*suitability*) tarkoitetaan Panon ym. (2018) mukaan niitä ominaisuuksia, jotka ilmaisevat, onko sovelluskehys *sopiva* täyttämään ne tehtävät, jota varten sitä ollaan ottamassa käyttöön. Käyttöliittymäsovelluskehyksiä ja niiden vertailua tarkastellessa sopivuus on aika epäolennainen tekijä, sillä kaikki käyttöliittymäsovelluskehykset pyrkivät tarjoamaan ratkaisun samaan tarkoitukseen, eli käyttöliittymän rakentamiseen. Sopivuutta on hyödyllisempi tarkastella käyttöliittymäsovelluskehysten laajennoksia tarkastellessa.

Päivityksillä Pano ym. (2018) tarkoittaa kehittäjien tarvetta käyttää sovelluskehystä, jota päivitetään jatkuvasti ja jonka ominaisuuksia laajennetaan, jotta sovelluskehys pysyisi kilpailukykyisenä. Panon ym. (2018) mukaan sovelluskehys, jota päivitetään usein uusilla ominaisuuksilla, nähdään positiivisena asiana. Pano ym. (2018) yhdistävät päivitykset myös selainyhteensopivuuksiin: he ehdottavat, että sovelluskehyksen avulla toteutettu sovellus tulee tukea eri selaimia.

Panon ym. (2018) oletus sovelluskehyksen modulaarisuuden tärkeydestä tarkoittaa sitä, että sovelluskehyksen tulee olla joustava. Sovelluskehyksellä toteutettuun sovellukseen täytyy pystyä tuomaan ulkoisia kirjastoja. Panon ym. (2018) materiaalissa käy ilmi, että tekijät kokevat tärkeänä sen, että uusia moduuleja voidaan ottaa projektiin mukaan sitä mukaa kun tarve sitä vaatii.

Eristyneisyydellä Pano ym. (2018) tarkoittavat sitä, että JavaScript sovelluskehyksen tulee suosia asiakaspään (*client-side*) kuormittamista sen sijaan, että

sovelluskehys kuormittaisi palvelinpäätä (*server-side*). Tämä on siinä mielessä erikoinen nosto, sillä osa JavaScript-sovelluskehyksistä, kuten NodeJs (Node), jota myös Panon ym. (2018) tutkimukseen osallistuneet henkilöt olivat käyttäneet, on pääosin palvelinpään sovelluskehys. Eristyneisyys on kuitenkin asia, joka on hyvä ottaa huomioon erityisesti käyttöliittymäsovelluskehysten yhteydessä. Toisaalta käyttöliittymäsovelluskehykset eivät teknisesti ottaen ole yhteydessä palvelinpäähän. Eristyneisyyden voitaisiin myös tulkita viittaavan siihen, että palvelinpään halutaan olevan mahdollisimman erillään palvelinpäästä, tai että käyttöliittymän sovelluskehyksen vaihtaminen ei saisi johtaa siihen, että palvelinpäässä jouduttaisiin tekemään merkittäviä muutoksia.

Viimeinen Panon ym. (2018) helpottavista olosuhteista on laajennettavuus ja sillä tarkoitetaan sitä, että sovelluskehykseen pitää pystyä tuomaan ulkoisia kirjastoja ilman mukauttamista. Tavallaan tämä sivuaa erittäin läheltä edellä mainittua *modulaarisuutta.* Itse näkisin laajennettavuuden ja modulaarisuuden asiana, jotka voisi yhdistää samaan alakategoriaan. Toisaalta laajennettavuudessa on kyse enemmänkin ulkoisten kirjastojen *yhteensopivuudesta* ilman niiden raskasta kustomointia.

3.2.5 Hinta

Pano ym. (2018) tarkoittaa hinnalla sovelluskehyksen käyttöönoton hintaa, eli esimerkiksi lisenssimaksua. Heidän mukaansa ilmaisia sovelluskehyksiä suositaan maksullisiin verrattuina. Tämän tutkimuksen osalta hinta on epäolennainen tekijä, sillä modernit käyttöliittymäsovelluskehykset ovat tyypillisesti ilmaisia, avoimen lähdekoodin sovelluskehyksiä mitkä eivät vaadi erillistä lisenssimaksua. Hintaan pureutuminen on siis tarpeetonta tämän tutkimuksen yhteydessä.

3.3 Käyttöliittymäsovelluskehyksen valintaan liittyvät erityispiirteet

Ferreiran ym. (2021) tuore tutkimus käyttää Panon ym. (2018) mallia pohjana tutkiessaan syitä käyttöliittymäsovelluskehysten omaksumiseen johtaneita tekijöitä ja toisaalta myös käyttöliittymäsovelluskehyksestä luopumiselle. Tutkimuksessa suurin osa vastanneista pitivät suosittuutta ja opittavuutta tärkeimpänä sovelluskehyksen valintaan johtaneena tekijänä. Varsinkin Vue nousi esiin vastaajien keskuudessa opittavuutensa puolesta; Vuen tyypillisin valintaan johtanut tekijä oli juuri sen helppo opittavuus. Jatkotutkimuksen kannalta on kiinnostavaa selvittää, millaisissa projekteissa helposti opittava sovelluskehys nähdään tärkeäksi valintakriteeriksi.

Vaikka aikaisemmissa tieteenalan sovelluskehysten arviointiin liittyvissä malleissa ja mittauksissa (Gizas ym. 2012; Pano ym. 2018) suorituskyky nähdään keskiössä, Ferreiran ym. (2021) tutkimuksen tulokset osoittavat, ettei suorituskyky ei ole nähty kovinkaan tärkeänä. Jatkotutkimuksen kannalta on mielenkiintoista nähdä, millaisissa projekteissa suositaan suorituskyvyltään kyvykästä sovelluskehystä. Erityisen kiinnostavan Ferreiran ym. (2021) tutkimuksesta tekee

näkökulma siitä, että käyttöön otettuja käyttöliittymäsovelluskehyksiä tarkasteltiin nimenomaan siitä näkökulmasta, että mikä osa-alue on suosittu minkäkin käyttöliittymäsovelluskehyksen käyttöönoton perusteena.

Ferreiran ym. (2021) tutkimus piti sisällään myös näkökulman käyttöliittymäsovelluskehyksestä vaihtamisen toiseen: yleisin tapaus oli Angularista siirtyminen Reactiin. Tyypillisin syy vaihtamiselle oli se, ettei kyseiselle teknologialle löytynyt kehittäjiä, tai ennemminkin nähtiin, että toiselle teknologialle löytyi merkittävästi enemmän tekijöitä.

4 YHTEENVETO KIRJALLISUUDESTA

Panon ym. (2018) malli JavaScript sovelluskehysten omaksumiseen johtavista tekijöistä tarjoaa hyvän pohjan käyttöliittymäsovelluskehyksen valinnalle ja se on tämän tutkimuksen teoreettinen viitekehys. Mallia ei ole kuitenkaan suunniteltu ainoastaan käyttöliittymäsovelluskehyksiä varten, vaan sitä voidaan soveltaa mitä tahansa sovelluskehyksen valintaa varten. Mallia voitaisiin siis laajentaa rajaamalla arvioinnin kohde pelkästään käyttöliittymäsovelluskehyksiin. Tämän tutkimuksen osalta meitä kiinnostaa, mitä osa-alueita Panon, Graziotinin ja Abrahamssonin (2018) mallista voitaisiin karsia, tai mitä osa-alueita siihen voitaisiin lisätä, että malli voitaisiin kohdentaa pelkästään käyttöliittymäsovelluskehyksille. Ferreiran ym. (2021) tutkimus antaa tälle jo hyvän pohjan.

Pano ym. sekä (2018) ja Ferreira ym. (2021) eivät kummatkaan ota tutkimuksissaan juurikaan huomioon projektin luonnetta. Tämä on myös yksi sellainen näkökulma, jota voisi tutkia tulevissa tutkimuksissa. Onko projektin tai esimerkiksi organisaation luonteella, tai asiakkaan toimesta sovellukselle asetetuilla vaatimuksilla vaikutusta siihen, mitä sovelluskehyksen valintaan johtavia tekijöitä koetaan tärkeiksi?

Yhtenä kirjallisuuskatsauksesta kummunneena havaintona todettakoon, että Panon ym. (2018) malli on varmasti hyvä pohja jatkotutkimuksen pohjaksi, sillä kyseinen malli toistuu jatkuvasti suhteellisen suppeassa aiheeseen liittyvästä kirjallisuudessa. Vaikka aihe on tuore ja siitä löytyy aika niukasti aineistoa, on siitä löytyvä vähäinen aineisto laadukasta. Myös ei-tieteellisistä lähteistä kuten kehittäjien keskustelupalstoilta sekä organisaatioiden tekemistä white papereista löytyy tutkimuksen kannalta mielenkiintoista dataa, mikäli sellainen koetaan jatkotutkimuksen kannalta tarpeelliseksi. Aikaisemmin kirjallisuudessa esitellyissä malleissa ja vertailuissa ei olla otettu huomioon sovelluskehysten teknisiä keskinäisiä poikkeavaisuuksia kuten JSX- tai HTML-sapluunoihin liittyviä eriäväisyyksiä tai TypeScript-natiivisuutta, eikä työmarkkinoihin liittyviä asioita. Nämä ovat sellaisia seikkoja mitä kannattaa nostaa esille tulevaisuuden tutkimuksessa.

Seuraavissa luvuissa tullaan käymään läpi empiirinen data, joka on kerätty haastattelemalla käyttöliittymäsovelluskehysten valintaprosessissa mukana olleita henkilöitä. Data on kerätty kvalitatiivisilla teemahaastatteluilla, ja käytetyt

haastattelukysymykset on rakennettu siten, että ne vastaisivat Panon ym. (2018) mallissa esiintyviä valintaan johtavia tekijöitä. Teemahaastattelun struktuuri pidettiin mahdollisimman vähäisenä, jotta myös sellaiset mahdolliset tekijät nousisivat esille, joita Panon ym. (2018) mallissa ei olla otettu huomioon. Haastatteludatan pohjalta on tarkoitus rakentaa Panon ym. (2018) malliin pohjautuva malli, josta olisi pudotettu pois käyttöliittymäsovelluskehyksille epäolennaiset osa-alueet.

5 TUTKIMUSMENETELMÄ

Tässä luvussa käsitellään sitä, kuinka tutkimuksen aineisto kerättiin ja miten sitä analysoitiin. Ensimmäinen alaluku sisältää kuvauksen tutkimuksessa käytetylle tutkimusmenetelmälle sekä perustelut sen valinnalle. Toisessa alaluvussa käydään läpi kvalitatiivista tutkimusta sekä sen luotettavuutta ja viimeisessä alaluvussa esitellään tutkimukseen osallistuneet henkilöt.

5.1 Tutkimusmenetelmä ja perusteet valinnalle

Tutkimuksen empiirinen osuus päätettiin toteuttaa kvalitatiivisilla, alhaisen struktuurin teemahaastatteluilla. Tutkimuksessa päätettiin hyödyntää Hsiehin ja Shannonin (2005) *teoriaohjattua analyysia (directed content analysis)*, joka tarkoittaa sitä, että tutkittavan ilmiön pohjalla on jokin valmis teoria, joka tässä tapauksessa on Panon ym. (2018) ehdottama malli JavaScript sovelluskehysten hyväksymiselle. Tutkimuksessa hyödynnetään joissain määrin myös Hsiehin ja Shannonin (2005) *tavanomaista sisältöanalyysia (conventional content analysis)* uusien löydösten esille nostamiseksi. Teoriaohjatun lähestymistavan valinta oli mielestäni varsin helppo päätös, sillä lähdekirjallisuudesta löytyi sovelluskehysten hyväksymisen malli, jota pystyi hyödyntämään hyvin tässä tutkimuksessa.

Hsiehin ja Shannonin (2005) mukaan teoriaohjatun analyysin tavoitteena on validoida tai laajentaa teoreettista viitekehystä tai teoriaa, joka kuvastaa hyvin tätä tutkimusta: tavoitteena on testata Panon ym. (2018) mallin osa-alueiden merkityksellisyyksiä, kun tarkastellaan asiaa nykyaikaisten käyttöliittymäsovelluskehysten valinnan näkökulmasta. Lisäksi tavanomainen sisältöanalyysi tulee olennaiseksi siinä kohtaa, kun keskustelussa nousee esille muita seikkoja, kuin mitä teoriapohjassa on esitetty. Haastatteluissa päädyttiin mahdollisimman vähästruktuuriseen lähestymistapaan juuri siitä syystä, että pystytään mahdollistamaan keskustelun nouseminen ennestään tuntemattomammistakin osa-alueista.

5.2 Kvalitatiivisen haastattelututkimuksen luotettavuus ja osallistujamäärä

Tutkimuksen luotettavuutta tarkastellessa on otettava huomioon, että tutkimukseen on valittu vain varsin rajallinen määrä haastateltavia, joka on varsin tyypillinen lähtökohta kvalitatiiviselle tutkimukselle. Kuudesta haastattelusta saatiin kuitenkin varsin kiitettävä määrä dataa, josta voidaan tehdä tiettyjä päätelmiä. Esimerkiksi tapauksissa, joissa kaikki haastateltavat ovat maininneet tietyn osa-alueen olevan merkittävä, voidaan se varsin hyvin yleistää olevan merkittävä myös suuremmassakin mittakaavassa. Tämän kvalitatiivisen tutkimuksen tarkoitus on havaita yleisimpiä käyttöliittymäsovelluskehysten valintaan vaikuttavia tekijöitä. Tulevaisuuden tutkimuksen kannalta voisi olla viisasta tehdä kvantitatiivinen tutkimus samoilla operationalisoinneilla, jotta otantaa saataisiin laaja-alaisemmin ja erilaisista työkulttuureista.

Kvalitatiivisen tutkimuksen aineisto on riittävä silloin, kun saavutetaan *saturaatio*, eli uusissa haastatteluissa ei nouse enää uusia asioita esille. (Saaranen-Kauppinen & Puusniekka, 2009; Eskola & Suoranta, 1998). Tässä tutkimuksessa kyllääntymispistettä ei asetettu vielä tutkimuksen alussa, vaan todettiin, että kun uusia asioita ei enää nouse esille, ei ole enää syytä etsiä enempää haastateltavia. Jo kuusi haastateltavaa osoittautui tuottamaan riittävästi mielenkiintoista kvalitatiivista dataa ja koettiin, että saturaatio saavutettiin, sillä uutta, mullistavaa tietoa ei enää tullut esille, vaan haastattelut alkoivat toistaa itseään. Toki haastatteluissa nousi esille lähestulkoon aina uusia pointteja, mutta ne olivat lähinnä teknisiä yksityiskohtia, joita ei tässä tutkimuksessa painoteta kovinkaan paljon, sillä ne ovat hyvin pitkälti kehittäjien teknisiä mieltymyksiä ja mielipiteitä.

5.3 Tutkimukseen osallistuneet haastateltavat

Tutkimukseen haastateltiin yhteensä kuutta eri henkilöä. Tutkimukseen osallistuneet haastateltavat koostuivat pääosin ohjelmistokehittäjistä, jotka ovat suoraan mukana kehittämässä sovellusta valitulla käyttöliittymäsovelluskehyksellä, lukuun ottamatta yhtä poikkeusta, jonka titteli oli Chief Digital Officer. Haastateltavien työkokemusvuodet vaihtelivat kuudesta vuodesta 15 työvuoteen asti.

Haastateltavat olivat kerätty henkilökohtaisista verkostoista työ-, vapaa-aika ja opiskelupiireistä. Haastateltavat ovat toisiinsa nähden suhteellisen erilaisista lähtökohdista huolimatta siitä, että he olivat kaikki ennestään tuttuja: työkokemukset ja tittelit vaihtelevat toisiinsa nähden ja haastattelutuloksista käy ilmi, että heillä on myös hyvissä määrin toisistaan eroavia näkökulmia käyttöliittymäsovellusten valintaan liittyvissä asioissa.

Kaikki haastattelevat työskentelevät konsulttitalossa, joten rajoitteena tutkimukselle on, että tuotetalojen näkökulma puuttuu. Tuotetalon näkökulma valinnalle voi erota merkittävästi konsulttitaloon verrattuna, joten tätä voisi olla mielenkiintoista tutkia tulevaisuuden tutkimuksissa. Voisi olla mielenkiintoista vertailla tuloksia ja nähdä, kuinka tulokset vaihtelevat. Taulukkoon 4 on kerätty

yhteenveto haastatteluun osallistuneista henkilöistä, heidän titteleistään, työkokemuksesta sekä heille annettu yksilöivä tunniste. Seuraavassa luvussa tullaan viittaamaan kuhunkin haastateltavaan tälle osoitetulla tunnisteella.

TAULUKKO 4 Teemahaastattelun osallistujat

Tunniste	Titteli	Työkokemus vuosina
H1	Full Stack Developer	6
H2	Full Stack Developer	8
H3	System Architect	10
H4	System Engineer	7
H5	Full Stack Developer	15
H6	Chief Digital Officer	7

6 TULOKSET

6.1 Suorituskykyodotukset

6.1.1 Suorituskyky

Moderneja käyttöliittymäsovelluksia tarkasteltaessa voidaan havaita, että suorituskykyerot ovat sovelluskehysten välillä niin marginaalisia, ettei sovelluskehyksen suorituskyvyllä ole merkitystä. Eräässä haastattelussa nousi esille, että suorituskyky koettiin tärkeäksi, koska se oli erikseen esitetty vaatimuksena projektille asiakkaan toimesta:

> H3: ”Tähän valittiin se Next.js sen suorituskykypuolen takia, että vois tarvittaessa generoida esim. staattisia sivuja, serveripuolella jo generoida asioita, että se clientille tuleva osuus olisi pienempi. [...] se on asiakkaan yksi päävaatimus, […] että se käyttöliittymä pitää olla tosi nopea ja […] pitäisi olla hakukoneoptimoitu, siitä syystä myös tämä valmiiksi renderöinti palvelimella.”

Myös kolmessa muussa haastattelussa todettiin, ettei suorituskykyä nähty merkittävänä, mutta tiedostettiin kuitenkin, että jos suorituskyky olisi *koettu tärkeäksi,* tai jos taustalla on *erityinen tarve suorituskyvylle,* olisi siihen voitu perehtyä:

> H2: ”Ei omissa mitä olen päätöksiä tehnyt niin tuo ei ole ollut merkittävä tekijä. Pitäisi olla jotenkin todella erityinen tarve, että tuo olisi semmoinen mitä tarvitsisi varsinaisesti miettiä.”

> H4: ”Suorituskyky on suhteellisen lähellä toisiaan noitten kaikkien vaihtoehtojen kesken, ja meillä ei ainakaan ollut vaatimuksissa mikään huippusuorituskyky”

> H5: ”Oli tieto, että se [suorituskyky] on riittävä. Jos olisi ollut huolia niin sen suhteen tai jotenkin merkittävä tekijä se suorituskyky niin silloin olisin saattanut perehtyä asiaan”

Muissa haastatteluissa suorituskyky ei noussut esille käyttöliittymäsovelluskehyksen valintaan vaikuttavana tekijänä.

> H1: ”Yleensähän nuo suorituskykyerot ovat aika marginaalisia, mut on niissäkin eroja. Mut ei se oo yleensä ollut siinä se mihin ois kaatunut mikään.”
>
> H6: ”Me ei koettu siinä kohtaa et niillä [sovelluskehyksillä] olisi merkittäviä eroja [suorituskykyjen välillä] ja se ei ollut myöskään niin painava tekijä, […], että ei koettu, että siellä olisi niin isoja eroja, että käyttäjät välttämättä huomaisi tai vaikuttaisi heidän kokemukseensa palvelusta.”

6.1.2 Koko

Sovelluskehyksen koko liittyy paljolti sovelluskehyksen *modulaarisuuteen.* Tarkasteltavista sovelluskehyksistä Vue ja React ovat paketin kokonsa puolesta reilusti pienempiä kuin Angular (Satrom, 2018). Pano ym. (2018) viittaavat *koolla,* sovelluskehyksellä kehitetyn ratkaisun koodirivien määrään. Ainoastaan yhdessä haastattelussa koko nousi esille, mutta sekin vaikutti vain ”joissain määrin”:

> H1: ”Kyllähän sitä jonkun verran mietitään yleensä, jos valitaan jotain johonkin projektiin. Niin kyllähän se vaikuttaa, jos joku mitä on ennen käytetty, tuottaa isoja tiedostoja ja tämmöistä niin saatetaan pohtia että kannattasiko sit käyttää jotain uudempaa tai toista jolla saa pienempää aikaan”

Samassa haastattelussa nousi myös esille, että projektin laajuus vaikuttaa siihen, otetaanko sovelluskehyksen koko tarkempaan tarkasteluun:

> H1: ” Kyllä pienen projektin tai verkkosivun jonkun interaktiivisuuden tehdä semmosella hyvin kevyellä ratkaisulla mikä ei tarvitse mitää compile-vaihetta tai mitään siihen väliin.”
>
> [..]
>
> Haastattelija: ”Eli keveys on tärkeää silloin kun on pieni projekti”
>
> H1: ”Niin, sun ei tarvi tuoda sitä 1MB frameworkia sitä varten, että sä saat buttonin tekstin vaihtumaan yhellä klikkauksella tai jotain vähän monimutkasempaa”

Muissa haastatteluissa kokoa ei nähty merkittävänä tekijänä.

6.2 Vaivattomuusodotukset

6.2.1 Automaatio

Yhdessäkään haastattelussa automaatio ei noussut esille olennaisena tekijänä käyttöliittymäsovelluskehyksen valinnalle. Myöskään Panon ym. (2018) tutkimuksessa, joka oli teoriapohjana tälle tutkimukselle, ei olla pureuduttu kovinkaan syvästi automaatioon. Pano ym. (2018) mainitsee sopivuuden (*suitability*) yhteydessä automaation: ”Yksinkertaiset tehtävät kuten tilanhallinta, DOM manipulaatio sekä reaaliaikaiset komponenttien päivitykset pitäisi olla automatisoituja”. Moderneille käyttöliittymäsovelluskehyksille tämä on tyypillistä, ja ikään kuin oletettuakin. Tämän valossa ehdotan automaation pudottamista pois tulevaisuuden malleissa, sillä se on nykykäsityksen mukaan oletuksena oleva toiminnallisuus.

6.2.2 Opittavuus

Opittavuus nousi esille varsin monessa haastattelussa. Tässä tutkimuksessa opittavuuteen mielletään myös aikaisempi kokemus sovelluskehyksestä:

> H1: ” Onhan se tärkeää kyllä, että esim. meillä käytetään paljon Vue’ta just sen takia. Ainakin se Vue 2 ja ihan alkuperäinenkin Vue oli tosi helposti ymmärrettävissä ja opittavissa uusillekin kehittäjille. Sen takia se onkin ehkä meillä käytetyin framework tällä hetkellä.”

Kahdessa haastattelussa opittavuus yhdistettiin suosittuuteen sekä dokumentaatioon, jonka Pano ym. (2018) lukevat osaksi ymmärrettävyyttä:

> H3: ” [Kysyttäessä opittavuuden tärkeydestä] Tosi tärkeänä, se että on paljon dokumentaatiota ja paljon esimerkkejä on kyllä tosi tärkeää. Siksi ehkä React-juttuja on tullut valituksi, kun se taitaa olla suosituin nyt. Siihen löytyy kaikista eniten materiaalia opetteluun.”

> H5: ”Jos ei muualta sanella, että on paljolti oma päätös niin silloin tulee helposti mentyä siihen suuntaan että onko ennestään tuttu, että mitä osaa tai tuntuu että mitä pystyisi helposti opettelemaan. Silloin yleensä tämmöiset suositut frameworkit on sellaisia mihin löytyy paljon materiaaliakin ja on helppo oppia, ei tarvitse yksin tarpoa suossa sitten, jos tulee ongelmia”

Eräässä haastattelussa opittavuus yhdistettiin vahvasti kehittäjien aikaisempaan kokemukseen. Kysyttäessä opittavuudesta, esille nousi aikaisempi kokemus:

> H6: ”Mehän siirryttiin käytännössä AngularJS:stä Angular 4:ään, mikä oli iso muutos, että periaatteessa samalla vaivalla olisi opetellut varmaan jonkin toisenkin. Mut ehkä se tuntu sit tutummalta kuitenkin, kun se oli samalta tiimiltä peräisin oleva tuote. Niin kyllä se niiku oppimiskäyrä on vaikuttava tekijä, mut suhteessa siihen olemassa olevaan osaamiseen.”

Angulariin päätynyt haastateltava tiedosti Angularin suuremman oppimiskäyrän suhteessa Reactiin, mutta aikaisempi kokemus sovelluskehyksestä nähtiin tärkeämmäksi:

> H6: ”Et Angular, onhan siinä enemmän learning curvea et sä saat sillä mitään aikaseksi, Reactilla sä voit tehä jotain tosi yksinkertaista tosi helposti, eikä sun tarvii lataa siitä puolta NPM kirjastoo sun koneelle”

Eräs haastateltava ei katsonut opittavuuden olevan kovinkaan olennainen tekijä, koska hänen mielestään kaikki JavaScript käyttöliittymäsovelluskehykset ovat suhteellisen helppoja opittavuudeltaan:

> [Kysyttäessä sovelluskehyksen helposta opittavuudesta]
>
> H2: ”Mielestäni en erityisen tärkeänä, mutta lähtökohta valinnoissa on ollut ettei oteta mitään super-eksoottista, missä täytyisi käyttää jotakin muuta kuin Javascriptiä tai TypeScriptiä”
>
> Haastattelija: ” Sanositko että kaikki nämä JavaScript fronttiframeworkit on kaikki aika helposti opittavia?”

> H2: ”Kyllä lähtökohtaisesti sanoisin että yleensä ovat aika helposti haltuun otettavia. Toki toiset voi olla helpompia kuin toiset, mut jo se että se on JavaScriptiä niin se helpottaa jo asiaa”

Opittavuutta ei nähty myöskään silloin tärkeäksi, kun kehittäjätiimi koostui vain yhdestä, kokeneesta henkilöstä:

> H4: ”Ei ollu korkeella. Lähinnä sen takia koska meillä ei oo yhtään, junioria, kenelle sitä pitäis opettaa. Jos joutuu ite jotain opettelemaan niin se ei ole semmoinen kynnys, että pitäisi muuttaa valintaa. Et se ei ollut oikeastaan tärkeä.”

Opittavuuteen yhdistettiin monessa tapauksessa aikaisempi kokemus sovelluskehyksestä tai vastaavista sovelluskehyksistä, joten tämä on seikka mikä kannattaa ottaa huomioon tulevaisuuden malleja rakentaessa: kokeneella ohjelmistokehittäjällä lähtökohta *opittavuuden* osalta on harvemmin nollassa.

6.2.3 Monimutkaisuus

Monimutkaisuus, tai pikemminkin yksinkertaisuus nähtiin selvänä kriteerinä sovelluskehyksen valinnalle lähes kaikissa haastatteluissa:

> H1: ”Ehkä se että se minkä takia me ollaan päädytty käyttää Vue’ta on se että se ei ole niin monimutkainen, tai sitten se piilottaa sen sinne konepellin alle sen koko DOM modelin muutokset ja sinun kehittäjä ei tarvi tietää välttämättä sieltä niin paljoa.”

> H2: ” [Jos mietitään] käytettyjä kirjastoja mitä kylkeen otetaan, niin kyllä niissä päätöksissä aika paljonkin tulee mietittyä sitä, että otetaanko siellä jotain sellaisia kovin erikoisia ratkaisuja, joissa on jotain epätavallisia konsepteja käytössä. [...] Yhtenä tulee mieleen yksi mitä on käytetty ja mikä oli React-maailmassa suosittu, on tilanhallintaan käytetty systeemi Redux-saagat, joka on siisti ilmaisuvoimainen systeemi sitten kun pääsee siihen sisään, mutta se muuttaa monia asioita niin paljon monimutkaisemmiksi että ei ole vaivan arvoista. Eli kyllä tällaisia juttuja tulee paljon mietittyä”

> H4: ” On. Se monimutkaisuus, tai oikeastaan se yksinkertaisuus on iso kriteeri. Se frontin edellinen versio oli Reactilla tehty + Reduxilla. Ja näillä ns. hienouksilla mitä oli vähän aikaa sitten ja Webpackilla paketoitu ja kaikilla tämmöisillä. Se monimutkaisuus mikä jo pelkästään siinä perus-stackin mukana tulee, on jo aika iso taakka. koska sitä kun pitäisi lähteä päivittämään, niin jokainen ylimääräinen monimutkainen osa mitä ei voi päivittää suoraan, tuo sitä lisätyötä. Niin nyt sen uuden frontin valinnassa se yksinkertaisuus on ollut se yks tärkeimmistä kriteereistä.”

> H5: ” Joo no sinänsä, jos oisin kokenut että React on hirmu monimutkainen niin sitten oisin ottanut jotain muuta.”

Myös monimutkaisuuden yhteydessä nousi esille jo olemassa oleva tieto: nollatilanteesta lähtevä kehittäjä voisi kokea Angularin monimutkaiseksi. Samassa yhteydessä monimutkaisuus liitettiin opittavuuteen:

> H6: ” […] monimutkaisuus on osa sitä oppimiskäyrää varmasti, paitsi jos se logiikka on jo sulla hallussa niin sit se ei enää oo niin monimutkainen. [...] Angularissa tulee paljon enemmän vakiona mukana, et kyllä siinä on paljon heti paljon enemmän kamaa ja siinä on paljon enemmän opeteltavaa, joten se on tavallaan, se minimisetti on monimutkaisempi.”

Angularista nousseena havaintona nostettakoon, että kun kehittäjällä on jo riittävä tieto sovelluskehyksestä, on sen koettu monimutkaisuus ja opittavuus alhaisempi.

6.2.4 Ymmärrettävyys

Ymmärrettävyys yhdistetään Panon ym. (2018) tutkimuksessa dokumentaatioon. Hyvän dokumentaation tärkeys mainitaan useassa eri haastattelussa:

> H1: [kysyttäessä tärkeintä päätökseen johtanutta seikkaa] ”Itelle varmaan kyl se et dokumentaatio oli tosi hyvä, oli silloin ja on vieläkin.”
>
> H3: ”Se että on paljon dokumentaatiota ja paljon esimerkkejä on kyllä tosi tärkeää.”
>
> H6: ”Hyvin tärkeänä. ja se toki liittyy myös siihen, että on laajalti käytössä oleva framework, jolloin myös löytyy enemmän tilannekohtaisia kysymyksiä netistä esim. StackOverFlowsta.”

Lopuissa haastatteluissa dokumentaatiota ei nähty tärkeäksi, koska koettiin, että dokumentaatio oli kaikkien tarkastelussa mukana olleiden, suosittujen sovelluskehysten osalta riittävä.

> H2: ”Ei oo ehkä erikseen koskaan tullu sitä pohdittua. Tuohan liittyy kans siihen, että ei valitse mitään ihan hirmu eksoottisia kirjastoja. Jos on yhtään käyttäjäkuntaa niin todennäköisesti dokumentaatio on riittävän hyvää.”
>
> H4: ”En nähnyt dokumentaatiota tärkeäksi tai tehnyt vertailua sen osalta, koska mun mielestä kaikissa näissä on riittävän hyvät dokumentaatiot.”
>
> H5: ”En erityisemmin [ottanut selvää dokumentaatiosta]. Se on osa sitä et kannattaa ottaa suosittu. Löytyy matskua, oli sitten virallista tai käyttäjien dokumentaatiota.”

6.3 Sosiaalinen vaikutus

Kaikki sosiaalisen vaikutuksen alla olevat alikohdat ovat tiukasti sidoksissa siihen, kuinka *suosittu* sovelluskehys on. On siis äärimmäisen hankalaa linjata, mihin Panon ym. (2018) sosiaalisen vaikutuksen alakohtaan mikin haastattelijoiden sanominen yhdistetään. Tässä tutkielmassa tullaan ehdottamaan sosiaalisen vaikutuksen alakohtien yhdistämistä selkeämmiksi alakohdiksi.

6.3.1 Kilpailija-analyysi

Missään haastattelussa ei suoraan käynyt ilmi, että kilpailijoiden käyttämiä teknologioita olisi otettu huomioon käyttöliittymäsovelluskehystä arvioitaessa. Aihetta kuitenkin sivuttiin siltä kantilta, että huomioon otettiin ajatus siitä, että valittavaan teknologiaan löytyisi tekijöitä jatkossa:

> H3: ” Tavallaan se Reacti ehkä valintana oli enemmän myös ehkä niiku sen opittavuudenkin joo, mutta myös se, että niitä tekijöitä on enempi saatavilla.”
>
> ”…jos ottaa jonku tosi tuntemattoman teknologian ni sillon varmaan pitäs perustella et miks, koska yleensä nää menee ylläpitoon ja muualle ja sit ois hyvä et löytyy sellaisia ihmisiä, jotka osaa sitä käyttää.”
>
> H5: ”Aikasemmin kun on tehty jotain valintoja, niin mentiin just sillä et pitää päästä jotain tekemään eikä oo hirveen tärkee sinänsä mikä se tekniikka [...] sama pätee myös ehkä

> ulospäin että mikä on tulevalle tiimille tuttua eli siihenkin pätee, että mikä on suosittua, niin siihen löytyy tekijöitä."

Tästä voitaisiin rivien välistä päätellä, että kilpailijoiden osaamista analysoitaisiin, tai siitä olisi vähintäänkin sovelluskehyksen *suosittuuden* perusteella olemassa valistunut arvio. Se, että tekijöitä on saatavilla, voidaan tulkita niin, että myös kilpailijat käyttävät kyseistä teknologiaa.

Eräässä haastattelussa teknologia valittu teknologia nähtiin työntekijöiden houkuttelevuustekijänä: teknologialla halutaan erottua kilpailevista ohjelmistotaloista, tai ei ainakaan haluta valita teknologiaa siten, että se vaikuttaisi kilpailijoihin nähden tylsältä:

> H6: "Koodari on niukka resurssi ja niitä on vaikea löytää [...] ni sit jos se sun firma tekee jotain ei-suosittua, jotenki koetaan vaik vanhanaikaseks tai et se ei oo kauheen pinnalla se teknologia, ni sun voi olla vaikee saada [tekijöitä]
>
> [...]
>
> Se voi olla tämmöinen työntekijähoukuttelevuuskysymys, [...] varmaan ihan vaikuttava tekijä joillekin et miksi päätetään ottaa joku just nyt joku tosi pinnalla oleva [...] yritykset etsii kaikkia keinoja houkutellakseen tekijöitä, kun niistä on pulaa, niin oisko teknologiavalinnat yks semmoinen millä saatetaan, lähtee houkuttelemaan ihmisiä"

6.3.2 Kollegojen neuvo

Panon ym. (2018) mukaan kollegojen neuvo (*collegial advice*) kattaa valintaa tekevän henkilön verkostoilta saatavat neuvot ja ehdotukset sovelluskehyksestä. Kollegojen neuvot. Tässä tutkimuksessa kollegojen neuvoon on päätetty liittää tiimin sekä valintaa tekevän henkilön aikaisempi kokemus sovelluskehyksestä, oli se sitten positiivinen tai negatiivinen.

Haastatteluissa kollegojen neuvo nousi esille lähinnä kollegojen ja tiimin aikaisemman kokemuksen muodossa. Lisäksi nousi pointteja siitä, mistä puhutaan ja mikä vaikuttaa kiinnostavalta.

> H1: "Siinä vaiheessa oli yks kehittäjä tehnyt yhden projektin Vuella ja muut oli kattonu vierestä tyyliin ja oltiin silleen et tuohan näyttää hyvältä et kokeillaan sitä [...] yheltä löyty sitä kokemusta siitä niin se vaikutti siihen [valintaan] suuresti"
>
> H2: "Myös kiinnostuksella, että onko motivaatiota vaihtaa, ottaa selvää jostakin mitä he eivät ole aikasemmin käyttäneet [...] Jos on olemassa jokin suosittu, kiinnostava framework mistä puhutaan paljon mutta ei ole itse päässyt testaamaan, niin se esimerkiksi ainakin [voisi olla peruste tutustua uuteen sovelluskehykseen]"
>
> H3: "No varmaan ehkä isoin vaikutus se, että valitaan semmonen mikä on kaikille tuttu framework. [...] Ehkä jos ois ollu varmaan Angularin tekijöitä suurin osa, semmosia joilla ois ollu siitä kokemusta niin sit ois varmaan semmonen valittu. Mut nyt oli kaikilla kokemusta Reactista ja se oli suosituin ja tiedetään että sillä saa ne [vaatimukset] aikaseksi niin valittiin se. Tosi tärkeä rooli on sillä et millä on aiemmin tehty"
>
> H4: "Se kokemus, [...] riippuu tietysti mitä mieltä on niistä frameworkeista mitä on käyttäny tähän mennessä, mutta pääosin se kokemus kertoo siitä millasii asioita haluu tehä tai mitkä asiat toimii jollakin frameworkilla millaisia asioita haluu sit sen uuden frameworkin tekevän paremmin; et se edellinen kokemus ei suoraan vaikuta uuden valintaan, koska jos sinä oot valitsemassa jotain uutta, sen pitää tehdä jotain eri tavalla koska muutenhan siinä

> ei oo mitään arvoa vaihtaa, niin enemmän se on sitä et se kokemus kertoo, mistä tykkäsit tai et tykännyt edellisessa frameworkissa."
>
> H6: "Oli jo olemassa olevaa osaamista Angularista. [...] ehkä se tuntu sit tutummalta kuitenkin, kun se oli samalta tiimiltä peräisin oleva tuote."

6.3.3 Yhteisön koko

Yhteisön koko nähtiin yleisesti ottaen erittäin tärkeänä tekijänä.
Osassa haastatteluista nousi esille, että yhteisön koko on tärkeä tekijä. Se tiedostetaan, mutta siitä ei varsinaisesti oteta sen enempää selvää.

> H1: " Ihan silleen vaan että katsottiin GitHubista projektin repoa ja luettiin netistä ja tällein [...] 5 vuotta sitten se oli vähän niiku harrasteprojekti melkein, mutta silloinkin sillä oli paljon käyttäjiä, niin se oli semmoinen varteenotettava vaihtoehto ja päädyttiin sit siihen"
>
> H2: "On se [suosittuus] semmoinen asia mistä oon kyllä aina ollut tietoinen, kun noita on tullut mietittyä, että varsinaisesti ei oo tullut tehtyä tutkimusta, kun on tullut käytettyä pääasiassa noita varsin suosittuja työkaluja. Onhan se merkittävä juttu, mutta sitä ei oo tarvinnut erikseen tutkia, tietää kuitenkin. Meillä ne päätökset on ollut noitten isojen ja tunnettujen, suosittujen vaihtoehtojen välillä anyway. On ollut aika hyvin tiedossa, että kyllä sieltä löytyy ekosysteemiä, että se tukee, valitsi kummin vaan."
>
> H5: "Joo, eli oli tiedossa, että oli käytännössä industry standard. [...] Tässä oli pohjalla kanssa se, että oli katottu sopivia projekteja ja lähes kaikissa haluttiin React-kokemusta. Eli kyllä se oli yksi suurimmista tekijöistä."

Eräässä haastattelussa suosittuus yhdistettiin myös päivityksiin: enemmän kehittäjiä ympärillä tarkoittaa sitä, että sovelluskehyksen ympärille rakennetaan nopeammin lisäosia:

> Haastattelija: "Otitteko selvää sovelluskehyksen suosittuudesta?"
>
> H3: "Kyllä. Aiemmin Vuen kanssa on ollu tiettyjä haasteita just sen takia et siel ei oo niitä uusimpia juttuja tehty niihin Vuen päälle mitä Reactissa olisi ollut tarjolla. Jos on tullu jotain uusia, vaikka JavaScriptin ominaisuuksia tai muita niin sitä ei oo niissä Vuen jutuissa monestikaan vielä hyödynnetty kauhean hyvin, kun taas Reactissa on joku tosi nopeasti taas tehnyt projektin päälle […] ja ehkä se yhteisön koko siinä on tietysti, et se on nii paljo isompi se Reactin yhteisö ja siellä on nii paljon enemmän niitä open-source-projekteja Reactilla."
>
> […]
>
> H3: "Mä luulen, että Reactissa on niin paljon enemmän vaan porukkaa tekemässä asioita, sekä itse Reactia että myös sen päälle tehtyjä ratkaisuja."

Eräässä haastatteluna suosittuus oli oletuksena lähtökohtana, eikä vähemmän suosittuja otettu edes mukaan vertailuun, samassa nousee esille sovelluskehyksen kypsyys:

> H6: "Se lähtökohta oli se et ne vaihtoehdot pitää olla semmoset et pitää olla tosi mainstream, et ei lähetä tekee millään pienellä. [...] käytännössä tää oli Angularin ja Reactin vertailua, Vue ei ollu sillon niin iso juttu. Puhutaan kuitenkin 2017, Vuehan tuli vasta 2014, ei se siinä kohtaa ollu viel niin iso peluri näissä markkinoissa."

Yhdessä haastattelussa nousi esille, että suosittuus ei enää vaikuta olennaisesti valintaan, mikäli tarkasteltavat sovelluskehykset ovat riittävän suosittuja.

Suosituimpien sovelluskehysten keskinäisessä vertailussa yhteisön koko ei ole enää niin merkittävä tekijä:

> H4: ”Kyllä sillä frameworkin ympärillä pitää olla riittävästi kehittäjiä [...] Toisaalta tietyn rajan jälkeen mitä se näyttää, että se pysyy omalla painollaan hengissä, niin sen lisäksi en ota enää sitä suositumpaa.”

6.3.4 Yhteisön reagointikyky

Yhteisön reagointikyky nähtiin tärkeänä tekijänä, ja monessa tapauksessa se yhdistettiin suoraan sovelluskehyksen ympärillä olevan ekosysteemin kokoon:

> H3: ”Jonkun verran paremmin löytyy keskustelua näihin React-maailman juttuihin, ku vaikka Vuen tai Angularin. Tällä on niin paljon enemmän niitä kehittäjiä niin tietty on enemmän myös kysymyksiä laitettu, vaikka StackOverflow’hun tai muualle. Ja samaten ehkä löytyy myös paljon kaikkia GitHub-issueita ja muita Reactin alta, joista sitten näkee selkeästi jotain ratkaisuehdotuksia.”

> H5: ” [kysyttäessä ottiko selvää, löytyykö StackOverFlow’sta ratkaisuja ongelmiin] Joo, että löytyy niitä. Yleensä jos on tosi harvinainen niin pitää itse selvitellä paljon asioita, mutta jos se on yleinen, niin löytyy jo valmiiksi vastauksia.”

> H6: ” [kysyttäessä dokumentaatiosta] [...] liittyy myös siihen, että on laajalti käytössä oleva framework, jolloin myös löytyy enemmän tilannekohtaisia kysymyksiä netistä esim. StackOverFlow’sta”

Yhteisön reagointikyvystä ei välttämättä otettu erikseen selvää, vaan taustalla oli käsitys suosittuudesta, joka antoi olettaa, että koska sovelluskehys on suosittu, on oletettua, että yhteisön reagointikyky on hyvä:

> H2: ”En oo erikseen pohtinut [reagointikykyä], vaan se on mennyt sitä kautta, että jos tietää jo, että framework on suosittu, niin kyllä siitä silloin löytyy vastauksia. En oo käynyt kattomassa miten paljon Vue, Svelte ym. kysymyksiä StackOverFlow’ssa on. [...] Kyllä se on aika varma oletus, että jos tietää, että sitä käytetään laajalti niin kyllä siihen silloin löytyy tukea ja tietoa hyvin.”

Yhteisön reagointikyky on tiukasti sidoksissa yhteisön kokoon. Myöhemmissä luvuissa ehdotetaan yhteisön koon ja yhteisön reagointikyvyn yhdistämistä yhdeksi alaluvuksi.

6.4 Helpottavat tekijät

6.4.1 Sopivuus

Panon ym. (2018) määritelmä sopivuudesta on käyttöliittymäsovelluskehysten arvioinnissa kohtuullisen epäolennainen osa-alue, sillä kaikki käyttöliittymäsovelluskehykset ratkaisevat saman ongelman: käyttöliittymän rakentamisen. Vain yhdessä haastattelussa sopivuus nousi esille, mutta siinäkin kyse oli käyttöliittymäsovelluskehyksen laajennoksesta. Sopivuutta pohdittiin ainoastaan sen takia, koska asiakkaalta oli tullut vaatimus hakukonenäkyvyydestä. Kyseinen kommentti viittasi kuitenkin enemmän

käyttöliittymäsovelluskehyksen lisäosaan kuin itse käyttöliittymäsovelluskehyksen valintaan:

> H3: ”[...] sit siellä on myöhemmin tulossa semmonen julkinen osa joka pitäisi olla hakukoneoptimoitu, siitä syystä myös tämä valmiiksi renderöinti palvelimella. [...] enemmän tutkittiin suoria materiaaleja, että miten tietynlaisia juttuja on ratkaistu, esim. just ne sivugeneroinnit ja muut ni sitä kautta tuli tuo Next.js esimerkiks vastaan”

6.4.2 Päivitykset

Päivitykset nousivat kohtuullisen merkittäväksi tekijäksi haastatteluissa

> H2: ”On hyvä tietää miten sitä ylläpidetään vastuullisesti, että on selkeä päivityspolku seuraavaan versioon, että rajapintamuutokset sun muut on aina hallittuja ja niistä tulee hyvät deprekaatiovaroitukset jo edellisissä versioissa, että tietää jos nykyisessä sovelluksessa ei tule varoituksia niin tietää vielä että se toimii suorilta ilman että tarvii tehdä koodimuutoksia.”

Taustalla oleva organisaatio, sekä ekosysteemi tuo turvallisuudentunnetta sekä näkyvyyttä jatkuvuuteen ja antaa luottoa siihen, että päivitykset ovat ajantasaisia ja sovelluskehyksen elinkaaresta pidetään huolta:

> H2: ”Kun on niinkin iso organisaatio kuin Facebook Reactin takana niin se auttaa paljon siihen, että ne päivitykset eivät tapahdu miten sattuu, vaan siihen on selkeä prosessi ja siihen voi luottaa, ettei se hajoa alta, koska ne itsekin käyttää sitä laajalti.”

> H4: ”Se kuin suosittu se on vaikuttaa varmasti siihen et sitä varmasti päivitetään, se pysyy hengissä.”

> H5: ”Onhan se ehdottoman tärkeää, ettei se ole semmonen missä tarvii pelätä, että se kohta kuolisi. Siinä taas päästään tohon että mikä on yleisesti paljolti käytetty niin yleensä voi luottaa siihen että se sitten pysyykin elossa”

> H6: ”Oletus oli et se on semmonen iso, tunnettu ja käytetty framework ja se projekti ei jää kesken, vaan se elää ja menee eteenpäin. Ja jos taustayhteisöt on Facebook ja Google niin oletus oli et nää projektit varmasti elää ja voi hyvin vielä jonkun aikaa, et uskaltaa sit sijoittaa siihen henkistä pääomaa, ja että on päivitettävyyttä”

Usein päivitykset yhdistetään sovelluskehyksen luotettavuuteen: sovelluskehys nähdään elinvoimaisena ja jatkuvana projektina kun sitä päivitetään.

6.4.3 Modulaarisuus ja laajennettavuus

Angulariin sisältyy enemmän toiminnallisuuksia muihin sovelluskehyksiin nähden, mikä samalla tekee siitä muita sovelluskehyksiä *monimutkaisemman* ja vaikeammin *opittavan*. Haastattelussa, jossa haastateltavan organisaatio oli päätynyt Angulariin, kävi ilmi, että modulaarisuutta ei nähdä tärkeänä tekijänä silloin, kun kehitettävä sovellus on tarpeeksi monimutkainen ja vaatimuksiltaan monipuolinen.

> H6: ”Reactilla sä voit tehä jotain tosi yksinkertaista tosi helposti, eikä sun tarvii lataa siitä puolta NPM kirjastoo sun koneelle, mut sit aina kun sä haluut tehä jotain lisää niin sä lataat lisää niitä kirjastoja.
>
> […]

> Mut sit tavallaan, kun ne sovellukset mitä me tehään niin ne vaatii kuitenkin aina [tiettyjä ominaisuuksia] mukaan
>
> […]
>
> sä joudut kuitenkin rakentaa sen et oli se sit lisäpalikoiden avulla tai pelkästään sillä frameworkilla sen tietyn setin niit kyvykkyyksiä mitä me tarvitaan et voidaan rakentaa niit sovelluksia mitä me rakennetaan."

Yleensä modulaarisuus havaittiin kuitenkin suhteellisen tärkeänä tekijänä, ja sen puute kuormittavana tekijänä:

> H1: "Jossain määrin joo, että jos sun tarvii tehä jotain niin sun ei tarvi tuoda sitä koko eläintarhaa mukana, sä voit sitten yksitellen ottaa niitä käyttöön sitä mukaa, ku sä tarvit niitä."
>
> H4: " Esimerkiksi siitä en tykännyt että Reactissa aika usein se starter-template millä alotellaan, on aika raskas, sit kun se puretaan, eli se kun ns. avaa itsensä niin siellä on 50 juttua, mistä ei oo hirveen hyvää tietoa [...] sieltä tulee jonkin verran niitä millä ei tee oikein mitään mut ne on pakko ottaa koska se on osa sitä templatea"

6.4.4 Eristyneisyys

Parissa haastattelussa nousi esille eristyneisyys, mutta sitä pidettiin ikään kuin itsestäänselvyytenä. Eristyneisyys nousi lähinnä silloin, kun haastateltavilta kysyttiin palvelinpään sovelluskehyksen vaikutuksesta käyttöliittymäsovelluskehyksen valintaan.

> H4: "Se tavoite on aina et ne on sen verran irrallisia, et sinun pitää pystyä vaihtamaan se back-endistä huolimatta se front-end"
>
> "Yleensä se [backend-sovelluskehys] ei vaikuta, että se on aika sama et millä sen frontin tekee. Joissain tapauksissa saattaa olla et joku juttu back-endissä vaatii jonkun tietyn frontin plugarin tai Vuen lisäosan et ne on suunniteltu toimimaan yhteen, mut harvemmin semmosta tilannetta on tullu."

Nykyisen yleisesti käytettyjen sovellusarkkitehtuurimallien mukaan onkin tyypillistä, että asiakaspää toimii erillisenä rajapintana sovelluksessa. Tästä kerrotaan lisää luvussa 6.9.1, jossa ehdotetaan, ettei eristyneisyyttä ole syytä ottaa huomioon käyttöliittymäsovelluskehystä valittaessa.

6.5 Yhteenveto käyttöliittymäsovelluskehysten valintaan vaikuttavista tekijöistä

Taulukkoon 5 on kerätty yhteenvetona kaikki haastateltavat ja kaikki sovelluskehyksen vaikuttavat tekijät. Taulukon tavoitteena on havainnollistaa, mitkä käyttöliittymäsovelluskehysten valintaan tekijät on koettu merkittävinä, ja mitkä niistä on koettu kaikista tärkeimpinä. Taulukkoon on merkitty tähdellä kaikki ne tekijät, jotka haastateltavat ovat maininneet olleen tärkeimpiä valintaan johtaneita tekijöitä.

Taulukkoa 5 tarkastellessa merkittävimpinä tekijöinä nousevat opittavuus, monimutkaisuus, kollegojen neuvo, yhteisön koko sekä päivitykset.

Automaatiota ja eristyneisyyttä ei noussut esille yhdessäkään haastattelussa, ja sopivuuttakin sivuttiin vain hyvin pintapuolisesti. Lisäksi kokoa ei nähty juuri ollenkaan merkittävänä tekijänä ja suorituskyky nähtiin merkittävänä vain silloin, kuin taustalla oli erityinen tarve, käytännössä asiakkaan asettama vaatimus.

TAULUKKO 5 Yhteenveto sovelluskehysten valintaan vaikuttavista tekijöistä

Haastateltava	Suoritusky-kyodotukset		Vaivattomuus-odotukset				Sosiaalinen vaikutus				Helpottavat olosuhteet				
	Suorituskyky	Koko	Automaatio	Opittavuus	Monimutkaisuus	Ymmärrettävyys	Kilpailija-analyysi	Kollegojen neuvo	Yhteisön koko	Yhteisön reagointikyky	Sopivuus	Päivitykset	Modulaarisuus	Eristyneisyys	Laajennettavuus
H1	E	J	-	*M	M	*M	E	M	M	M	J	-	M	-	M
H2	V	E	-	J_1	M	E	E	M	M*	J_4	-	M*	-	-	-
H3	V	E	-	M*	E	M	M	M*	M*	M	M	-	-	-	-
H4	V	E	-	E_2	M*	E	E	J	J_3	E	-	M	M*	-	M
H5	V	E	-	M	M	E	M	M*	M*	M	-	M*	-	-	-
H6	E	E	-	M*	M	M	M	M*	M	E	-	M	E_5	-	E

*) Tärkeimpänä pidetty valintaan johtanut seikka
Koettiin merkittäväksi tekijäksi
Koettiin jossain määrin merkittäväksi tekijäksi
Ei koettu merkittäväksi tekijäksi
Tiedostettiin mahdollisesti merkittäväksi jos esitetty erillisenä vaatimuksena
Asia ei noussut esille haastattelussa

Selitteet taulukon numeroiduille merkinnöille:

1. Kaikki JavaScript -sovelluskehykset koettiin verrattain helposti opittaviksi, eikä siitä syystä ollut kovinkaan merkittävä tekijä.
2. Opittavuutta ei pidetty tärkeänä, koska kehittäjätiimissä ei ollut juniorkehittäjiä, ja se koostui käytännössä vain haastateltavasta. Haastateltava totesi itsekin, että veikkaisi isommilla tiimeillä opittavuuden olevan suuremmassa merkityksessä, koska sovelluskehyksen toimintaan täytyy pystyä perehdyttämään useampi henkilö
3. Koettiin tärkeäksi, mutta tietyn rajan jälkeen ei enää merkittävä seikka
4. Yhteisön reagointikykyä ei tutkittu erikseen, koska sen oletettiin olevan hyvä laajan ekosysteemin takia
5. Ei nähty merkittäväksi, koska nähtiin, että projektit vaativat aina kaikki valitun sovelluskehyksen (Angularin) moduulit

6.6 Muita huomioita

Muita haastatteluissa nousseita huomioita ovat taustaorganisaation tuoma turva, tekniset mieltymykset sekä työmarkkinoihin liittyvät asiat, joita käydään läpi tarkemmin seuraavissa alaluvuissa.

6.6.1 Taustaorganisaation tuoma turva

Useassa haastattelussa nousi esille sovelluskehyksen taustalla olevan organisaation tarjoama turva:

> H2: ”Sitten kun on niinkin iso organisaatio kuin Facebook Reactin takana niin se auttaa paljon siihen, että ne päivitykset eivät tapahdu miten sattuu vaan siihen on selkeä prosessi ja siihen voi luottaa ettei se hajoa alta, koska ne itsekin käyttää sitä laajalti.”

> H3: ”Iso organisaatio, semmoinen kuin Vercel on sen takana...”

> H6: ”Ja jos taustayhteisöt on Facebook ja Google, niin oletus oli et nää projektit varmasti elää ja voi hyvin vielä jonkun aikaa.”

Monet näkevät siis taustalla olevan organisaation takuuna sille, että sovelluskehyksen päivitettävyys säilyy ja projekti pysyy elinvoimaisena.

6.6.2 Tekniset mieltymykset

Yksi haastattelija mainitsi TypeScript-tuen olevan ehdoton. Sovelluskehystä ilman TypeScript -tukea ei lähdetä edes harkitsemaan:

> H4: ”Ehdoton on myös Frameworkin TypeScript-tuki. Jos framework ei ole TypeScript-yhteensopiva niin en käyttäisi aikaa sen harkitsemiseen.”

Myös eräässä toisessa haastattelussa TypeScriptin yhteensopimattomuutta Vuen HTML-sapluunan kanssa harmiteltiin:

> H2: ”Jos miettii vaikka mikä Vuen kanssa on nyppinyt, niin ehkä vähän spesifejä pointteja. Se, että miten Vuen kanssa ei voi kunnolla hyödyntää TypeScriptiä jos käyttää HTML-templateja näkymien tekemiseen.”

Kaikki tässä tutkielmassa käytetyt sovelluskehykset tukevat TypeScriptin käyttöä enemmän tai vähemmän, pois lukien Vuen HTML-sapluunaosuus. Uusien, kilpailevien sovelluskehysten tullessa markkinoille kannattaa ammatinharjoittajien ottaa huomioon, onko JavaScript-sovelluskehyksellä TypeScript-valmiudet.

6.6.3 Työmarkkinaan liittyvät asiat

Eräs haastattelija valitsi sovelluskehyksen, koska halusi kokemusta tästä nimenomaisesta sovelluskehyksestä, haastateltava näki sovelluskehyksen osaamiskokemuksen työmarkkinaetuna:

> H5: ”Tässä oli pohjalla kanssa se että oli katottu sopivia projekteja ja lähes kaikissa haluttiin React-kokemusta. Eli kyllä se oli yksi suurimmista tekijöistä... [...] se että on vähän kokemusta aiheesta niin haluu oppia lisää.”

Myös toisessa haastattelussa sivuttiin sitä, kuinka valintaan voisi vaikuttaa se, mitä kehittäjätiimin jäsenet haluavat oppia

> H3: ”Tekijöille on aina tekijöiden motivaation kannalta [kannattaa] valita semmosia uusia teknologioita, joille on paljon kysyntää niitten osaamiselle, et kyllä se vaikuttaa myös siihen teknologiavalintaan myös se et mitä tekijät haluavat oppia ja millekä osaajille työmarkkinoilla on kysyntää.”

Eräässä haastattelussa nousi esiin työmarkkinoihin liittyvä houkuttelevuus, jota sivuttiin jo kilpailija-analyysia koskevassa alaluvussa. Tämä oli kuitenkin sen verran kiinnostava nosto, jota on syytä korostaa:

> H6: ”Koodari on niukka resurssi ja niit on vaikee löytää [...] ni sit jos se sun firma tekee jotain ei-suosittua, jotenki koetaan vaik vanhanaikaseks tai et se ei oo kauheen pinnalla se teknologia, ni sun voi olla vaikee saada [tekijöitä]
>
> [...]
>
> Se voi olla tämmöinen työntekijähoukuttelevuuskysymys, [...] varmaan ihan vaikuttava tekijä joillekin et miksi päätetään ottaa joku just nyt joku tosi pinnalla oleva [...] yritykset etsii kaikkia keinoja houkutellakseen tekijöitä, kun niistä on pulaa, niin oisko teknologiavalinnat yks semmoinen millä saatetaan lähtee houkuttelemaan ihmisiä”

Työmarkkinaan liittyvät asiat ovat mielenkiintoinen näkökulma, minkä myös Satrom (2018) ottaa esille: hänen mukaansa sovelluskehystä valittaessa kannattaa ottaa huomioon, että valittava sovelluskehys on sellainen, johon on helppo löytää tekijöitä; sellaisia, jotka jo osaavat kyseisen teknologian tai sellaisen, jonka opetteleminen ei ole hankalaa.

Työmarkkinaan liittyvät seikat kuten teknologioiden *houkuttelevuus* on mielenkiintoinen ilmiö mitä olisi syytä tutkia tulevaisuuden tutkimuksissa: monissa alan työpaikkailmoituksissa korostetaan *moderneja teknologioita,* mutta teknologioiden houkuttelevuus ja se, mikä teknologiasta tekee houkuttelevan, ei tuota juurikaan hakutuloksia etsittäessä tieteellisiä artikkeleita.

6.7 Valintaan vaikuttavat toimijat

Kaikissa haastatteluissa sovelluskehyksen valinnasta päätti kehitystiimi. Itse prosessissa keskustelussa oli mukana muitakin henkilöitä, mutta varsinaisen päätöksen teki yleensä ohjelmistokehittäjä tai -arkkitehti.

Vaikka asiakasta ei yleensä oteta mukaan käyttöliittymäsovelluskehyksen valintaprosessiin, asiakkaan asettamat vaatimukset projektille voivat kuitenkin vaikuttaa käyttöliittymäsovelluskehyksen valintaan, ja asiakas näin ollen vaikuttaa sovelluskehyksen valintaan epäsuorasti. Edellisissä alaluvuissa esiteltiin, kuinka asiakkaan esittämä vaatimus sovelluksen erityisen korkeasta suorituskykyisyydestä voi vaikuttaa sovelluskehyksen valintaan. Sama pointti toistui modulaarisuutta tai laajennettavuutta tarkastellessa: jos projektin vaatimukset

edellyttävät, että sovelluskehyksen lisäksi joudutaan ottamaan käyttöön useita eri lisäkirjastoja, ei silloin ole juuri väliä, onko käytössä yksi laaja sovelluskehys, vai otetaanko käyttöön kevyempi sovelluskehys lukuisine lisäosineen.

6.8 Malli käyttöliittymäsovelluskehysten hyväksymiselle

6.8.1 Käyttöliittymäsovelluskehyksille epäolennaiset osa-alueet

Edellisissä alaluvuissa havaittiin, että muutamia Panon ym. (2018) mallin osa-alueita ei koettu tärkeiksi suuressa osassa haastatteluita ja joitakin osa-alueita ei noussut esille missään haastattelussa. Voidaan siis tulkita, että kyseiset osa-alueet ovat epäolennaisia käyttöliittymäsovelluskehystä valittaessa. Tällaisia osa-alueita olivat:

- Koko
- Automaatio
- Sopivuus
- Eristyneisyys

Ainoastaan yhdessä haastattelussa sovelluskehyksen avulla kirjoitetun ohjelman koodirivien määrä ja sovelluksen koko nousi esille. Sitä ei siis nähdä lähtökohtaisesti kovinkaan tärkeänä tekijänä käyttöliittymäsovelluskehystä valittaessa. Koko voidaan tavalla sisällyttää toiseen alakohtaan, *monimutkaisuuteen*, sillä vähemmän koodia sisältävä sovellus voidaan nähdä vähemmän monimutkaisena.

Panon ym. (2018) mallissa automaatiota ei käyty kovinkaan laajalti läpi. Modernille käyttöliittymäsovelluskehykselle automaattiset DOMin manipulaatiot ja komponenttien päivitykset ovat ikään kuin oletuksena. Näin ollen ehdotan, ettei automaatiota tarvitse ottaa huomioon käyttöliittymäsovelluskehystä valittaessa.

Myöskään sopivuutta ei mainittu juurikaan; sopivuus nousi esille lähinnä, kun keskusteltiin käyttöliittymäsovelluskehyksen laajennoksista, mutta suoraan käyttöliittymäsovelluskehyksen valintaan sopivuudella ei ollut vaikutusta, sillä kaikki käyttöliittymäsovelluskehykset ratkaisevat saman ongelman: käyttöliittymän esittämisen. Toki sopivuutta käsitteenä mietitään sikäli, kun pohditaan, mikä käyttöliittymäsovelluskehys *sopii* parhaiten kyseiselle tiimille, mutta tällöin ei puhuta itse käyttöliittymäsovelluksen teknisten ominaisuuksien sopivuudesta vaatimuksiin nähden. Tulkitsisin kuitenkin, että Panon ym. (2018) mallin sopivuudessa on kyse enemmänkin sovelluskehyksen sopivuudesta käyttötarkoitukseensa, joten tässä käyttöliittymäsovelluskehysten yhteydessä sopivuus on epäolennainen tekijä ottaen huomioon aikaisemmin mainitsemani perustelut.

Eristyneisyys on käyttöliittymäsovelluskehysten osalta oletettua, eikä sitä ole mitään syytä ottaa huomioon valintaa tehdessä. Modernilla käyttöliittymäsovelluksella tehty yksisivuinen web-sovellus on jo oletusarvioisesti oma sovelluksensa, eikä sitä pysty sekoittamaan sovelluksen palvelinpuoleen. Kaluža & Vukelić, 2018 listaavatkin tämän yhtenä yksisivuisen web-sovellusten

hyötyinä: käyttöliittymää koskeva koodi on erillään käyttöliittymäpuolella palvelinpuolen sijaan ja se tarjoaa ohjelmoijille selkeän erittelyn, jotta voidaan keskittyä yhteen asiaan kerrallaan. Lisäksi Gong, Gu, Chen & Wang (2020) toteavat, että asiakaspään ja palvelinpään toisiinsa sotkeminen johtaa lukuisiin ongelmiin, ja asiakas- ja palvelinpään erottaminen toisistaan erillisinä mikropalveluina on yksi tyypillisimmistä lähestymistavoista web-sovelluksen kehittämiseen. Gongin ym. (2020) mukaan käyttöliittymäsovelluskehykset mahdollistavat palvelin- ja asiakaspään eristämisen toisistaan.

6.8.2 Samankaltaisten osa-alueiden yhdistäminen

Useassa haastattelussa puhuttiin *suosittuudesta*, eikä niinkään määritelty erikseen tuliko käsitys suosittuudesta kollegoilta vai siitä, mitä sovelluskehyksiä esimerkiksi kilpailijat käyttävät. Sekä kilpailija-analyysi että kollegojen neuvo voitaisiin yhdistää saman kategorian alle, mikä voisi käsittää koetun suosittuuden. Tähän samaan voitaisiin yhdistää myös kehitystiimin kokemukset kyseisestä sovelluskehyksestä. Tätä tekijää voisi kuvastaa nimellä *kokemukset*, ja se käsittäisi sekä positiiviset että negatiiviset kokemukset kyseisestä sovelluskehyksestä – oli ne sitten valintaa tekevän henkilön omia kokemuksia, tai mitä hän on kuullut muilta.

Toinen sosiaalisen vaikutuksen alakohta voisi käsittää *ekosysteemin,* mikä pitäisi sisällään sekä yhteisön koon ja yhteisön reagointikyvyn. Näiden osa-alueiden yhdistäminen käy järkeen, sillä mitä suurempi yhteisö on, sitä reaktiivisemman sen voidaan olettaa olevan. Tämä oletus oli vahvasti esillä myös haastatteluissa:

> H6: ”[...] laajalti käytössä oleva framework, jolloin myös löytyy enemmän tilannekohtaisia kysymyksiä netistä esim. StackOverFlow’sta”

> H2: ”Kyllä se on aika varma oletus, että jos tietää, että sitä käytetään laajalti, niin kyllä siihen silloin löytyy tukea ja tietoa hyvin.”

Myös Panon ym. (2018) mukaan yhteisön koko menee käsi kädessä havaitun reagointikyvyn kanssa, joten mielestäni kyseisiä osa-alueita on turha pitää erillään.

Kolmas ehdotus samankaltaisten osa-alueiden yhdistämisestä on laajennettavuuden ja modulaarisuuden yhdistäminen. Tätä teemaa sivuttiin jo aikaisemmin luvussa 3.2.4. Pano ym. (2018) mainitsevat ulkoisten kirjastojen helpon liittämisen sekä modulaarisuuden, että laajennettavuuden yhteydessä, eikä edellä mainitut termit juurikaan eroa toisistaan käyttöliittymäsovelluskehysten yhteydessä. Ehdotukseni on, että käytettäisiin vain yhtä termiä, *modulaarisuus*, joka käsittäisi sen, että itse sovelluskehys olisi modulaarinen, eli siihen ei tarvitsisi välttämättä sisällyttää kaikkia moduuleita kerrallaan, sekä sen, että sovelluskehys olisi helposti laajennettava, eli sitä voitaisiin laajentaa helposti ulkoisilla kirjastoilla.

6.8.3 Käyttöliittymäsovelluskehysten valintaan vaikuttavien osa-alueiden relaatiot

Haastatteluissa pystyttiin havaitsemaan relaatioitta tiettyjen osa-alueiden välillä: Sovelluskehyksen ympärillä oleva ekosysteemi vaikuttaa sovelluskehyksen päivityksiin, ymmärrettävyyteen, monimutkaisuuteen ja opittavuuteen: suosittuus johtaa siihen, että keskustelupalstoilta löytyy enemmän keskustelua ja ohjeistuksia kehittäjille, tehden sovelluskehyksestä selkeämmän ja helposti opittavan.

> H2: ”Kyllä se on aika varma oletus, että jos tietää, että sitä käytetään laajalti, niin kyllä siihen silloin löytyy tukea ja tietoa hyvin”
>
> H5: ”Oikeastaan nämä kaikki voi tavallaan johtaa siitä suosittuudesta. Se että on suosittu, johtaa siihen, että sitä käytetään, ja sitä halutaan ja sille löytyy tekijöitä ja sille tulee päivityksiä
>
> [...]
>
> En erityisemmin [ottanut selvää dokumentaatiosta]. Se on osa sitä et kannattaa ottaa suosittu. Löytyy matskua, oli sitten virallista tai käyttäjien dokumentaatiota””
>
> H6: ” [kysyttäessä dokumentaation tärkeydestä] Hyvin tärkeänä. ja se toki liittyy myös siihen, että on laajalti käytössä oleva framework, jolloin myös löytyy enemmän tilannekohtaisia kysymyksiä netistä esim. StackOverFlow’sta”

Päivitykset mainittiin usein sovelluskehyksen suosittuuden yhteydessä:

> H4: ”Kyl sillä frameworkin ympärillä pitää olla riittävästi kehittäjiä, sekä siinä ite päähommassa, siinä pääkomponentissa, plus sitten kaikissa lisäosissa, jotta se pysyy hengissä”
>
> H6: ” Oletus oli, ja se oli lähtöoletus et se on semmonen iso, tunnettu ja käytetty framework ja se projekti ei niiku jää kesken, vaan se elää ja menee eteenpäin”

Tästä voitaisiin päätellä, että käyttöliittymäsovelluskehyksen ympärillä oleva ekosysteemi vaikuttaa kaikkiin vaivattomuusodotuksiin sekä sovelluskehyksen päivityksiin.

6.8.4 Käyttöliittymäsovelluskehyksen valintaan vaikuttavat toimijat ja niiden vaikutukset eri osa-alueisiin

Panon ym. (2018) mallissa on mukana neljä eri sovelluskehyksen valintaan vaikuttavaa toimijaa: Asiakas, kehittäjä, tiimi sekä tiimin johtaja. Tässä tutkimuksessa tiimi sekä kehittäjä päätettiin yhdistää *kehittäjätiimiksi*, sillä päätöksestä vastaa haastattelujen perusteella tyypillisesti kehittäjätiimi yhden kehittäjän sijaan.

Haastatteluissa ei noussut esille juuri muita päätökseen vaikuttavia toimijoita kuin sovelluskehyksen parissa työskentelevät sovelluskehittäjät. Asiakkaalla on kuitenkin epäsuora vaikutus sovelluskehyksen valintaan asettamalla sovellusta käsittelevät vaatimukset, jotka voivat vaikuttaa myös käyttöliittymäsovelluskehyksen valintaan.

KUVIO 10 Käyttöliittymäsovellukseen valintaan vaikuttavat toimijat

Haastatteluissa havaittiin kehittäjätiimin vaikutuksia käyttöliittymäsovelluskehysten valintaa vaikuttaviin tekijöihin: kehittäjätiimin kokemukset ja aikaisempi osaaminen teknologiasta tai vastaavista teknologioista vaikuttaa olennaisesti kaikkiin vaivattomuusodotuksiin sekä kokemuksiin käyttöliittymäsovelluskehyksestä: kehittäjätiimin

Yksi asiakkaan asettamista, käyttöliittymäsovelluksen valintaan vaikuttava vaatimus sovellukselle on suorituskyky, eli jos asiakkaalta nousee vaatimus erityisen hyvästä suorituskyvystä, kannattaa silloin painottaa valintaa suorituskyvyltään tehokkaaseen sovelluskehykseen. Muussa tapauksissa suorituskykyä ei nähty juurikaan tärkeäksi, sillä suorituskykyerot ovat käyttöliittymäsovelluskehysten välillä melko marginaalisia, mutta joissain määrin eroja kuitenkin löytyy.

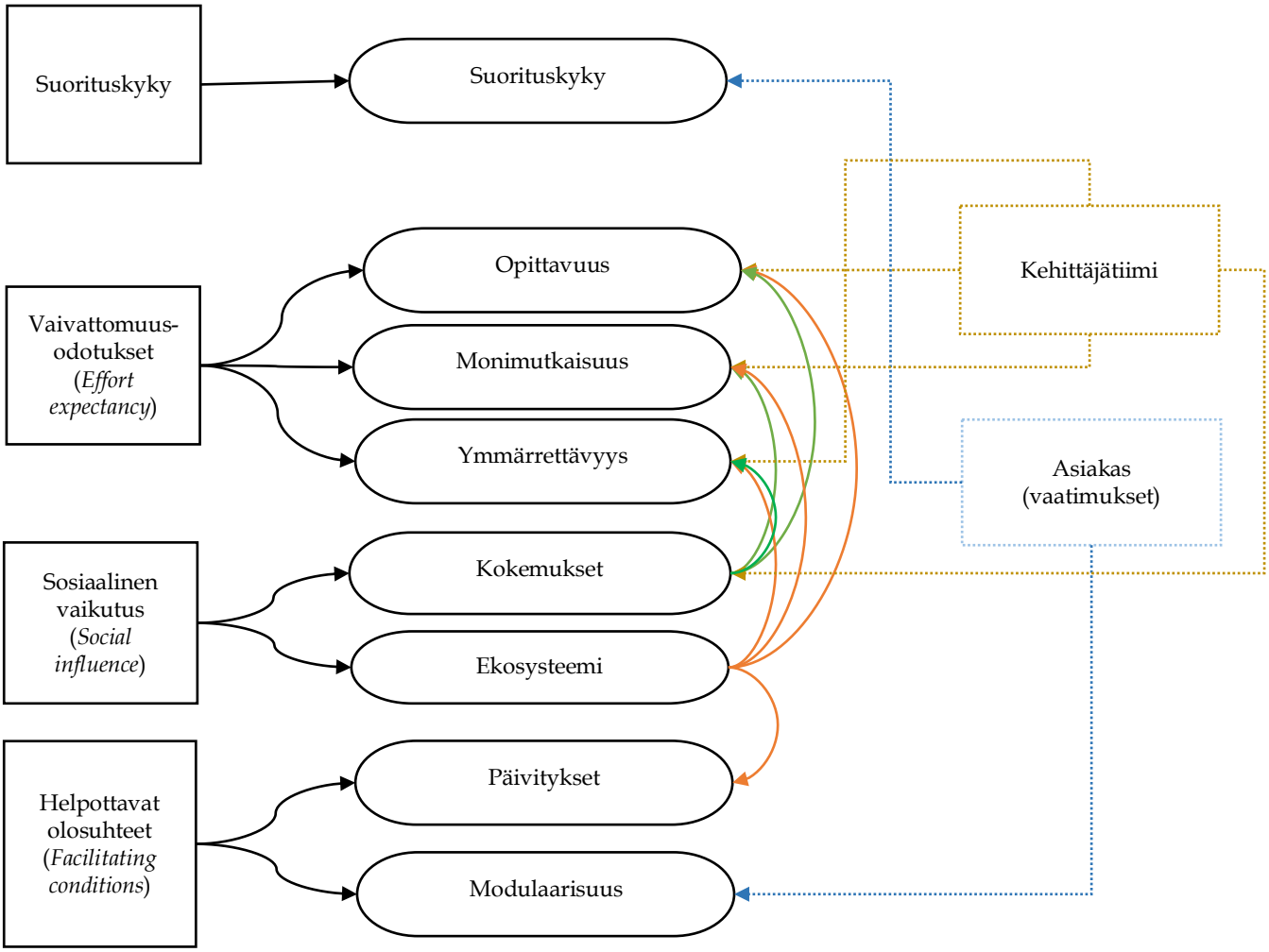
Toinen tekijä, johon asiakas vaikuttaa, on modulaarisuus. Haastateltaessa haastateltavaa numero 6 nousi esille, että Angularin vähäinen modulaarisuus on epäolennainen valintakriteeri, sillä kyseisen haastateltavan edustamalle yritykselle on tyypillistä, että projektit, joita yritys tekee, vaativat usein suuren osan Angularin ominaisuuksista: Jos sovellus tehtäisiin jollakin toisella sovelluskehyksellä, jouduttaisiin mukaan kuitenkin ottamaan suuri määrä moduuleita, eikä modulaarisuudella tällöin saavutettaisi juuri mitään etua. Asiakkaalta siis tunnistettiin kaksi relaatiota eri osa-alueisiin: vaikuttavuussuhde suorituskyvyn sekä modulaarisuuden merkityksellisyyteen.

Asiakkaan asettamien vaatimusten lisäksi kehittäjätiimin aikaisempien kokemusten ja preferenssien myötä löydettiin relaatioita eri sovelluskehyksen valintaa vaikuttaviin tekijöihin: Kehittäjätiimin aikaisempi kokemus vaikuttaa olennaisesti sovelluskehyksen kaikkiin vaivattomuusodotuksiin, eli opittavuuteen, monimutkaisuuteen ja ymmärrettävyyteen, sekä lisäksi kokemuksiin kehittäjien omien kokemuksien myötä sovelluskehyksestä.

6.8.5 Yhteenveto käyttöliittymäsovelluksen valintaan vaikuttavista tekijöistä

Käyttöliittymäsovelluskehyksen valintaan vaikuttavia tekijöitä on onnistuttu tunnistamaan sekä käyttöliittymäsovelluskehyksille epäolennaisia tekijöitä on pystytty karsimaan Panon ym. (2018) mallista. Kyseisen mallin pohjalta on tehty alla olevaan kuvioon ehdotus mallista käyttöliittymäsovelluskehyksen valintaan vaikuttavista tekijöistä, toimijoista sekä niiden välisistä keskinäisistä vaikutussuhteista.

Malli käyttöliittymäsovelluskehysten valintaa vaikuttavista tekijöistä poikkeaa osa-alueidensa osalta Panon ym. (2018) JavaScript-sovelluskehysten valintaa vaikuttavista tekijöistä siten, että suorituskykyodotuksista on karsittu pois alakohta koko, sekä vaivattomuusodotuksista on poistettu *automaatio* edellä perusteltujen seikkojen vuoksi. Sosiaalisen vaikutuksen alakohdat on yhdistetty selkeyden vuoksi kahteen eri alakohtaan: kokemuksiin ja ekosysteemiin. Helpottavista olosuhteista puolestaan on sopivuus ja eristyneisyys, ja yhdistetty *modulaarisuus* ja *laajennettavuus* samaksi alakohdaksi, modulaarisuudeksi.



KUVIO 11 Käyttöliittymäsovelluskehykseen vaikuttavat tekijät ja toimijat ja niiden väliset vaikutussuhteet Panon ym. (2018) malliin pohjautuen

Kuvioon 11 on koottu yhteen tässä tutkimuksessa havaitut käyttöliittymäsovelluskehyksen valintaan vaikuttavat tekijät ja niiden keskinäiset vaikutussuhteet. Oikealle on listattu käyttöliittymäsovelluskehyksen valintaan vaikuttavat toimijat, sekä havainnollistettu toimijoiden vaikutukset eri valintaan vaikuttaviin tekijöihin.

Alla olevaan taulukkoon (TAULUKKO 6) on kerätty yhteenveto haastatteluissa nousseista käyttöliittymäsovelluskehyksen valintaan vaikuttavista tekijöistä noudattaen tässä kappaleessa esitettyä mallia. Taulukosta voidaan päätellä, että kaikista eniten käyttöliittymäsovelluskehyksen valintaan vaikuttavat sosiaaliset tekijät sekä vaivattomuusodotukset: sekä ekosysteemi että kokemukset nousivat keskustelussa jokaisessa haastattelussa valintaan vaikuttavana tekijänä, sekä myös sovelluskehyksen monimutkattomuus ja opittavuus koettiin tärkeinä. Myös sovelluskehyksen päivitykset nähtiin merkittävänä tekijänä, tärkeää oli, että sovelluskehyksen elinkaari säilyisi.

TAULUKKO 6 Käyttöliittymäsovelluskehyksen valintaan vaikuttavat tekijät

Haastateltava	Suorituskykyodotukset	Vaivattomuusodotukset			Sosiaalinen vaikutus		Helpottavat olosuhteet	
	Suorituskyky	Opittavuus	Monimutkaisuus	Ymmärrettävyys	Ekosysteemi	Kokemukset	Päivitykset	Modulaarisuus
H1	E	M	M	M	M	M	-	M
H2	V	J	M	E	M	M	M	-
H3	V	M	E	M	M	M	-	-
H4	V	E	M	E	J	J	M	M
H5	V	M	M	E	M	M	M	-
H6	E	M	M	M	M	M	M	E

Koettiin merkittäväksi tekijäksi
Koettiin jossain määrin merkittäväksi tekijäksi
Ei koettu merkittäväksi tekijäksi
Tiedostettiin mahdollisesti merkittäväksi jos esitetty erillisenä vaatimuksena
Asia ei noussut esille haastattelussa

Alla olevaan taulukkoon (TAULUKKO 7) on kerätty yhteenveto tämän tutkimuksen empiirisen datan pohjalta käyttöliittymäsovelluskehyksen valintaan vaikuttavista tekijöistä. Taulukosta voidaan havaita, että sovelluskehyksen sosiaalisen vaikutuksen alla olevan tekijät, ekosysteemi ja kokemukset koetaan tärkeimmiksi tekijöistä sovelluskehystä valittaessa. Heti perässä seuraa vaivattomuusodotukset, joista valintaan vaikuttavat pääosin sovelluskehyksen monimutkaisuus ja opittavuus. Suorituskyky nousi esiin vaikuttavana tekijänä vain silloin, jos kyseinen seikka oli nostettu esille projektin vaatimuksissa.

TAULUKKO 7 Yhteenveto käyttöliittymäsovelluskehyksen valintaan vaikuttavista tekijöistä tämän tutkimuksen empiirisessä datassa

Valintaan vaikuttava tekijä	Koettiin merkittäväksi tekijäksi	Koettiin jossain määrin merkittäväksi tekijäksi	Merkittävä tekijä jos erillinen vaatimus esitetty
Ekosysteemi	5	1	
Kokemukset	5	1	
Monimutkaisuus	5		
Opittavuus	4	1	
Päivitykset	4		
Ymmärrettävyys (dokumentaatio)	3		
Modulaarisuus	2		
Suorituskyky			4

7 YHTEENVETO JA POHDINTA

Kvalitatiivinen teemahaastattelu tarjosi paljon laadukasta tietoa käyttöliittymäsovelluskehysten valintaan vaikuttavista tekijöistä. Sovelluskehyksen valintaan vaikuttavia tekijöitä tunnistettiin lukuisa määrä ja niitä onnistuttiin yhdistämään Panon ym. (2018) malliin. Lisäksi onnistuttiin muodostamaan Panon ym. (2018) mallin pohjalta malli käyttöliittymäsovelluskehyksen valintaan johtavista tekijöistä.

7.1 Tutkimuksen tavoitteet

Tutkimuksen tavoitteena oli vastata seuraaviin tutkimuskysymyksiin: ”Mitkä eri seikat vaikuttavat ohjelmistokehitysprojektin käyttöliittymäsovelluskehyksen valintaan?”, ”Mitä eri mittareita käyttöliittymäsovelluskehysten vertailussa voidaan käyttää?”, ”Mitkä kriteerit koetaan tärkeimmiksi käyttöliittymän sovelluskehystä valittaessa?” sekä ”Miten projektin luonne vaikuttaa käyttöliittymäsovelluskehyksen valintaan?”

Käyttöliittymäsovelluskehysten valintaan vaikuttavia seikkoja tarkasteltiin luvussa kuusi käyttäen pohjana Panon ym. (2018) mallia JavaScript-sovelluskehysten valintaan vaikuttavista tekijöistä. Tutkimuksessa havaittiin, että kaikki samat tekijät, mitä Pano ym. (2018) mainitsevat tutkimuksessaan, eivät ole olennaisia rajattaessa tarkastelua pelkästään käyttöliittymäsovellusten kontekstiin. Havaitut käyttöliittymäsovelluskehysten valintaan vaikuttavat tekijät olivat:

- suorituskyky
- opittavuus
- ymmärrettävyys
- sovelluskehyksen ympärillä oleva ekosysteemi
- kokemukset sovelluskehyksestä
- sovelluskehyksen (ja sen lisäosien) päivitykset, elinkaari
- modulaarisuus

Samaisia tekijöitä voidaan käyttää myös vastaamaan kysymykseen numero kaksi käyttöliittymäsovelluskehysten vertailemiseen käytettävistä mittareista. Käyttöliittymäsovelluskehysten valintaan vaikuttavat toimijat olivat pääasiassa kehitystiimi, mutta valintaan vaikutti myös epäsuorasti asiakas asettamillaan vaatimuksilla projektille. On kuitenkin huomioitava, että yksi havainto oli, ettei suorituskykyä ole syytä ottaa mukaan valintaan vaikuttavana tekijänä, eikä näin ollen vertailukriteerinä, ellei erityisen hyvä suorituskyky ole erikseen määriteltynä vaatimuksena kehitettävälle sovellukselle. Samaten modulaarisuutta ei nähty tärkeänä tekijänä vaikuttamassa valintaan silloin, kun sovellus oli vaatimuksiltaan monimutkainen.

Kolmas tutkimuksen tutkimuskysymys koskee käyttöliittymäsovelluskehysten valintaan tärkeimmiksi koettuja kriteereitä. Taulukossa 6 oli merkitty tähdellä kunkin haastateltavan mainitsema tärkeimmäksi koettu kriteeri ja esille nousi ***ekosysteemi***, ***kokemukset*** sekä ***opittavuus***. Sama voidaan tulkita myös puhtaasti tärkeäksi mainittujen osa-alueiden lukumäärissä taulukosta 7, jossa myös *monimutkaisuus* nousi esille tärkeänä osa-alueena. Tärkeimmät käyttöliittymäsovelluskehykseen vaikuttavat tekijät voidaan siis katsoa olevan ***ekosysteemi***, ***kokemukset***, ***opittavuus*** ja ***monimutkaisuus***.

Tutkimuksen neljäs ja viimeinen tutkimuskysymys käsitteli projektin luonteen vaikutusta käyttöliittymäsovelluskehyksen valintaan. Tämä nousi esille lähinnä kehitettävän sovelluksen vaatimusten vaikutuksesta sovelluskehyksen valintaan, ja mitä jo sivuttiin tässäkin luvussa; sovelluskehyksen suorituskyvyllä nähtiin olevan merkitystä vain silloin, jos sovellukselle oli asetettu erillinen vaatimus erityisen hyvästä suorituskyvystä. Toinen havainto projektin luonteen vaikutuksesta käyttöliittymäsovelluskehyksen valintaan liittyi sovelluskehyksen modulaarisuuteen: kun kehitettävä sovellus on riittävän monimutkainen vaatimuksiltaan, ei modulaarisuutta nähdä silloin niin tärkeänä tekijänä, joten ei ole merkitystä, sisältyvätkö toiminnallisuudet itse sovelluskehykseen vai tuodaanko ne sovellukseen moduuleina. Kolmas havainto liittyi projektitiimin kokoon: opittavuutta ei nähty niinkään tärkeäksi silloin, kun kehitystiimi koostui vain yhdestä henkilöstä: olettama on, että kun tiimi koostuu suuresta määrästä kehittäjiä, halutaan tällöin käyttöön helposti opittava sovelluskehys, sillä se joudutaan opettamaan usealle eri kehittäjälle.

7.2 Implikaatiot

Tutkimuksessa onnistuttiin rakentamaan Panon ym. (2018) JavaScript-sovelluskehysten valintaa vaikuttavien tekijöiden mallin pohjalta malli käyttöliittymäsovelluskehysten valintaan vaikuttavista tekijöistä. Mielenkiintoista oli huomata, ettei suorituskykyä koettu lainkaan merkittävänä tekijänä, ellei suorituskyky ollut erityisen tärkeässä osassa kehitettävän sovelluksen vaatimusten vuoksi. Aikaisemmin sovelluskehyksiä vertailtaessa suorituskyky oli yksi useimmiten vertailluista mittareista, eikä sosiaalista näkökulmaa ollut otettu huomioon ennen Panon ym. (2018) havaintoja.

Tutkimuksessa kehitetty malli käyttöliittymäsovelluskehysten valintaan vaikuttavista tekijöistä tarjoaa hyvää teoriapohjaa alan tutkimukselle: kyseistä mallia ja siinä käytettyjä operationalisointeja voitaisiin käyttää kvalitatiivisissa tutkimuksissa. Lisäksi kvalitatiivisen tutkimuksen havainnot tarjoavat mielenkiintoisia oletuksia, joita voidaan todentaa ja laajentaa kvantitatiivisin tutkimuksin.

Tämän tutkimuksen havainnoista on hyötyä myös ammatinharjoittajille: sovelluskehyksiä vertailtaessa voidaan käyttää tutkimuksessa rakennettua mallia, josta voidaan havaita valintaan vaikuttavat tekijät ja käyttää sovelluskehyksiä vertailtaessa.

7.3 Tutkimuksen rajoitteet

Kaikki tutkimukseen osallistuneet haastateltavat työskentelivät konsulttitaloissa, joten tuotetalojen näkökulmaa käyttöliittymäsovelluskehysten valintaa vaikuttavista tekijöistä ei saatu tässä tutkimuksessa. Tämäkin on itseasiassa mielenkiintoinen jatkotutkimuksen kannalta: koetaanko tuotetaloissa samat seikat tärkeiksi ja onko valintaan vaikuttavat toimijat samat? Lisäksi kaikki haastateltavat työskentelivät suomalaisissa organisaatioissa – voisivatko tekijät kuten esimerkiksi organisaation koko ja ketteryys tai demografiset tekijät olla sellaisia, mitkä vaikuttaisivat sovelluskehyksen valintaprosessiin?

Luvussa 5.2 käsiteltiin laadullisen tutkimuksen luotettavuutta. Tutkimuksessa oli mukana kuusi haastateltavaa, jotka tarjosivat hyvän määrän kiinnostavaa dataa. Haastateltavien suhteellisen vähäisestä määrästä huolimatta tutkimuksessa saavutettiin saturaatio, ja havaittiin, ettei suurempi määrä haastateltavia tarjoisi välttämättä uutta, tutkimuksen kannalta mullistavaa tietoa. Suuremman mittakaavan kvalitatiivinen tutkimus olisi kuitenkin tarpeen, jotta dataa saataisiin laajemmin ja erilaisista organisaatioista.

7.4 Jatkotutkimus

Kappaleessa 6.6.3 sivuttiin teknologioiden - erityisesti käyttöliittymäsovelluskehysten houkuttelevuutta kehittäjille työmarkkinassa: monessa organisaatiossa pohditaan, kuinka houkutella kehittäjiä firmoihin, ja monessa työpaikkailmoituksessa mainitaan *modernit teknologiat* houkuttimena kehittäjille. Teknologioiden houkuttelevuus ja sen käyttäminen rekrytointivalttina on mielenkiintoinen aihe, josta ei löydy juurikaan tutkimustietoa.

Pano ym. (2018) mainitsevat artikkelissaan tarpeen kvantitatiiviselle tutkimukselle heidän listaamille käyttöliittymäsovelluskehyksen valintaan vaikuttaville ominaisuuksille, jotta metriikkaa voidaan kehittää eteenpäin. Tällaista tutkimusta ei ole vieläkään tehty, ja se olisi varmasti tarpeellinen. Vaikka tässä tutkimuksessa onnistuttiin tarkentamaan Panon ym. (2018) mallia käyttöliittymäsovelluskehysten kontekstiin, olisi kvantitatiivinen tutkimus aiheesta

varmasti paikallaan – olisi se sitten käyttöliittymäsovelluskehyksistä tai sovelluskehyksistä ylipäänsä. Lisäksi jatkotutkimuksessa voitaisiin lähestyä sovelluskehysten valintaan vaikuttavia tekijöitä esimerkiksi asiakaspään ja palvelinpään sovelluskehysten vertailun näkökulmasta; vaikuttavatko samat asiat palvelinpään sovelluskehysten valintaan kuin käyttöliittymäsovelluskehysten valintaan?

Kiinnostava haastatteluissa noussut teema oli työmarkkinaan liittyvät asiat, jotka eivät välttämättä liittyneet suoraan sovelluskehyksiin: miten teknologiavalintoja voidaan käyttää kilpailuvalttina IT-alan rekrytoinnissa ja mikä tekee teknologiavalinnasta houkuttelevan ja trendikkään? Tästä aiheesta ei löydy juurikaan aikaisempaa tutkimusta ja siitä olisi varmasti hyötyä käytännön ammatinharjoittajille – myös käyttöliittymäsovelluskehykset ovat vahvasti brändättyjä ja herättävät kehittäjissä tiettyjä mielikuvia: asiaa harvemmin mietitään, mutta taustalla on varmasti ollut monimutkainen brändinrakennusprosessi, jossa tarkoituksena on rakentaa sovelluskehykselle pysyvä ekosysteemi.

Sovelluskehykset ovat ammatinharjoittajien näkökulmasta varsin jokapäiväinen ilmiö ja ne nousevat IT-alan työpaikoilla keskusteluun jatkuvasti. Sovelluskehykset ovat kuitenkin tieteellisen tutkimuksen osalta suhteellisen vähän tutkittu ilmiö. Käytännön työelämän ja tieteellisen tutkimuksen välillä on ilmiön osalta siis suuri kuilu, joten aiheella on selkeä tarve lisätutkimukselle.

LÄHTEET

Angular. (2021). *Angular.* https://angular.io/(Viitattu 6. joulukuuta 2021)

Aggarwal, S. (2018). Modern web-development using reactjs. International Journal of Recent Research Aspects, 5(1), 133-137.

Bierman, G., Abadi, M. & Torgersen, M. (2014). Understanding TypeScript. ECOOP 2014 – object-oriented programming (s. 257-281). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-662-44202-9_11

C. M. Novac, O. C. Novac, R. M. Sferle, M. I. Gordan, G. BUJDOSó & C. M. Dindelegan. (2021). Comparative study of some applications made in the Vue.js and React.js frameworks. (s. 1-4) doi:10.1109/EMES52337.2021.9484149

Eskola, J. & Suoranta, J. (1998). Johdatus laadulliseen tutkimukseen. Vastapaino.

Ferreira, F., Borges, H. S. & Valente, M. T. (2021). On the (un‐)adoption of JavaScript front‐end frameworks. Software, Practice & Experience, doi:10.1002/spe.3044

Findel, H. & Navon, J. (2015). A test environment for web single page applications (SPA) SCITEPRESS - Science and and Technology Publications. doi:10.5220/0005428000470054

Github. (2021a). *Vue*. https://github.com/vuejs/vue/ (Viitattu 6. joulukuuta 2021)

Github. (2021b). *Angular*. https://github.com/angular/angular/ (Viitattu 6. joulukuuta 2021)

Github. (2021c). *React*. https://github.com/facebook/react/ (Viitattu 6. joulukuuta 2021)

Gitstar Ranking (2022). *Repositories Ranking*. https://gitstar-ranking.com/repositories/ (Viitattu 11. maaliskuuta 2022)

Gizas, A., Christodoulou, S. & Papatheodorou, T. (2012). Comparative evaluation of javascript frameworks. (s. 513-514) ACM. doi:10.1145/2187980.2188103

Gong, Y., Gu, F., Chen, K., & Wang, F. (2020). The Architecture of Micro-services and the Separation of Frond-end and Back-end Applied in a Campus Information System. In 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA) (pp. 321-324). IEEE.

Graziotin, D. & Abrahamsson, P. (2013). Making sense out of a jungle of JavaScript frameworks: Towards a practi- tioner-friendly comparative analysis. Springer-Verlag.

Halstead, M. H. (1977). Elements of Software Science (Operating and programming systems series). Elsevier Science Inc..

Hsieh, H. F., & Shannon, S. E. (2005). Three approaches to qualitative content analysis. Qualitative health research, 15(9), 1277-1288.

Jadhav, M. A., Sawant, B. R. & Deshmukh, A. (2015). Single page application using angularjs. International Journal of Computer Science and Information Technologies, 6(3), 2876-2879.

Kaluža, M. & Vukelić, B. (2018). Comparison of front-end frameworks for web applications development. Zbornik Veleučilišta U Rijeci, 6(1), 261-282. doi:10.31784/zvr.6.1.19

McCabe, T. J. (1976). A complexity measure. IEEE Transactions on Software Engineering, SE-2(4), 308-320. doi:10.1109/TSE.1976.233837

Meta Open Source. (2021). *React.* https://opensource.fb.com/projects/react/ (Viitattu 6. joulukuuta 2021)

Mozilla. (2021) *What is JavaScript?* https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (Viitattu 5. joulukuuta 2021)

O. C. Novac, D. E. Madar, C. M. Novac, G. Bujdosó, M. Oproescu & T. Gal. (2021). Comparative study of some applications made in the angular and vue.js frameworks. (s. 1-4) doi:10.1109/EMES52337.2021.9484150

Pano, A., Graziotin, D. & Abrahamsson, P. (2018). Factors and actors leading to the adoption of a JavaScript framework. Empirical Software Engineering : An International Journal, 23(6), 3503-3534. doi:10.1007/s10664-018-9613-x

Quora. (2018). *Why is Vue.js trending more in China?* https://www.quora.com/Why-is-Vue-js-trending-more-in-China/ (Viitattu 11. maaliskuuta 2022)

Saaranen-Kauppinen, A., & Puusniekka, A. (2009). Menetelmäopetuksen tietovaranto KvaliMOTV. Kvalitatiivisten menetelmien verkko-oppikirja. Yhteiskuntatieteellisen tietoarkiston julkaisuja.

Satrom, B. (2018). Choosing the Right JavaScript Framework for Your Next Web Application. Prog./Kendo UI, 34.

Shahzad, F. (2017). Modern and responsive mobile-enabled web applications. Procedia Computer Science, 110, 410-415.

State of JavaScript. (2021). *State of JavaScript 2020.* https://2020.stateofjs.com/ (Viitattu 6. joulukuuta 2021)

Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User acceptance of information technology: Toward a unified view. MIS quarterly, 425-478.

Vue-View. (2022). *Why Vue.js is trending in China?* https://vue-view.com/why-vue-js-is-trending-in-china-how-popular-vue-is-there/ (Viitattu 11. maaliskuuta 2022)

Welker, K. D. (2001). The software maintainability index revisited. CrossTalk, 14, 18-21.

Williams, N. (2022). *Medium.* https://medium.com/@aretheregods/vue-is-enormously-popular-in-china-and-chinese-speaking-countries-more-popular-than-either-react-31379ae34be0 (Viitattu 11. maaliskuuta 2022)

Yue, C., & Wang, H. (2009, April). Characterizing insecure JavaScript practices on the web. In Proceedings of the 18th international conference on World wide web (pp. 961-970).

Xing, Y., Huang, J. & Lai, Y. (2019). Research and analysis of the front-end frameworks and libraries in E-business development. (s. 68-72) ACM. doi:10.1145/3313991.3314021

LIITE 1 HAASTATTELURUNKO

Suorituskykyodotukset (Performance expectancy)

Otitteko sovelluskehyksen valintaa tehdessä huomioon sovelluskehyksen suorituskykyä? (Latausnopeus tms.)

Vaivattomuusodotukset (Effort expectancy)

Kuinka tärkeänä valintakriteerinä piditte sovelluskehyksen helppoa opittavuutta?

Oliko sovelluskehyksen monimutkaisuudella merkitystä valintaa tehdessä?

Sosiaalinen vaikutus (Social Influence)

Otitteko selvää sovelluskehyksen suosittuudesta?

Otitteko selvää siitä, kuinka hyvin esim. StackOverflow'sta löytyy ratkaisuja ongelmiin / keskustelua?

Oliko tiimin aikaisemmalla kokemuksella sovelluskehyksestä vaikutusta valintaan?

Helpottavat olosuhteet

Oliko muilla jo valituilla sovelluskehyksillä tai taustajärjestelmillä vaikutusta sovelluskehyksen valintaan?

Vaikuttiko frameworkin päivitykset valintaan?

Muuta:

Tuleeko mieleen muita tekijöitä, joilla voisi olla ollut vaikutusta sovelluskehyksen valintaan?

Mikä oli mielestäsi tärkein päätökseen johtanut seikka?

Ketä oli mukana valintaprosessissa? Otettiinko mahdollinen asiakas mukaan valintaprosessiin?