Arttu Matilainen

WEB-SOVELLUSPROJEKTISSA KÄYTETTÄVÄN VIITEKEHYKSEN VALINTA

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2021

TIIVISTELMÄ

Matilainen, Arttu
Web-sovellusprojektissa käytettävän viitekehyksen valinta
Jyväskylä: Jyväskylän yliopisto, 2020, 28 s.
Tietojärjestelmätiede, kandidaatintutkielma
Ohjaaja(t): Seppänen, Ville

JavaScript-viitekehysten yleistyessä, ja valittavien vaihtoehtojen lisääntyessä ei välttämättä ole aina selvää, mikä valittavissa olevista viitekehyksistä on paras omaan käyttötarkoitukseen. Aihetta on pyritty tutkimaan aikaisemmin kartoittamalla haastattelun keinoin tekijöitä, joita kehittäjät pitävät tärkeinä viitekehystä valitessaan, mutta varsinaista viitekehysten vertailua näiden kriteerien nojalta on tehty hyvin vähän. Tämä tutkielma pyrki tuottamaan aiempaan tutkimukseen ja kirjallisuuteen nojaten viitekehysten vertailussa käytettävän mallin, ja vertailemaan yleisesti käytettyjä viitekehyksiä näin valikoiduin kriteerein. Vertailulla saavutettujen löydösten pohjalta suoritettiin analyysi, josta selvisi viitekehysten vahvuudet ja heikkoudet toisiinsa nähden. Käytetyillä tutkimusmetodeilla ei päästy selkeään käsitykseen siitä, onko jokin viitekehyksistä yksiselitteisesti muita parempi, mutta tuloksista voidaan päätellä tapauksia, joissa on edullista käyttää jotain niistä ennen muita.

Asiasanat: web-viitekehykset, javascript, reactjs, angular, vuejs

ABSTRACT

Matilainen, Arttu
Choosing a framework to be used in a web-application project
Jyväskylä: University of Jyväskylä, 2020, 28 pp.
Information Systems Science, Bachelor's thesis
Supervisor(s): Seppänen, Ville

As the amount of available JavaScript frameworks in the market increases, it is not always necessarily easy to make the decision on which of these frameworks is the most suitable for one's specific use case. Previous research on factors that developers consider important when making the decision has been conducted by interviewing developers, but little research has been conducted that evaluates the available frameworks based on the suggested criteria. This thesis aimed to produce a model to be used in evaluating frameworks by reviewing existing research and literature. Three commonly used frameworks were then evaluated using this model, and the findings were then analysed to see the pros and cons of each of the analysed frameworks compared to each other. Using these methods, it was not possible to gain definite insight on the superiority of any of the compared frameworks, but judging from the results, it is clear to see that some of the frameworks are clearly better suited for certain types of projects than others.

Keywords: web frameworks, javascript, reactjs, angular, vuejs

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Viitekehykset ovat olennainen osa web-sovelluskehitystä, ja niitä on eri käyttötarkoituksiin ja ohjelmointikielelle tarjolla useita. Jo pelkästään JavaScriptille – joka on Stack Overflow:n vuonna 2020 toteuttaman käyttäjätutkimuksen (Stack Overflow Developer Survey, 2020) mukaan käytetyin ohjelmointikieli 67,7% vastaajista käyttäessä sitä – on tarjolla monia viitekehyksiä, joista tässä tutkielmassa tarkastellaan kolmea tällä hetkellä samaisen tutkimuksen mukaan ajankohtaisinta JavaScript-viitekehystä: Reactia, Angularia, ja Vuea.

Tässä tutkielmassa etsitään vastausta kahteen tutkimuskysymykseen:

1) Mitkä tekijät vaikuttavat web-viitekehyksen valintaan? ja
2) Kuinka vertaillut viitekehykset eroavat toisistaan näiden tekijöiden osalta, ja onko jokin niistä sopivampi valinta tietynlaiseen projektiin?

Ensimmäiseen tutkimuskysymykseen tässä tutkielmassa etsitään vastausta kirjallisuuskatsauksen keinoin luvussa 3, erittelemällä aikaisempia tutkimuksia viitekehysten vertailuun liittyen. Tähän vaiheeseen lähdekirjallisuutta on etsitty asiasanoilla:

- JavaScript
- Web Framework
- Evaluating
- Comparing
- React
- Angular
- Vue

sekä näiden hakusanojen yhdistelmät. Tarkempaan tarkasteluun päätyi lopulta kaksi tekstiä, niiden vastatessa ensimmäiseen tutkimuskysymykseen lähes täysin. Nämä olivat Graziotinin, Abrahamssonin ja Panon (2016) kirjoittama ”Rationale leading to the adoption of a JavaScript framework”, joka on kattava haastatteluihin pohjautuva artikkeli JavaScript-viitekehysten valintaan

vaikuttavien tekijöiden erittelytavoista, sekä Brandon Satromin (2017) kirjoittama ”Choosing the Right JavaScript Framework for Your Next Web Application” joka erittelee tarkemmin juuri tarkasteluun valittuja viitekehyksiä, ja esittelee myös oman ehdotuksensa kriteeristöstä, jolla viitekehyksiä voidaan verrata. Näiden tekstien yhtäläisyyksiä ja eriävyyksiä vertaillen ja analysoiden johdetaan tässä työssä käytettävä malli, johon nojaten luvussa 4 vertaillaan viitekehyksiä toisiinsa.

Luvussa 4 esitellään tarkemmin vertailuun valitut viitekehykset, sekä vertaillaan niitä toisiinsa aiemmin kirjallisuuskatsauksen tuloksista johdetun mallin mukaisesti. Tässäkin vaiheessa vertailussa käytetty tieto viitekehyksistä on hankittu lähdekirjallisuutta mahdollisuuksien mukaan hyödyntäen, sekä viitekehysten virallista dokumentaatiota käyttäen.

Tätä aihetta on aikaisemmin tutkittu enemmänkin teorian tasolla, ja ajankohtaista viitekehyksien systemaattista erittelyä ei juuri ole tehty. Aihetta vastaavat tutkimukset käsittelevätkin joko vertailumalleja ylipäänsä, tai viitekehysten vertailua tietyltä kantilta, kuten suorituskyky. *Graziotin ym.* (2016) esittävät tekstissään jatkotutkimuksen tarpeeksi vertailuanalyysiä, josta on käytännöllistä tosielämän hyötyä. Tämä tutkielma pyrkii vastaamaan tähän tarpeeseen soveltamalla viitekehysten vertailumallia yleisesti käytössä oleviin viitekehyksiin. Tästä informaatiosta arvokasta tekee se, että oikean web-viitekehyksen käytöllä voidaan lisätä kehittäjän tehokkuutta, nopeuttaa kehitysprosessia, sekä mahdollisesti näin ollen leikata kustannuksia. Tästä syystä on tärkeä saada käsitys siitä, mitkä ovat olennaisia tekijöitä web-viitekehyksen valinnassa. Tämä tutkielma pyrkii luomaan mallin, joka on hyödyllinen sekä pienissä että suurissa projekteissa käytettävien viitekehyksien valinnassa, sekä soveltamaan mallia viitekehyksiin, joita voidaan käyttää sekä suurissa, että pienissä web-ohjelmistoprojekteissa.

2 KÄSITTEET

Tässä luvussa käsitellään tutkielman olennaisimpia käsitteitä, ja tuodaan ilmi, mistä näkökulmasta niitä tässä työssä tarkastellaan. Joidenkin käsiteltävien teknologioiden kohdalla luokittelu viitekehyksen ja kirjaston välillä on tulkinnanvaraista, ja usein varsinkin puhekielessä viitekehyksen määritelmä voi sisältää myös laajempia kirjastoja. Tässä luvussa pyritään perustelemaan sitä, miten työssä käsiteltävät teknologiat on luokiteltu valittuihin kategorioihin.

2.1 Web-viitekehys

Internetin alkuaikoina kaikki applikaatiot ohjelmoitiin manuaalisesti. Tästä seurasi oletettavasti enemmän virheitä ja prosessiin kului huomattavasti enemmän aikaa. Tämä ongelma on ajankohtainen myös nykyaikana, ja erityisesti laajojen web-applikaatioiden kehittämisen haastavuus johtuu siitä, että ohjelmalogiikka on jakautunut asiakasohjelmiston ja palvelinohjelmiston välille. Tästä syystä koodin jakaminen näiden kahden välillä on vaikeaa. (Richard-Foy, Barais, and Jézéquel, 2014)

Web-viitekehykset ovat käytännössä työkaluja, joiden tarkoitus on helpottaa verkkosivujen rakentamista, auttaa välttämään virheitä ja säästämään aikaa. *Curien ym*. (2019) mukaan viitekehykset voidaan jakaa kahteen kategoriaan, asiakaspuolen (*client side*) ja palvelinpuolen (*server side)* viitekehykset. Heidän määritelmänsä mukaan asiakaspuolen viitekehykset ovat vastuussa käyttöliittymän implementointiin liittyvistä ominaisuuksista, kuten animaatioista ja asettelumalleista. Esimerkkeinä muun muassa *Angular*, *Vue*, ja *React*. Palvelinpuolen vastuualoja sen sijaan ovat logiikka, rajapintakutsut, tietokantayhteydet sekä sivuston arkkitehtuuri. Jotkin palvelinpuolen viitekehykset saattavat sisältää myös joitain tietoturvaan vaikuttavia tekijöitä. Esimerkkejä palvelinpuolen viitekehyksistä ovat muun muassa Django, Ruby on rails, Nodejs ja ExpressJS. (Curie ym., 2019)

Edellä mainitusta kaksijakoisesta määritelmästä huolimatta on olemassa myös viitekehyksiä, jotka yhdistävät asiakas- ja palvelinpuolen ominaisuudet yhdeksi kokonaisuudeksi. Tällaisia ovat esimerkiksi Reactiin pohjautuva Next.js sekä Vue.js:ään pohjautuva Nuxt.js. Näiden ominaisuuksiin lukeutuvat myös sivustojen hahmonnus palvelinpuolella (server side rendering) sekä staattinen sivujen generointi (static site generation). Sivujen hahmonnus palvelinpuolella tarkoittaa, että sivuston HTML-koodi luodaan palvelimella pyydettäessä (request time) (Server render | Jamstack, 2020). Staattisella sivujen generoinnilla tarkoitetaan, että sen sijaan että sivut generoitaisiin pyydettäessä (request time), eli esimerkiksi linkkiä painettaessa, ne generoidaan jo sivua kootessa. Näin ollen käyttäjän pyytäessä palvelimelta sivua, hän saa sen valmiiksi generoituna ja hahmonnettuna (What is a Static Site Generator? How do I find the best one to

use?, 2020). Tämä tarkoittaa käytännössä sitä, että sivuston lataamisen tarvittava aika vähenee huomattavasti.

2.1.1 Asiakaspuoli

Toteutukseltaan sivustot voivat olla joko yhden sivun sovelluksia *(Single Page Application, SPA)* tai monen sivun sovelluksia (*Multiple Page Application, MPA*). *Kaluža ja Vukelić* (2018) erittelevät MPA- ja SPA-sovelluksia eroja seuraavasti: MPA-sovellukset ovat useimmiten laajempia järjestelmiä, jotka sisältävät useampia erilaisia palveluita. SPA-sovellukset ovat uudempi lähestymistapa web-kehitykseen, ja ovat useimmin yksinkertaisempia sivustoja, joilla on vähemmän sisältöä (Kaluža ja Vukelić, 2018).

MPA-sovellukset toimivat ”perinteisellä” tavalla; jokaiseen muutokseen selaimessa, kuten datan liikkumiseen tai näyttämiseen, liittyy uusien sivujen hakemista palvelimelta (Neoteric 2016). Sivujen polut on rekisteröity palvelimelle, ja jokainen http-pyyntö asiakkaalta palvelimelle johtaa uuden HTML-sivun hakemiseen. Tästä johtuen jokaisen pyynnön seurauksena on joko pyydetty sivu, tai virheilmoitus. Suurin osa ohjelmalogiikasta sijaitsee palvelimella, ja asiakaspääte on ainoastaan haetun sivun vastaanottaja.

Scottin (2016) mukaan SPA on kokonainen web-sovellus, joka koostuu vain yhdestä sivusta, joka toimii ”kuorena” käyttöliittymän kaikille osille. Suurin osa resursseista (Javascript, HTML5, CSS) ladataan vain kerran sovelluksen ajon aikana, ja data siirtyy edellisestä tilasta (*state*) uuteen. Sovelluksen avaamisen yhteydessä lähetetty HTML-dokumentti toimii lähtöpisteenä sovelluksen muille osille. Kaikki muut osat ladataan dynaamisesti ja ”kuoresta” riippumatta, uudelleenlataamatta koko sivua, luoden käyttäjälle käsityksen siitä, että sivu on vaihtunut. Kuori on rakenteeltaan minimaalinen, ja sisältää usein yhden ainoan tyhjän tagin ( ) jonka sisälle loput sovelluksen sisällöstä rakentuu (Scott, 2016).

2.1.2 Palvelinpuoli

Palvelinpuolen viitekehyksistä tässä työssä olennaisin on Node. *Shahzadin* (2017) mukaan Node.js tai Node on viitekehys ja runtime-ympäristö, joka mahdollistaa selaimen ulkopuolella ajettavien Javascript-ohjelmien kehittämisen. Node.js-viitekehys sisältää APIn (*Application Program Interface*) muunmuassa tiedostojärjestelmän kanssa vuorovaikuttamiseen, tietokantoihin yhdistämiseen ja http-pyyntöjen kuunteluun (Shahzad, 2017). Tässä tutkielmassa palvelinpuolta tarkastellaan tutkielmaan valittujen viitekehysten osalta vain siinä määrin, kuin viitekehysten arviointimalliin nojaten on järkevää.

2.2 Kirjasto

Tietotekniikassa kirjastot (mm. ohjelmakirjasto, luokkakirjasto) ovat kokoelmia, aliohjelmia, luokkia ja/tai ohjelmia, joita käytetään tietokoneohjelmien modulaarisessa kehittämisessä sekä ohjelmien suorittamisen aikana. Tällaisen koodin uudelleenkäytön on laajalti todettu olevan olennainen työkalu tuotettaessa ohjelmistoja nopeasti ja kustannustehokkaasti (Abdalkareem, 2017).

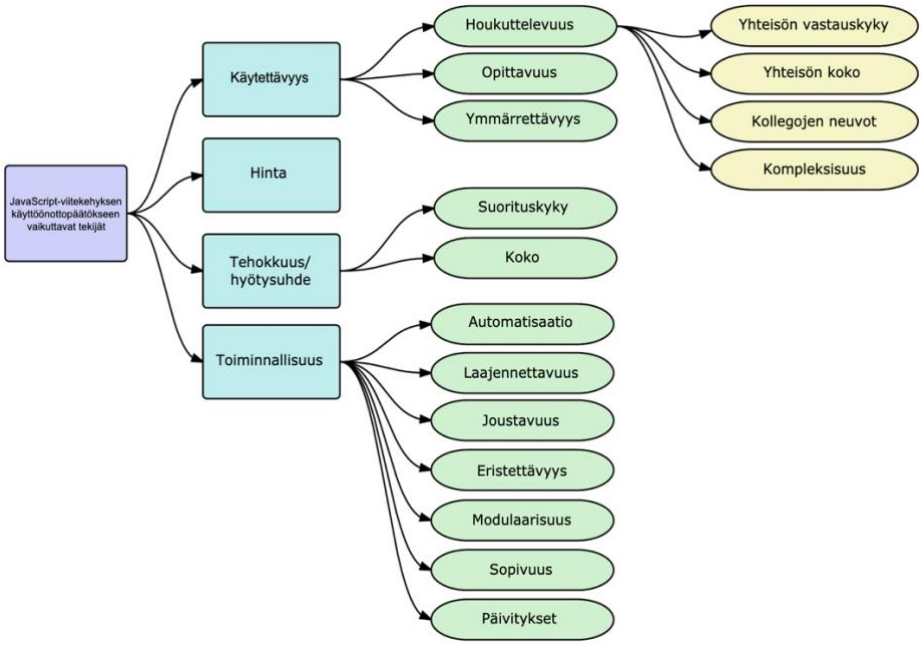
Web-kehityksessä kirjastot ovat suuressa roolissa, pääosin npm (*node package manager*) -pakettien muodossa. Tässä tutkielmassa tarkasteltuja viitekehyksiä käytettäessä projektiin lisätään usein toiminnallisuutta laajentavia npm-paketteja tai muita lisäkirjastoja.

Tässä työssä kirjaston tarkempi määritelmä on sisällytetty juuri siksi, että lisäkirjastot ovat olennainen osa web-kehitystä, kun viitekehys on käytössä.

3 VIITEKEHYKSEN VALINTAAN VAIKUTTAVAT TEKIJÄT

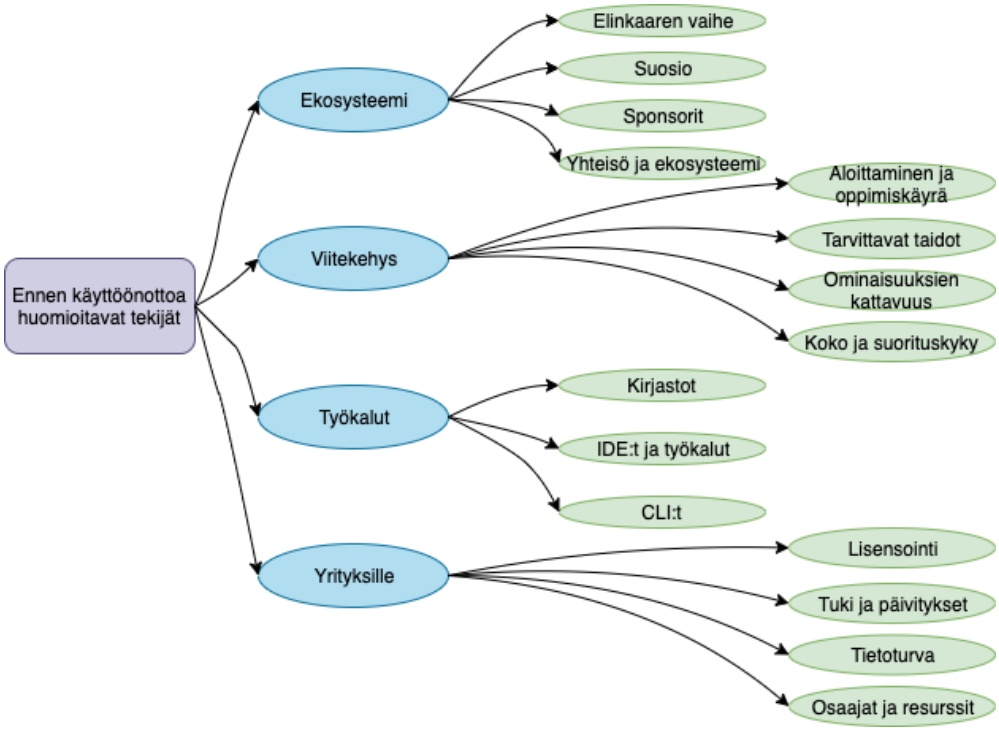
Web-viitekehystä valittaessa tulee vaihtoehtoja tarkastella useasta eri näkökulmasta. Tässä luvussa esitellään kirjallisuuskatsauksen muodossa kahdessa eri tutkimuksessa kuvatut kriteerit, joilla viitekehyksiä niiden tekijöiden mukaan voidaan evaluoida ennen käyttöönottoa. Lisäksi näitä kahta mallia verrataan toisiinsa, ja niiden pohjalta johdetaan tässä tutkielmassa käytettävät kriteerit, joilla viitekehyksiä arvioidaan luvussa 4.

Graziotin ym. (2016) erittelevät tutkimuksessaan tekijöitä, jotka vaikuttavat JavaScript-viitekehyksen käyttöönottopäätökseen. Pääkategorioiksi ovat valikoituneet käytettävyys, hinta, tehokkuus ja toiminnallisuus. Näiden alakategoriat on eritelty tarkemmin seuraavassa kuvaajassa (Kuvio 1).



Kuvio 1 JavaScript viitekehyksen käyttöönottopäätökseen vaikuttavat tekijät (Graziotin ym., 2016) (käännetty suomeksi)

Seuraavassa kuvaajassa (Kuvio 2) on puolestaan kuvattu *Satromin* (2017) esittämät tekijät, jotka hän nostaa tärkeiksi huomioida ennen viitekehyksen käyttöönottoa. Pääkategorioiksi ovat valikoituneet ekosysteemi, viitekehys, työkalut sekä yritystoiminnassa huomioitavat seikat.



Kuvio 2 Ennen käyttöönottoa huomioitavat tekijät (Satrom, 2017). (käännetty suomeksi, oma kuvio)

Verrattaessa näitä kahta mallia toisiinsa voidaan huomata, että pääkategorioiden eroavuuksista huolimatta yhtäläisyyksiä on malleissa paljon. Näistä yhtäläisyyksistä yksi on *yhteisö. Graziotin ym., (2016)* haastattelivat tutkimuksessaan ammattilaisia, jotka ovat olleet erinäisten JavaScript-kirjastojen kanssa tekemisissä vähintään 4 vuotta. He mainitsevat, että haastatteluissa puhuttaessa viitekehyksen houkuttelevuudesta ja suosiosta, haastateltavat mainitsivat usein yhteisöt viitekehyksen kehityksessä. (Graziotin ym., 2016.) Yhteisön koko ja vastauskyky ovat olennaisia tekijöitä, kun tarkastellaan kehittäjän työskentelytehokkuutta ja kykyä oppia käyttämään viitekehystä. Kohdatessaan ongelman kehittäjä voi kääntyä yhteisön puoleen, ja parhaassa tapauksessa jollakin muulla yhteisön jäsenellä on jo ollut sama ongelma ja hänen kysymykseensä on vastattu, tai kyseisestä ongelmasta saattaa olla kirjoitettu artikkeli. Mitä suurempi yhteisö viitekehyksellä on, sitä helpompaa apua on saada. *Satrom* (2017) puolestaan nostaa esiin myös yhteisön kyvyn tuottaa tukikirjastoja, ja hänen mukaansa esimerkiksi viitekehyksen käyttäjälle saatavilla olevien tukikirjastojen ja liitännäisten määrä, laskettuna npm-pakettien määrästä, voi olla hyvä mittari sille kuinka terve yhteisö viitekehyksellä on (Satrom, 2017). Huomattakoon, että myös *Graziotin ym.,* (2016) ovat ottaneet laajennettavuuden lisäkirjastoilla huomioon, mutta eivät varsinaisesti yhteisön tuottamien pakettien näkökulmasta. Molemmissa malleissa päivitykset on sisällytetty mukaan erillisenä ominaisuutena. *Graziotin ym.* (2016) lukevat päivitykset osaksi toiminnallisuuksia, kun taas *Satrom* (2017) listaa päivitykset osana tekijöitä, jotka ovat olennaisia yrityksen näkökulmasta tarkastellessa. Päivitykset voidaan kuitenkin sisällyttää käsitteen ”yhteisö”-alle, johtuen siitä, että viitekehyksiä ja

niiden tukikirjastoja on kehittämässä monessa tapauksessa nimenomaan yhteisö, varsinkin puhuttaessa avoimen lähdekoodin kirjastoista, ja suurempi tai aktiivisempi yhteisö tässä tapauksessa voi vaikuttaa päivitysten tiheyteen.

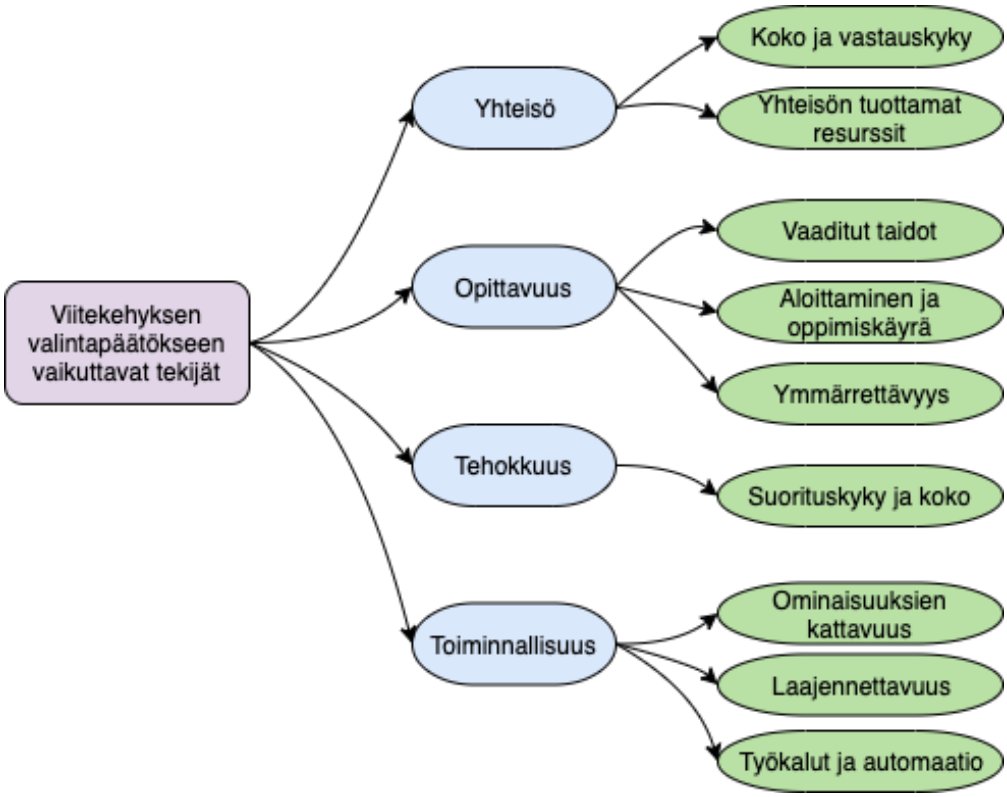
Seuraava toistuva tekijä on viitekehyksen *opittavuus*. *Graziotin ym.* (2016) tarkoittavat opittavuudella tutkimuksessaan sitä, kuinka paljon vaivaa kehittäjän on nähtävä oppiakseen käyttämään viitekehystä. He mainitsevat, että nämä vaatimukset ovat riippuvaisia kehittäjän teknisistä taidoista ja kokemuksesta. *Satrom* (2017) sen sijaan on erottanut mallissaan viitekehyksen käyttöön tarvittavat taidot sen oppimisesta, ja nostaa esiin oppimiskäyrän käsitteen (Satrom, 2017). Tämä on olennainen lisäys, sillä viitekehyksen käyttö voi olla hyvinkin helppo aloittaa, mutta siirryttäessä edistyneempiin käyttötarkoituksiin vaativuustaso voi nousta jyrkästi. Opittavuuden käsitteen alle voitaneen sisällyttää myös *Graziotinin ym.* (2016) erikseen mainitsema käsite ymmärrettävyys, jolla he tarkoittavat viitekehyksen piirteitä, jotka auttavat käyttäjää tunnistamaan, kuinka viitekehystä voidaan soveltaa käytäntöön. Tällaisia piirteitä ovat heidän mukaansa kattava dokumentaatio ja selkeä koodin rakenne.

Seuraava merkittävä yhtäläisyys on *suorituskyky* tai *tehokkuus*. *Satrom* (2017) on sisällyttänyt suorituskyvyn käsitteen malliinsa yhdessä viitekehyksen koon kanssa. Hänen määritelmänsä mukaan suorituskyky mittaa sitä, kuinka viitekehys suoriutuu kompleksisessa sovelluksessa, ja sitä, miten viitekehyksen ominaisuudet parantavat kehitetyn sovelluksen ajonopeutta. Koolla hän tarkoittaa viitekehyksen käyttöön tarvittavien kooditiedostojen kokoa kilotavuina. (Satrom, 2017.) *Graziotin ym.* (2016) ovat sisällyttäneet molemmat edellä mainituista käsitteistä käsitteen ”tehokkuus”-alle. Heidän määritelmässään tehokkuutta mitataan sovelluksen suorituskyvyn perusteella, mitaten sovelluksen ajonopeutta ja käytettyjä laitteistoresursseja, kuten muistia ja kaistaa. Suorituskyvyn alakäsitteeseen he sisällyttävät myös sen, kuinka monta riviä koodia tarvitaan toteuttamaan yksinkertainen toiminnallisuus. Koolla he tarkoittavat koko käännetyn paketin kokoa. Liian suuri paketti voi olla liian raskas suorittaa esimerkiksi älypuhelimen selaimella. (Graziotin ym., 2016.)

Viimeinen olennainen yhtäläisyys on *toiminnallisuus*. Toiminnallisuuden käsitteessä mallien välillä on hieman selkeämpiä eroja. *Satromin* (2017) mallissa toiminnallisuuden alle lukeutuvia käsitteitä voitaneen katsoa olevan ominaisuuksien kattavuus, jolla hän tarkoittaa vertailua sen välillä, tarjoaako viitekehys valmiit työkalut ja toimintamallit koko projektin skaalalle, vai onko viitekehys keskittynyt spesifimmän osa-alueen toteuttamiseen. Esimerkkinä tästä hän mainitsee *Angularin*, joka sisältää kaiken olennaisen käyttöliittymän hallinnasta kompleksiseen tilanhallintaan, reitittämiseen, testaamiseen ja muuhun. Toisena esimerkkinä hän mainitsee *Reactin*, joka on keskittyneempi käyttöliittymän toteutukseen. Reactin tapauksessa kehittäjän on siis sisällytettävä projektiin muita kirjastoja saadakseen käyttöönsä ominaisuuksia kuten kompleksinen tilanhallinta tai reitittäminen sivujen tai näkymien välillä. *Graziotin ym.* (2016) ottavat ominaisuuksien kattavuuteen kantaa lähinnä toiminnallisuuden alakäsitteenä määrittelemänsä sopivuuden yhteydessä.

Heidän havaintojensa mukaan viitekehys, joka voi luoda kokonaisen applikaation on etulyöntiasemassa, sillä se nähdään hyvänä ratkaisuna ajan säästämiseen ja sopivien kirjastojen etsimisen välttämiseen. Toisaalta he ovat maininneet laajennettavuuden yhtenä viitekehykseen valintaan vaikuttavista tekijöistä. *Graziotin ym.* (2016) nostavat mallissaan esiin myös liudan muita toiminnallisuuksia, kuten automatisaatio. Tällä viitataan viitekehyksen kykyyn automatisoida kehitystyötä joiltain osin. (Graziotin ym., 2016.) Tätä toiminnallisuutta voidaan katsoa tukevan myös *Satromin* (2017) mallissa mainitut työkalut-kategorian alla olevat IDE:t ja työkalut, sekä CLI:t (*command line interface*, komentorivityökalu). Erityisesti CLI:t ovat Satromin määritelmässä suuressa osassa tätä automaatioprosessia. Mikäli viitekehys tarjoaa komentiorivityökaluja, joilla voidaan esimerkiksi luoda uusi projekti kaikkine riippuvaisuuksineen sekä projektin perusrakenne valmiiksi paikallaan, se nähdään olevan eduksi. Muita automatisaatiota hyödyntäviä ominaisuuksia ovat hänen mukaansa esimerkiksi erilaiset testityökalut, koodin oikeinkirjoitusta tarkastavat *lintterit,* sekä pakkaus- ja käyttöönottotyökalut. (Satrom, 2017.)

Tässä tutkielmassa käytetty vertailumalli koostuu siis edellä johdetuista kategorioista: yhteisö, opittavuus, tehokkuus sekä toiminnallisuus. Näiden sisällään pitämät huomioon otettavat tekijät on kuvattu seuraavassa kuvaajassa (Kuvio 3). Niiden valikoituminen on perusteltu edeltävissä tekstikappaleissa.



Kuvio 3 Kirjallisuuskatsauksen pohjalta johdettu vertailumalli

Seuraavassa luvussa esitellään jokainen tarkasteluun valittu viitekehys yksi kerrallaan, ja analysoidaan niitä valittujen kriteerien mukaisesti.

4 VIITEKEHYSTEN VERTAILU

Tässä työssä käytetyt vertailukriteerit on valittu kirjallisuuskatsauksessa esiteltyjä tutkimuksia yhdistellen, valikoiden kriteerien joukosta ne kohdat, jotka parhaiten vastaavat tutkimuskysymyksiin, eli millä perustein tietty viitekehys tulee valituksi projektiin, ja mitä eroa tutkielman kirjoittamishetkellä yleisimmin käytössä olevilla viitekehyksillä on näillä kriteereillä verrattuna.

Tässä työssä eritellään kolmea web-viitekehystä, jotka ovat kaikista laajimmin käytetyt tutkielmaa kirjoittaessa. Nämä viitekehykset ovat Facebookin ylläpitämä React, Googlen ylläpitämä Angular, sekä Vue, joka ei ole minkään yrityksen tukema tai ylläpitämä, vaan sen kehityksestä ja ylläpidosta vastaa Vuen ydintiimi ja kehittäjäyhteisö. Seuraavissa alaluvuissa esitellään ensin viitekehys, jonka jälkeen sitä tarkastellaan mallin kohtien mukaisesti. Vertailussa lähteenä on käytetty viitekehyksistä tehtyjä aiempia tutkimuksia, viitekehysten virallisia dokumentaatioita, sekä Stack Overflow:n tuottamaa käyttäjätutkimusta vuodelta 2020.

4.1 React

Aggarwalin (2018) mukaan React on interaktiivisten käyttöliittymien kehittämiseen tarkoitettu Facebookin ylläpitämä komponenttipohjainen kirjasto, jota käyttäen voidaan kehittää uudelleenkäytettäviä käyttöliittymäkomponentteja. Reactin avulla on mahdollista kehittää laajoja ja kompleksisia web-applikaatioita, joiden data voi muuttua ilman perättäisiä sivun uudelleenlatauksia.(Aggarwal, 2018.)

4.1.1 Opittavuus

Satromin (2017) mukaan React vaatii kehittäjää käyttämään modernia JavaScriptia (ES6+). Reactissa käytetään viitekehyksen tasolla useita ES6-spesifikaatin konsepteja, kuten nuolifunktioita ja moduuleja, ja näiden ymmärtäminen on kehittäjälle olennaista hänen voidakseen käyttää tätä viitekehystä. JSX:n käyttö Reactissa on suurin eroava tekijä muista viitekehyksistä, ja vaikka sen käyttö ei ole pakollista, suurimmassa osassa koodiesimerkeistä sitä on kuitenkin käytetty, joten sen ymmärtämisestä on hyötyä kehittäjälle. Reactin käyttö edellyttää myös HTML:n ja CSS:n ymmärtämistä.

Reactin helppous ja suoraviivaisuus mahdollistaa viitekehyksen nopean omaksumisen. Reactissa käytetty JSX (JavaScript XML) on hyvin luonteva tapa kirjoittaa komponentteja. (Aggarwal, 2018.) JSX näyttää hyvin paljon perinteiseltä HTML-merkintäkieleltä, huomattavimpina eroina on, että mukana ovat kehittäjän kustomoimat React-komponentit, ja JSX-koodin ollessa

pohjimmiltaan JavaScriptiä, sekaan voi kirjoittaa JavaScriptiä ja näin lisätä toiminnallisuutta komponentteihin helposti. Reactin kääntäjä (*compiler*) kääntää JSX:n puhtaaksi JavaScriptiksi. Tämän vuoksi jopa aloittelevan kehittäjän tai pelkästään HTML:ää tuntevan henkilön, kuten graafisen suunnittelijan, on helppo ymmärtää React-koodia. Reactin virallisessa dokumentaatiossa sanotaan myös, että React on suunniteltu alusta pitäen asteittaista käyttöönottoa varten, ja sitä voi projektissaan käyttää niin vähän tai niin paljon kuin haluaa. Virallisesta dokumentaatiosta löytyy myös kattavat ohjeet aloittamiseen, videoita viitekehykseen liittyvistä konsepteista ja esimerkkejä. Saatavilla on myös oppaat aloittelijoille ja suunnittelijataustaisille henkilöille, joille ohjelmoinnin käsitteet eivät vielä ole tuttuja. (React, 2020.)

4.1.2 Toiminnallisuus

Reactin käyttötarkoitus rajautuu erityisesti MVC-mallin (*Model-View-Controller*) View-ulottuvuuden toteutukseen (Aggarwal, 2018). Toisin sanoen Reactin erikoisalaa ovat käyttöliittymän ulkoasuun ja tietojen esitykseen liittyvät käyttökohteet. Tämän vuoksi React-projekti tarvitsee useita muita kirjastoja toimiakseen. Näitä ovat esimerkiksi jokin kääntäjä (esim. *Babel*), joka kääntää Reactissa käytetyn JSX-koodin selaimen ymmärtämäksi JavaScriptiksi, sekä projektin erinäisten moduulien paketointiin tarkoitettu *Webpack* tai vastaava kirjasto, joka pakkaa koko projektin kaikki tiedostot yhdeksi ajettavaksi minimoiduksi tiedostoksi. Usein React-projektit koostuvatkin useiden kirjastojen koostamasta yhdistelmästä, jolloin tätä kokonaisuutta voidaan pitää viitekehyksenä. Reactin tapauksessa kehittäjällä on siis vapaat kädet valita mistä osista tämä kokonaisuus koostuu.

Reactin pääasiallisiin toiminnallisuuksiin kuuluu mahdollisuus suunnitella yksinkertaisia näkymiä (*view*) sovelluksen jokaiselle tilalle (*state*), joita voidaan helposti päivittää datan muuttuessa, ja oikeat komponentit voidaan hahmontaa näkymään tarvittaessa. Komponentit voivat myös tarvittaessa hallita omaa tilaansa. (Ivanova ja Georgiev, 2019.) Yleisesti on kuitenkin pidetty parhaana lähestymistapana tehdä komponenteista ”tyhmiä”, eli ne eivät tiedä omasta tilastaan mitään, vaan jokin komponenttipuussa ylempänä oleva komponentti välittää niille muuttuvan datan ”proppeina” (*props*).

React tukee myös sivujen hahmontamista palvelimella, ennen kuin sivu lähetetään käyttäjälle, Node.js:n avulla. Reactia voidaan myös käyttää mobiiliapplikaatioiden toteuttamiseen käyttämällä React Nativea. (Ivanova ja Georgiev, 2019.)

React on myös laajennettavissa erilaisilla toiminnallisuutta kasvattavilla kirjastoilla, joita Reactille on tarjolla muihin verrattuihin viitekehyksiin verrattuna huomattavasti enemmän (ks. Taulukko 1). Mainittavia esimerkkejä näistä ovat esimerkiksi edistyneempään tilanhallintaan tarkoitettu React Redux

(React Redux, 2020), sekä React Router (ReactTraining/react-router: Declarative routing for React, 2020)

Työkalujen suhteen Reactilla on kattava tarjonta. React tarjoaa yksinkertaisien React-sovellusten tekemiseen komentoriviltä ajettavaa työkalua "*create-react-app*" joka perustaa lähtöpisteen projektille, mukaan lukien tiedostorakenteen, toimintaa tarvittavat tiedostot, sekä npm-skriptit, joilla kehitysympäristö voidaan ajaa paikallisesti, tai koota valmiiksi jakeluun sopivaksi paketiksi. Reactin dokumentaatio ohjaa myös edistyneemmissä käyttötarkoituksissa käyttämään edistyneempiä työkalusarjoja (*toolchain*). Näitä ovat Next.js, jota suositellaan käyttämään kehittäessä palvelimella hahmonnettuja sivustoja Node.js:ää hyödyntäen, sekä Gatsby, kun kehitetään staattisia, sisältöpainotteisia sivuja, jotka usein integroidaan johonkin sisällönhallintajärjestelmään (*content management system, CMS*). (Create a New React App – React, 2020.)

4.2 Angular

Ivanovan ja Georgievin (2019) mukaan Angular on TypeScriptiin pohjautuva JavaScript-viitekehys, jota kehittää ja ylläpitää Google. Se julkaistiin alun perin vuonna 2010 nimellä AngularJS. Angular (tunnettu myös nimillä Angular 2+, Angular 2 tai ng2) on AngularJS:n uudelleenkirjoitettu ja taaksepäin yhteensopimaton seuraaja, joka julkaistiin vuonna 2016. Tässä tutkielmassa tarkastellaan tätä uudempaa versiota. Uusin versio tutkielmaa kirjoittaessa on 11. Angular on suunnattu dynaamisten web-sovellusten kehittämiseen, ja sen ominaisuuksiin lukeutuvat kaikki yleisesti web-kehitysprojektissa toivotut ominaisuudet, muun muassa kaksisuuntainen datan sitominen, sapluunat (*templates*) komponenteille ja sivuille, sekä REST-API. (Ivanova and Georgiev, 2019.)

4.2.1 Opittavuus

Tarvittavat taidot, joita Angular edellyttää, ovat kuten muidenkin käsiteltyjen viitekehyksien tapauksessa HTML, CSS sekä JavaScript. Huomattavaa on kuitenkin, että Angular on kirjoitettu TypeScriptillä, ja sen tarkoituksenmukainen käyttö edellyttää sen osaamista. TypeScript vastaa käytännössä ES6-standardin mukaista JavaScriptia sillä olennaisella erolla, että toisin kuin JavaScript, TypeScript tukee nimensä mukaisesti datan vahvaa tyypitystä. (TypeScript Documentation, 2020.)

Angularilla on toisiin viitekehyksiin verrattuna suhteellisen jyrkkä oppimiskäyrä, joskaan ei heti alusta. Projektin valmisteleminen vie vain

muutamia minuutteja seuraamalla Angularin sivustolta löytyviä virallisia ohjeita (Angular, 2020). Angularin sivusto tarjoaa myös kurssin, joka esittelee yleisimmät Angularin toiminnallisuudet. Näillä opeilla voidaan kuitenkin toteuttaa vain yksinkertaisia sovelluksia, ja Angularin kattava oppiminen vaatii edistyneempien toiminnallisuuksien hallintaa. Kokeneelle kehittäjälle, jolla on kokemusta uusien teknologioiden oppimisesta, tämän ei kuitenkaan tulisi olla ongelma, sillä Angularin dokumentaatio (Angular Documentation, 2020) on varsin kattava, ja helposti ymmärrettävä.

4.2.2 Toiminnallisuus

Ivanovan ja Georgievin (2019) mukaan Angular on kattava viitekehys, tarkoittaen että kehittäjän ei tarvitse tehdä valintoja erilaisten ominaisuuksien, kuten reitittämisen toteuttamiseen tarvittavien pakettien tai kirjastojen kanssa, Angularin sisältäessä kaiken tarvittavan jo valmiiksi. (Ivanova ja Georgiev, 2019.)

Angularin virallinen verkkosivu mainitsee merkittävimpinä ominaisuuksina esimerkiksi mahdollisuuden tuottaa sovelluksia usealle kohdealustalle, kuten PWA:na (progressive web application) mobiililaitteille, asennettavina työpöytäsovelluksina Windowsille, MacOS:lle ja Linuxille, sekä natiivisovelluksina mobiililaitteille. Muita ominaisuuksia ovat nopeuteen ja suorituskykyyn vaikuttavat koodin generointi (code generation), joka tarkoittaa, että Angular kääntää kehittäjän kirjoittaman koodin JavaScript-virtuaalikoneille optimoituun muotoon, sekä koodin jakaminen (code splitting), joka tarkoittaa automaattista koodin jakamista siten, että käyttäjän tarvitsee ladata vain se koodi, joka milläkin hetkellä tarvitaan näkymän hahmontamiseen.

Angular tarjoaa kehitystyötä tukemaan myös monia työkaluja, kuten liitännäisiä useimpiin ohjelmointiympäristöihin, tarjoten esimerkiksi ennakoivaa koodin syöttöä ja virheilmoituksia ennen ajoa, sekä komentorivityökalut koontiin ja ajoon, kuten myös testaamiseen. Angular sisältää myös testaamiseen käytetyn yksikkötestaustyökalu Karman.

Lisäksi Angular ottaa myös kantaa saavutettavuuteen ARIAa (Accessible Rich Internet Applications) tukevin komponentein, sekä opastaa näiden luomisessa. Angularin sisäinen sovellusrajapinta sisältää myös funktiot komponenttien ja näkymien animointiin. (Angular - Features & Benefits, 2020.)

Angular on muiden verrattujen viitekehysten tavoin laajennettavissa lisäkirjastoilla, joskin tarjonta on huomattavasti esim. Reactia pienempi (ks. Taulukko 1).

4.3 Vue

*Vue*n virallinen dokumentaatio kuvaa Vue:ta progressiiviseksi viitekehykseksi. Tällä tarkoitetaan, että se on suunniteltu voitavaksi ottaa käyttöön asteittain, tai sitä mukaa kun kun käyttäjä itse haluaa. Koko sovelluksen ei siis esimerkiksi tarvitse olla kirjoitettu Vue:ta käyttäen. Vuen ydinkirjasto keskittyy ainoastaan näkymätason (*View layer*) toteuttamiseen, ja sen sanotaan olevan helppo omaksua ja integroida toisten kirjastojen tai olemassaolevien projektien kanssa. Vue:lla on myös mahdollista toteuttaa yhden sivun sovelluksia (SPA, single page application) jos sen kanssa käytetään moderneja työkaluja ja tukikirjastoja (Vue.js documentation, 2020).

4.3.1 Opittavuus

Vuen oppimiskäyrä on hyvin loiva, ja se on helppo integroida projekteihin. Vuen pääasiallisena ohjelmointikielenä toimii Reactin tavoin ES6-spesifikaattia noudattava JavaScript, joten sen ymmärtäminen on eduksi. Versiosta 2.5 lähtien Vue tukee myös TypeScriptiä, joskaan sen käyttö ei ole pakollista. Lisäksi Vuessa komponenttien kuvaaminen toteutetaan HTML:llä, ja tyylittely CSS:llä, joten näidenkin hallinta vaaditaan. Vuen dokumentaatio on hyvin kattava, ja sisältää esimerkit kaikille olennaisimmille ominaisuuksille. (Introduction | Vue.js, 2020.)

4.3.2 Toiminnallisuus

Toiminnallisuuksiltaan Vue on Reactin kanssa verrattavissa. Vue keskittyy ainoastaan MVC-mallin View-osion toteutukseen, joten rakentaakseen täysiä sovelluksia Vuella kehittäjän on otettava käyttöön myös muita kirjastoja. Toisin kuin React, Vue tarjoaa kuitenkin virallisina lisäkirjastoina reitittämiseen tarkoitetun kirjaston Vue Router (Vue Router, 2020) sekä edistyneeseen sovelluksen tilanhallintaan (*state management*) virallisen kirjaston Vuex (Vuex, 2020).

Mainittavimpina eroina toiminnallisuuksissa on, että Vue käyttää Angularin tavoin laajennettua syntaksia komponenttien sapluunoissaan (template), esimerkiksi *v-for,* jota voidaan käyttää, kun halutaan hahmontaa tietty elementti jokaista määritettyä taulukon elementtiä kohden. Näitä kutsutaan Vuessa direktiiveiksi, kuten Angularissakin. Vue tarjoaa kuitenkin myös lyhenteitä useimmin käytettyihin direktiiveihin. Toinen merkittävä ominaisuus on, että toisin kuin React, Vue tukee Angularin tavoin kahdensuuntaista datan sitomista (*two-way data binding*).

Vue tarjoaa myös kehitystyötä auttavia työkaluja, kuten komentorivityökalut projektin perustamiseen ja koontiin, testityökalut, sekä kehitystyökaluliitännäisen selaimeen. Vue:a on mahdollista laajentaa kolmannen

osapuolen kirjastoilla kuten Reactia ja Angulariakin, ja NPM-paketteja on ladattavissa noin yhtä paljon kuin Angularillekin (ks. Taulukko 1).

4.4 Yhteisöt ja ekosysteemit

Tässä työssä yhteisöön liittyviä tekijöitä tarkastellaan mittareilla, jotka ovat tasavertaiset ja kaikkiin kolmeen viitekehykseen sovellettavissa. Näitä ovat saatavilla olevien viitekehykseen liittyvien NPM-pakettien määrä, viitekehyksen Github-repositorion tähdet, sekä viitekehyksen Github-repositorion avoimet ongelmat (issue) suhteessa ratkaistuihin. Näistä NPM-pakettien määrä mittaa yhteisön tuottamia resursseja, Github-tähdet yhteisön kokoa ja viitekehyksen suosiota, ja avoimien ongelmien suhde suljettuihin yhteisön vastauskykyä (pienempi suhdeluku tarkoittaa parempaa vastauskykyä). Viitekehyksiä verratessa saatiin seuraavat tulokset (Taulukko 1):

	React	**Angular**	**Vue**
NPM-pakettien määrä	146535 kpl	45995 kpl	42608 kpl
Github-tähdet	160000 kpl	68200 kpl	176000 kpl
Ongelmat (avoin/suljettu) => suhde	505/9303 => **0,05**	2486/19366 => **0,13**	337/8920 => **0,04**

Taulukko 1 Viitekehyksien yhteisöt

Taulukkoa tarkastellessa voidaan huomata Angularin yhteisön koon ja suosion sekä avoimien ongelmien määrän olevan selvästi huonommassa suhteessa keskenään verrattuna kahteen muuhun vertailtuun viitekehykseen. Reactiin ladattavien tukikirjastojen ja liitännäisten määrä NPM-paketeissa mitattuna taas on huomattavasti suurempi kuin muilla vertailluilla viitekehyksillä. Vue taas on Github-tähtien perusteella suosituin vertailluista viitekehyksistä, tai sillä on suurin yhteisö. Myös avoimien ongelmien suhde on kaikista pienin.

4.5 Tehokkuus

Viitekehyksen tehokkuutta on tässä työssä mitattu Stefan Krausen JavaScript-viitekehysten mittaustyökalulla (https://stefankrause.net/js-frameworks-benchmark8/table.html) jossa vertailuun on valittu viimeisimmät versiot Angularista, Reactista, ja Vuesta. Tässä tutkielmassa tuloksista on huomioitu käynnistystä mittaavat mittarit, joita työkalu on mitannut Googlen Lighthouse-työkalulla, simuloiden sivun ajamista mobiililaitteella. Nämä mittarit ovat:

- Yhdenmukaisesti interaktiivinen (consistently interactive), pisin tarvittu aika siihen, että sivu on interaktiivinen, ja suoritin ja verkkoliikenne ovat joutilaana

- Skriptien käynnistysaika (script bootup time) Kaikkien sivuston skriptien jäsentämiseen, koontiin ja evaluointiin tarvittu aika millisekunteina
- Pääsäikeen työmäärä (main thread work cost) prosessin pääsäikeen suorittamiseen käytetty aika, sisältäen tyylit, asettelun yms.
- Ladatut tavut (total byte weight) verkon yli ladattujen resurssien koko tavuina, pakattuna.

Tulokset ovat nähtävillä seuraavassa kuviossa (Kuvio 4).

Käynnistyksen mittaustulokset (Google Lighthouse mobiilisimulaatiolla)

Viitekehys	angular-v6.1.0-keyed	react-v16.4.1-keyed	vue-v2.5.16-keyed
Yhdenmukaisesti interaktiivinen (consistently interactive), pisin tarvittu aika siihen, että sivu on interaktiivinen, ja suoritin ja verkkoliikenne ovat joutilaana	3,278.5 ± 4.0 (1.5)	2,477.6 ± 0.6 (1.1)	2,252.7 ± 0.2 (1.0)
Skriptien käynnistysaika (script bootup time) Kaikkien sivuston skriptien jäsentämiseen, koontiin ja evaluointiin tarvittu aika millisekunteina	235.2 ± 8.5 (4.3)	65.6 ± 2.3 (1.2)	55.1 ± 1.5 (1.0)
Pääsäikeen työmäärä (main thread work cost) prosessin pääsäikeen suorittamiseen käytetty aika, sisältäen tyylit, asettelun yms.	679.3 ± 5.3 (1.6)	466.0 ± 3.9 (1.1)	420.6 ± 67.5 (1.0)
Ladatut tavut (total byte weight) verkon yli ladattujen resurssien koko tavuina, pakattuna.	365,497.0 ± 0.0 (1.7)	251,915.0 ± 0.0 (1.2)	215,445.0 ± 0.0 (1.0)

Kuvio 4 Mitattujen suorituskykyyn liittyvien arvojen vertailu

Seuraavaksi käydään läpi tärkeimpiä johtopäätöksiä, joita tuloksia tarkastelemalla voidaan tehdä. Taulukosta voidaan huomata, että Angularilla kuluu skriptien käynnistämiseen yli nelinkertainen aika verrattuna Vueen. React häviää tässä Vuelle vain hyvin niukasti, noin kymmenellä millisekunnilla. Myös pääsäikeen suorittamiseen kulunut aika, ja ladattujen resurssien määrä ovat Angularin tapauksessa huomattavasti suuremmat. Tämä selittynee sillä, että Angular on viitekehyksenä suurempi, ja se tarjoaa eniten toiminnallisuuksia. Testattujen sovellusten ollessa kuitenkin ominaisuuksiltaan täysin vastaavat, voidaan todeta, että Angular ei sovellu pieniin projekteihin, johtuen kaikesta ylimääräisestä koodista, jota joudutaan lataamaan sitä ajettaessa. Testistä ei käy ilmi, ovatko tulokset vastaavat suurikokoisemmissa projekteissa, joissa Reactin ja Vuen yhteydessä ladataan myös suurempi määrä tukikirjastoja.

5 YHTEENVETO

Tässä tutkielmassa on tarkasteltu ja vertailtu kirjallisuutta, joka käsittelee JavaScript-viitekehyksien valintapäätökseen johtavia tekijöitä, ja johdettu tarkastelun perusteella malli, jolla verrataan tässä työssä vertailtavia viitekehyksiä. Tällä vertailulla pyrittiin selvittämään A) mitkä tekijät vaikuttavat web-viitekehyksen valintaan? Ja B) Kuinka kolme käytetyintä viitekehystä eroavat toisistaan tällä mittapuulla? Onko joku näistä kolmesta selvästi paras valinta johonkin tiettyyn käyttötarkoitukseen? Tässä kappaleessa vedetään yhteen tehtyjä johtopäätöksiä, tarkastellaan tutkimusmenetelmän puutteita, ja esitellään mahdollisia jatkotutkimusaiheita.

Kysymykseen siitä, mitkä tekijät vaikuttavat web-viitekehyksen valintaan, on vastattu luvussa 3. *Satromin* (2017) ja *Graziotinin ym.* (2016) malleja tarkastelemalla saatiin kattava käsitys siitä, että web-viitekehyksen valintaan vaikuttaa suuri määrä tekijöitä, ja usein tämä valinta riippuu projektin vaatimuksista. Tässä työssä valikoin kriteerien joukosta sellaiset, jotka mahdollisimman hyvin kuvaavat viitekehysten eroja, ollen samalla kuitenkin tutkittavissa ja mitattavissa ilman empiiristä tutkimusta.

Toiseen tutkimuskysymykseen, eli kuinka kolme käytetyintä viitekehystä eroavat toisistaan, ja onko jokin näistä kolmesta selkeästi parempi valinta johonkin tiettyyn käyttötarkoitukseen, etsittiin vastausta luvussa 4. Vertailtavia viitekehyksiä, Reactia, Angularia ja Vue:ta tarkasteltiin yleisellä tasolla, sekä eriteltiin niiden opittavuutta ja toiminnallisuuksia. Tämän jälkeen niiden ympärillä olevia yhteisöjä ja ekosysteemejä mitattiin, saaden Taulukon 1 mukaiset tulokset. Lisäksi viitekehysten suorituskykyä mitattiin eräällä vertailutyökalulla.

Verrattaessa viitekehysten yhteisöjä ja ekosysteemejä voitiin huomata, että Reactin ekosysteemi on NPM-pakettien tarjonnan osalta selvästi kaikkein suurin, ja Reactin Github-repositoriolla on näistä kolmesta viitekehyksestä kaikkein eniten tähtiä. Tähtien määrä kertoo kehittäjien mielenkiinnosta viitekehystä kohtaan. Yhteisön vastauskykyä pyrittiin mittaamaan avoimien ongelmien suhteella suljettuihin, ja tällä mittarilla Vue vei voiton, Angularin suhdeluvun ollessa kaikista suurin (ks. Taulukko 1). Tämä lukema kertoo kuitenkin vain ongelmista ja virheistä viitekehyksessä, eikä niinkään siitä, kuinka yhteisö auttaa siinä tapauksessa, kun kehittäjällä on ongelma saada jokin asia tehdyksi omien taitojensa puutteesta johtuen. Tätä olisi voitu mitata paremmin tarkastelemalla esimerkiksi Stack Overflow-forumille lähetettyjen kysymyksien määrällä

Opittavuuden puolesta helpoimmaksi osoittautui Vue. Vuen käyttämä syntaksi on kaikista lähimpänä perinteistä JavaScriptin, HTML:n ja CSS:n muodostamaa kokonaisuutta, ja komponentit rakentuvat näistä kolmesta palasesta yhteen tiedostoon. Aloitteleva kehittäjä pääsee todennäköisimmin nopeimmin kaikista pisimmälle tällä viitekehyksellä. Vaikein omaksuttava on selkeästi monoliittinen ja laaja Angular, joka tosin maksaa jyrkän oppimiskäyränsä takaisin kattavilla ominaisuuksilla. Kattavammat ja

luotettavammat tulokset viitekehysten opittavuudesta olisi voitu saada toteuttamalla kyselytutkimus, sillä nyt tieto siitä, onko viitekehys helppo tai vaikea oppia, perustuu täysin lähteisiin, joissa usein lähteenä oli viitekehyksen omilla sivuilla määritelty arvio viitekehyksen vaikeustasosta. Tämä tieto ei varmastikaan ole valheellista tai väärää, mutta ei myöskään tarkasti mitattavaa eikä täten luotettavasti verrattavissa.

Suorituskykyä mitattaessa suurimmat puutteet suorituskyvyssä havaittiin olevan Angularissa. Angularin toiminta oli kauttaaltaan Reactia ja Vue:a huomattaviasti hitaampaa. Huomattavaa on kuitenkin, että kyseinen testi on suoritettu yhdellä, kaikilla viitekehyksillä vastaavanlaisilla projekteilla, eikä se täten anna kattavaa kuvaa siitä, vaikuttaako esimerkiksi projektin laajuus suorituskykyyn missä määrin. Itse viitekehyksen koko sen sijaan on Angularin tapauksessa kaikista suurin, vaikka toisaalta Vue ja React vaativat saman toiminnallisuuden saavuttaakseen tietyn määrän lisäkirjastoja, jotka kasvattavat ladattavan paketin kokoa. Pienissä, yksinkertaisissa sovelluksissa tämä ei tietenkään pidä paikkaansa.

Toiminnallisuuksien osalta Angular tarjoaa kattavimmat ominaisuudet ilman lisäosia. Tästä on se etu, että Angularilla projektia kehittäessä kehittäjällä on käytössään tietty pino (stack) teknologioita, eikä tarvitse puntaroida eri vaihtoehtojen välillä. Tästä on myös se hyöty, että aloittaessaan työskentelyn Angular-projektisssa uutena työntekijänä, kokenut Angular-kehittäjä pääsee helposti kärryille siitä, kuinka koodi toimii, sillä kaikki Angular-projektit ovat rakenteeltaan hyvin samanlaisia. Tämän varjopuolena on kuitenkin osaltaan vaihtoehtojen puute, eikä valinnanvaraa ole, toisin kuin Vuen ja Reactin tapauksessa. Vaikka React ja Vue tarjoavat toiminnallisuudet vain näkymien toteuttamiseen, kehittäjällä on vapaat kädet laajentaa viitekehystä haluamillaan kirjastoilla kaikille projektin osa-alueille. Tämä edellyttää kuitenkin kokemusta, ja yhteensopivuusongelmat ovat myös mahdollisia. Huomattakoon myös, että Vue tarjoaa virallisina lisäkirjastoina aiemmin esitellyt Vuex:n ja Vue Routerin, ja React tarjoaa vastaavat React Reduxin ja React Routerin kolmannen osapuolen kirjastoina, ja näiden avulla voidaan saavuttaa huomattava lisäys toiminnallisuuteen ilman yhteensopivuusongelmia. On siis hyvin tilannekohtaista, kehittäjästä ja käyttötarkoituksesta riippuvaa, mikä näistä kolmesta on paras valinta toiminnallisuuksien suhteen.

Näiden havaintojen perusteella voidaan pyrkiä vastaamaan toisen tutkimuskysymyksen toiseen osaan, eli onko jokin näistä viitekehyksistä selvästi muita parempi johonkin tiettyyn käyttötarkoitukseen. Pienten, prototyypinomaisten sovellusten, tai vaikkapa kotisivujen tapauksessa React tai Vue ovat selvästi parempi ratkaisu kuin Angular. Näitä käyttäen voidaan karsia kaikki epäolennainen pois, ja saavuttaa pienin mahdollinen pakettikoko ja paras suorituskyky. Laajemmissa järjestelmissä, joissa tarvittujen toiminnallisuuksien määrä on suurempi, ja sivusto koostuu useammista sivuista, on Angular todennäköisesti luotettavampi vaihtoehto. Yhteensopivat koko projektin kattavat rajapinnat, sekä TypeScriptin tarjoama vahva tyypitys helpottavat ylläpidettävyyttä laajoissa projekteissa.

Tutkimuksesta voidaan johtaa joitain käytännön implikaatioita viitekehysten soveltuvuudesta. Vertailusta voidaan tulla johtopäätökseen, että viitekehysten joukosta ei löydy yhtä viitekehystä, joka olisi paras valinta jokaisessa käyttötarkoituksessa. Jokaisella viitekehyksellä on kuitenkin omat vahvuutensa, ja valintapäätöstä tehtäessä on pystyttävä erottamaan näiden joukosta ne seikat, joilla on kunkin projektin yhteydessä eniten merkitystä.

React on viitekehyksistä suosituin, ja sillä on laajin ekosysteemi, sekä Vuen kanssa lähes yhtä suuri määrä Github-tähtiä. Tästä johtuen kehittäjiä, joilla on kokemusta Reactista, on todennäköisesti eniten, ja osaavia kehittäjiä on helpointa löytää. Laajan yhteisön etu on myös korkea yhteisön vastauskyky, ja täten myös viitekehyksen toimivuus nopeasta viitekehyksen ongelmien ja virheiden löytymisestä ja korjaamisesta seuraten. React on myös helposti myös vähemmän ohjelmointikokemusta omaavan ymmärrettävissä, joka tehostaa ohjelmistoprojekteissa esimerkiksi kehittäjien ja käyttöliittymäsuunnittelijoiden yhteistyötä. Suorituskyvyltään React on myös erittäin kilpailukykyinen. React on siis hyvä valinta sekä laajoihin kaupallisiin ohjelmistoprojekteihin toiminnallisen varmuutensa ja ominaisuuksiensa puolesta, että harrasteprojekteihin opittavuutensa puolesta.

Angular puolestaan ei ole yhtä suosittu kuin React, ja sen yhteisö ja ekosysteemi ovat huomattavasti pienemmät. Kokeneita Angular-kehittäjiä voidaan siis olettaa olevan vähemmän, kuin React-kehittäjiä. Vaikka Angularin avoimien ongelmien suhde ratkaistuihin ongelmiin onkin muita vertailtuja viitekehyksiä suurempi, voidaan Angularia kuitenkin pitää edelleen vakaatoimintaisena viitekehyksenä. Angularin toiminnallisuuksien ollessa kattavimmat, ja lisäkirjastojen tarve ollessa pienin, yhteensopivuusongelmat ovat myös oletettavasti kaikista epätodennäköisimpiä. Angular soveltuu siis erinomaisesti laajoihin kaupallisiin ohjelmistoprojekteihin, mutta jyrkän oppimiskäyrän ja raskaan pakettikokonsa vuoksi se ei ole suorituskykyisin valinta pieniin harrasteprojekteihin.

Vue on kolmesta vertaillusta viitekehyksestä kaikista tuorein, mutta sillä on siitä huolimatta eniten Github-tähtiä, josta voidaan päätellä sen olevan kehittäjien keskuudessa suosittu. Vue tarjoaa myös ydintoiminnallisuuden laajentamiseen virallisesti tuettuja lisäkirjastoja, joka poistaa näiden osalta yhteensopivuusongelmien riskin. Kolmannen osapuolen lisäkirjastoja on tarjolla saman verran kuin Angularin tapauksessa. Vuen avointen ongelmien suhde ratkaistuihin on kaikista pienin, joten voidaan olettaa sen olevan hyvin varmatoiminen viitekehys. Erittäin kokeneita Vuen osaajia on kuitenkin viitekehyksen suhteellisesta tuoreudesta johtuen todennäköisesti vähemmän kuin Reactin ja Angularin tapauksessa. Vue:a voidaan kuitenkin pitää soveltuvana laajoihin ohjelmistoprojekteihin, sekä opittavuutensa ja suorituskykynsä vuoksi erittäin hyvin soveltuvana myös harrasteprojekteihin.

Tutkimuksessa oli myös tutkimusmenetelmien rajallisuuden vuoksi tiettyjä puitteita. Suorituskykyä mitattiin vain yhdellä työkalulla, joka mittasi kaikkia viitekehyksiä yhdellä alustalla, ja testissä mitattiin vain yhden kokoista kokoonpanoa kullakin viitekehyksellä. Jotta voitaisiin verrata eri kokoisten

projektien suorituskykyä toteutettuna eri viitekehyksillä, tarvittaisiin aiheesta enemmän tutkimusta. Tämä tulisi toteuttaa mitaten suorituskykyä pienessä, keskisuuressa, ja suuressa projektissa, ja testata tulokset eri kohdealustoilla. Lisäksi jotkin vertausmallin määreistä, kuten opittavuus, ovat suhteellisia ja yksilöittäin vaihtelevia. Niistä saataisiin luotettavampi tulos suorittamalla esimerkiksi kyselytutkimus, jossa kyselyyn vastanneet jaoteltaisiin ohjelmointikokemuksen perusteella.

LÄHTEET

Abdalkareem, Rabe. 2017. "Reasons and Drawbacks of Using Trivial Npm Packages: The Developers' Perspective." In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, , 1062–64. https://doi.org/10.1145/3106237.3121278 (October 2, 2020).

Aggarwal, Sanchit. 2018. 5 International Journal of Recent Research Aspects *Modern Web-Development Using ReactJS*.

"Angular." https://angular.io/guide/setup-local (November 27, 2020).

"Angular - FEATURES & BENEFITS." https://angular.io/features (November 27, 2020).

"Angular Documentation." https://angular.io/docs (November 27, 2020).

"Create a New React App – React." https://reactjs.org/docs/create-a-new-react-app.html (November 27, 2020).

Curie, Dasari Hermitha, Joyce Jaison, Jyoti Yadav, and J. Rex Fiona. 2019. "Analysis on Web Frameworks." In *Journal of Physics: Conference Series*, Institute of Physics Publishing, 12114. https://iopscience.iop.org/article/10.1088/1742-6596/1362/1/012114 (September 8, 2020).

Graziotin, Daniel, Pekka Abrahamsson, and Amantia Pano. 2016. *What Leads Developers towards the Choice of a JavaScript Framework? Rationale Leading to the Adoption of a JavaScript Framework*. https://www.researchgate.net/publication/303221742 (November 24, 2020).

"Introduction | Vue.Js." https://v3.vuejs.org/guide/introduction.html#what-is-vue-js (November 27, 2020).

Ivanova, Slavina, and Georgi Georgiev. 2019. "Using Modern Web Frameworks When Developing an Education Application: A Practical Approach." In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2019 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 1485–91.

Neoteric. 2016. "Single-Page Application vs. Multiple-Page Application | by Neoteric | Medium." https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58 (November 3, 2020).

"React." https://reactjs.org/docs/getting-started.html (November 24, 2020).

"React Redux." https://react-redux.js.org/ (November 27, 2020).

"ReactTraining/React-Router: Declarative Routing for React." https://github.com/ReactTraining/react-router (November 27, 2020).

Richard-Foy, Julien, Olivier Barais, and Jean-Marc Jézéquel. 2014. "Efficient High-Level Abstractions for Web Programming." *ACM SIGPLAN Notices* 49(13). http://dx.doi.org/10.1145/2517208.2517227 (September 23, 2020).

Satrom, Brandon. 2017. *Choosing the Right JavaScript Framework for Your Next Web Application*.

Scott, Emmit A. 2016. *SPA Design and Architecture : Understanding Single-Page Web Applications*. MANNING.

"Server Render | Jamstack." https://jamstack.org/glossary/server-render/ (December 9, 2020).

Shahzad, Farrukh. 2017. "Modern and Responsive Mobile-Enabled Web Applications." In *Procedia Computer Science*, Elsevier B.V., 410–15.

"Stack Overflow Developer Survey 2020." https://insights.stackoverflow.com/survey/2020#technology-web-frameworks-all-respondents2 (September 23, 2020).

"Vue.Js Documentation." https://vuejs.org/v2/guide/ (November 23, 2020).

"Vue Router." https://next.router.vuejs.org/ (November 27, 2020).

"Vuex." https://next.vuex.vuejs.org/ (November 27, 2020).

"What Is a Static Site Generator? How Do I Find the Best One to Use?" https://www.netlify.com/blog/2020/04/14/what-is-a-static-site-generator-and-3-ways-to-find-the-best-one/?utm_source=jamstackorg&utm_medium=what-are-ssg-pnh&utm_campaign=devex (December 9, 2020).