Katsaus web-sovelluskehysten arviointimetodeihin

Turun yliopisto
Tietotekniikan laitos
LuK-tutkielma
Tietojenkäsittelytiede
Toukokuu 2024
Emil Hellberg

TURUN YLIOPISTO
Tietotekniikan laitos

Emil Hellberg: Katsaus web-sovelluskehysten arviointimetodeihin

LuK-tutkielma, 22 s.
Tietojenkäsittelytiede
Toukokuu 2024

Web-sovelluskehykset ovat tämän päivän web-sovelluskehityksessä sekä ohjelmistotuotannossa korvaamattoman tärkeitä kehittäjätyökaluja. Web-sovelluskehysten nopea kehitys 2010- sekä 2020-luvuilla onkin synnyttänyt tarpeen niiden kehittäjälähtöiselle arvioinnille ja vertailulle.
Tämä tutkielma on kirjallisuuskatsaus web-sovelluskehyksien arviointia käsitteleviin tutkimusaineistoihin, jossa selvitetään miten ja mistä näkökulmasta lähtien web-sovelluskehysten suoriutumista ohjelmistotuotannossa ja sovelluskehityksessä on lähdetty mallintamaan, sekä millaisia arviointimetodeja ja arviointien tuloksia kirjallisuudessa on muodostunut.
Tutkielman tuloksista selviää, että web-sovelluskehyksien mallintamiseen vaikuttaisi olevan kolme päätulokulmaa: suosion mallintaminen, suorituskyvyn mallintaminen sekä ohjelmistotuotannollisen ja sovelluskehityksellisen käytön mallintaminen. Tuloksista selviää myös, minkälaisia arviointimetodeja kirjallisuudessa on kehittynyt, sekä millaisia tuloksia näillä arviontimetodeilla on tutkimuksissa saavutettu, kuten esimerkiksi valmiiden sovellusten renderöintiaikojen vertailu, matemaattiset vertailumallit tai ohjelmistotuotannollinen kvalitatiivinen vertailu.

Asiasanat: web-sovellus, arvionti, arviointimetodi

Sisällys

1 Johdanto

Web-sovelluskehitys on eräs alati muuttuvimmista ohjelmistokehityksen alalajeista, sekä riippuvuuksien kannalta myös äärimmäisen monimutkainen. Web-sovelluskehykset on kehitetty yksinkertaistamaan, helpottamaan ja tehostamaan modernia web-sovelluskehitystä. Frontend-web-sovelluskehysten tarkoituksena on mahdollistaa lukuisten ohjelmistotuotannossa ja sovelluskehityksessä pitkään käytössä olleiden periaatteiden soveltaminen JavaScript-perusteisen web-sovelluksen toteuttamiseen. Tällaisia kehitysperiaatteita ovat muun muassa kehityksen aikainen komponenttihierarkia sekä näkymien, mallien ja käsittelijöiden erottaminen toisistaan ohjelmakoodissa.

Jatkuvan muutoksen sekä uusien tekniikoiden lomassa on web-sovelluskehittäjälle olennaista kyetä arvioimaan sekä vertailemaan erilaisia sovelluskehyksiä keskenään. Sovelluskehyksen valinta vaikuttaa sekä kehittäjän omaan tuottavuuteen ja ajankäyttöön, että tuotantoon päätyvän koodin toimivuuteen [1]. Tässä tutkielmassa selvitetään, millaisia web-sovelluskehysten arviointimetodeja on kirjallisuudessa tutkittu, frontend-web-sovelluskehysten suhteen. Tämän perusteella esitetään seuraavat tutkimuskysymykset:

TK1: Millaisilla lähestymistavoilla web-sovelluskehysten suoriutumista sovelluskehityksessä sekä ohjelmistotuotannossa on lähdetty mallintamaan?

TK2: Mitä arviointimetodeja kehysten arvioinneissa on käytetty?

TK3: Millaisia tuloksia arvioinneissa on saavutettu?

Aineistohaku aloitettiin Web of Science -tietokannasta, hakulausekkeilla ”JavaScript frameworks comparison” sekä ”front end framework comparison”, haun perusteella löytyi kaksi sopivaa aineistoa, molemmat IEEE:n julkaisuja. Tämän jälkeen hakua laajennettiin Google Scholarilla, samoilla hakulausekkeilla. Lopuksi haettiin vielä Google Scholarista sekä IEEE:n tietokannasta hakulausekkeilla ”frontend framework evaluation” sekä ”frontend framework comparison”.

Johdantoluvun jälkeen luvussa kaksi taustoitetaan web-sovelluskehitykseen liittyvää teoriaa ja tekniikoita sekä taustoitetaan sitä, mikä on lopulta johtanut web-sovelluskehysten sekä frontend-web-sovelluskehysten syntyyn. Luvussa kolme tehdään kirjallisuuskatsaus arviointimetodeista kolmen web-sovelluskehitykseen liittyvän pääteman kautta: kehysten suosio, kehysten suorituskyky ja valmiiden sovellusten tiedostokoko sekä viimeisenä sovelluskehitys ja ohjelmistotuotanto. Tämän lisäksi kolmannen luvun neljännessä aliluvussa koostetaan saadut havainnot yhteen taulukkoon ja esitetään niistä pohdintaa. Viimeinen eli neljäs luku sisältää yhteenvedon tutkielmasta.

2 Tausta

2.1 Web sovelluskehitysalustana

Web-sovelluksella tarkoitetaan verkon kautta kommunikoivaan HTTP-protokollaan peustuvan hypermediaverkon avulla toteutettuja hajautettuja sovelluksia. HTTP-protokollaa käytetään hypertekstidokumenttien eli HTML-tiedostojen lähettämiseen palvelimilta selaimille. Protokolla luotiin alun perin vain HTML-dokumenttien siirtoa varten, mutta webin kehittyessä sitä alettiin myös käyttää muiden sovellusten tarvitsemien resurssien, kuten kuvien, tyylitiedostojen ja JavaScript-koodin siirtämiseen. Web-sovellus noudattaa karkeasti jaoteltuna kolmitasoista arkkitehtuuria, johon kuuluvat frontend, jolla viitataan selainpuolen sovelluksiin, backend, johon sisältyy palvelimien toteutus, sekä tietokantaan [2].

Olennaisena erona backendin ja frontendin välillä on, että frontend-sovellukset suoritetaan aina paikallisesti käyttäjän omalla koneella selaimen suoritusympäristössä, kun taas backend-sovellukset ovat jatkuvasti käynnissä palvelinkoneessa. Frontend-sovellusten koodi siis ladataan ja käynnistetään uudestaan aina kun sivusto avataan, tietyin poikkeuksin. Selaimiin on lisätty tiettyjä toiminnallisuuksia sovelluksien käyttämien tietojen pidempiaikaista varastointia varten, kuten evästeet ja IndexedDB -ohjelmointirajapinta. Tässä tutkielmassa keskitytään frontend-web-sovelluskehityksessä käytettäviin sovelluskehyksiin.

Frontend-sovelluskehitys voidaan pohjimmiltaan jakaa kolmen pääkielen tai tek-

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <nav id="site-navigation">
        <ul>
            <li><a href="#about-me">About Me</a></li>
            <li><a href="#contact-me">Contact Me</a></li>
        </ul>
    </nav>
    <main id="main-content">
        <section id="about-me">
            <h1>About Me</h1>
            <p>My name is Zac. I like code and coffee!</p>
        </section>
        <section id="contact-me">
            <h1>Contact Me</h1>
            <form action="/thank-you" method="POST"
            ↪ name="contact-form">
                <label for="name">Your Name:</label>
                <input type="text" id="name" name="user-name"
                ↪ required>

                <label for="email">Your Email:</label>
                <input type="email" id="email" name="user-email"
                ↪ required>

                <input type="submit" value="Contact Me">
            </form>
        </section>
    </main>
</body>
</html>
```

Ohjelmalistaus 1: Esimerkki HTML-tiedoston sisällöstä

niikan hallitsemiseen: HTML, CSS sekä JavaScript. HTML, lyhenne sanoista Hypertext Markup Language, on web-sivujen rakenteen merkintäkieli, jota voi ajatella sivuston eli sovelluksen selkärankana. Listauksessa 1 esitetään HTML-tiedoston sisältämä määrittely yksinkertaiselle web-sovelluksen komponenttirakenteelle: juurikomponentti `<html>` sisältää `<head>-` sekä `<body>` -komponentit joihin jakaantuvat sivuston metatiedot ja varsinainen sisältö, kuten ohjelmalistauksen 1 rivin 14 elementti `<h1>`, joka määrittelee ensimmäisen tason otsikon web-sivulle.

HTML-tekstidokumentteihin voidaan kiinnittää CSS-tyylejä (Cascading Style Sheets), joilla voidaan hallita muun muassa sovelluksen värejä, visuaalisten elementtien reunuksien tyylejä tai esimerkiksi animoida näkymien vaihdoksia. Sovelluksen toimintalogiikka ja kaikki suoritettava koodi taas toteutetaan JavaScriptillä, joka liitetään dokumenttiin script-elementeillä. Tämä mahdollistaa interaktiivisuuden sekä mukautettujen toiminnallisuuksien lisäämisen web-sovellukseen [3]. CSS:n ja JavaScriptin liittävien tagien sisään on merkitty niiden suhteellinen tiedostopolku palvelimen hakemistossa, josta valmis web-sovellus tarjotaan. CSS- ja JS-koodit voivat sijaita omissa tiedostoissaan. Listauksessa 2 on havainnollistettu, miten erillisessä tiedostossa (myscripts.js) sijaitseva JavaScript-koodi liitetään osaksi HTML-dokumenttia sekä miten erillisessä tiedostossa (index.css) olevat css-tyylit lisätään HTML-dokumenttiin.

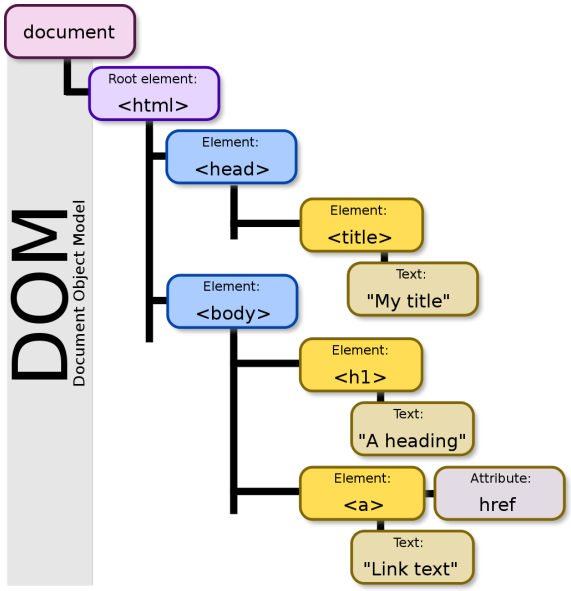
DOM (Document Object Model) on selaimen HTML-dokumentista muodostama tietorakenne, jonka muuttamiseen selain tarjoaa web-sovellukselle JavaScript-ohjelmointirajapinnan, joka mahdollistaa sivuston rakenteen manipuloinnin dynaamisesti sovelluksen suorituksen aikana, eli kun sivusto on käyttäjällä näkyvissä. DOM-tietorakenteen manipuloinnin JavaScript-koodista mahdollistaa document-rajapinta, jonka metodeja kutsumalla voidaan hakea, muokata, lisätä tai tietyissä tilanteissa poistaa elementtejä DOM-puusta web-sovelluksen suorituksen aikana. Esimerkiksi `getElementById()`-metodin avulla saadaan viitattua JavaScript-koodista

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <link rel="stylesheet" type="text/css"
       ↪  href="./bootstrap/bootstrap.min.css">
      <link href="./resources/css/index.css" type="text/css"
       ↪  rel="stylesheet">
      <script src="./src/myscripts.js"></script>
      <title></title>
   </head>
   <body>
       ...
   </body>
</html>
```

Ohjelmalistaus 2: Esimerkki CSS- ja script -tägeistä

käsin DOM-puun elementtiin elementin yksilöivän id:n avulla, tai `innerHTML`-attribuutti, jonka avulla päästään käsiksi HTML-elementin sisältävään HTML-koodiin. Yksittäisillä HTML-elementeillä on joko yksilöivä tunniste, `id`, tai luokkatunniste `class`, joiden avulla niihin voidaan viitata. Koko HTML-tiedosto, joka on siis objektipuun juurielementti, voidaan sitoa muuttujaan `document`-olion `getElementById`-metodilla antamalla sille argumentiksi `'html'`. Tämän jälkeen tätä viittausta voidaan käyttää esimerkiksi uusien HTML-elementtien renderöintiin tai vanhojen poistamiseen näkymästä.

Rajapinnan avulla voidaan myös käsitellä yksittäisiä elementtejä niiden id:n avulla ja muuttaa vaikka niiden väriä, suhteellista kokoa tai tekstisisältöä. Kuvassa 2.1 on havainnollistettu selaimen muodostamaa DOM-puuta. Esimerkiksi elementti `<a>` löytyisi kuvan mukaisesta DOM-puusta etenemällä document-oliosta juurielementtiin `<html>`, siitä sisällön sisältävään osaan eli `<body>`-elementtiin, josta edelleen edetään puussa alas kunnes löydetään haluttu elementti, joko vertaamalla lopulta lehtisolmun id:tä tai nimeä hakuparametriin tai palauttamalla kokoelma kaikista löydetyistä tietyn tyyppisistä, tai tiettyyn luokkaan kuuluvista elementeistä.



Birger Eriksson (CC BY-SA) Lähde: https://commons.wikimedia.org/wiki/File:DOM-model.svg

Kuva 2.1: Esimerkki selaimen muodostamasta DOM-puusta

2.2 MPA- sekä SPA-arkkitehtuurien erot

Perinteisissä MPA-web-sovelluksissa (Multi Page Application) jokainen sovelluksen näkymä ladataan erikseen palvelimelta HTML-muodossa, mikä lisää näkymien välillä siirtymiseen kuluvaa aikaa. MPA-arkkitehtuurissa selaimen vastuulla on ainoastaan renderöidä palvelimelta saatu HTML-tiedosto sekä lähettää pyynnöt uusista HTML-tiedostoista näkymien vaihdon yhteydessä. Palvelimen rooliksi jää tässä tapauksessa sovelluksen toimintalogiikka; palvelimen sovelluslogiikka määrää lähetettävät .HTML-, .CSS- sekä JavaScript-tiedostot ja niiden sisältämän datan sekä sisällön. Web-sovelluksen tila on tällöin käytön aikana palvelimella, ja MVC-arkkitehtuurin osien kannalta ainoastaan ”View” eli näkymä on selaimessa esitettyinä kokonaisuudessaan. Tässä kontekstissa näkymällä viitataan siis web-sivun rakenteeseen ja ulkoasuun eli HTML:ään ja CSS:ään, kun taas tietomallit ja tiedon käsittely sijaitsevat palvelinkoneella, josta tiedot haetaan ja syötetään käyttöliittymään eli selaimeen.

Taulukko 2.1: Selaimen ja palvelimen roolit eri arkkitehtuureissa

Ark.	Selain	Palvelin
SPA	• Näkymän tila voi muuttua ja komponentit päivittyä • JavaScriptistä käsin voidaan tehdä XHR-pyyntöjä (AJAX)	• Pidempiaikainen sovellustietojen säilytys • Tarjoaa näkymän tarvitsemat tiedot rajapintana
MPA	• Näkymän tila ei muutu, renderöidään palvelimen lähettämä kuvaus näkymästä • Muutokset näkymään vaativat koko websivun uudelleenlataamisen	• Sisältää pääosan sovelluksen tilasta, mukaanlukien näkymän • Tarjoaa jokaisen näkymän omana HTML-muodossa • Pidempiaikainen sovellustietojen säilytys

Asynchronous JavaScript and XML (AJAX) on web-sovellusten toimintaperiaate, joka mahdollistaa asynkronisten HTTP-pyyntöjen lähettämisen selaimessa ajettavasta JavaScript-sovelluksesta palvelimelle. Konsepti tuli virallisesti osaksi web-tekniikoita vuonna 2006, jolloin W3C-konsortio julkaisi ensimmäisen prototyypin XMLHttpRequest-oliosta, joka mahdollisti tietojen hakemisen palvelimelta ilman koko sivun uudelleen lataamista [4]. AJAX on sittemin vakiinnuttanut asemansa osana web-tekniikoita, ja tuki sille löytyy nykyään kaikista käytetyimmistä selaimista. Moderneilla SPA-arkkitehtuuria (Single Page Application) noudattavilla web-sovelluksilla on ominaista, että DOM:ia voidaan sovelluksen suorituksen aikana muuttaa ja AJAX-tekniikkaa käyttäen hakea yksittäisiä resursseja tarpeen mukaan palvelimelta, ilman että koko sivustoa tarvitsisi ladata uudestaan. Web-sovellusten toimintalogiikan painopisteen siirtyminen palvelimesta selaimeen onkin johtanut frontend-koodipohjan laajenemiseen sekä monimutkaistumiseen. Taulukossa 2.1 on koostettuna MPA- sekä SPA-sovellusarkkitehtuurien pääasialliset erot, niin asiakkaan kuin palvelimenkin puolelta tarkasteltuna.

2.3 Web-sovelluskehysten ilmaantuminen

Ohjelmoinnissa sekä ohjelmistotuotannossa käytettyjen sovelluskehysten ideana on tarjota valmiita projektihakemistopohjia, ohjelmointikirjastoja sekä yleisiä toimintaperiaatteita luotaville sovelluksille, eli luoda niin kutsuttu kehys sovelluskehitykselle. Sovelluskehysten perimmäisenä tarkoituksena on siis tukea sovelluskehitystä ja ohjelmointia tarjoamalla valmiita projektihakemistoja, rajapintoja, koodi- sekä testauskirjastoja sekä tuen koodin uudelleenkäyttöön. Sovelluskehysten tehtävä on insinööritieteellisestä näkökulmasta ohjauksen kääntäminen (engl. inversion of control), jossa sovelluksen kontrollitoiminnot siirtyvät sovelluskehyksen vastuulle ja varsinainen sovellusohjelmointi tarkoittaa valmiiden luokkien laajentamista ja tarvittavien riippuvuuksien lisäämistä. Ohjauksen kääntäminen tekee koodista modulaarisempaa, helpottaa yksikkötestausta sekä tehostaa tehtävän implementointia [5].

HTTP/2 sovellusprotokollastandardi (alun perin Googlen kehittämän SPDY-protokollan perusteella kehitetty) julkaistiin yleisölle vuonna 2015, jonka lisäksi REST-arkkitehtuurin yleistyminen on mahdollistanut sovelluskehittäjille dynaamisten, suorituskykyisten ja visuaalisesti näyttävien web-sovellusten tuottamisen. Webtekniikoiden yleinen kehitys viimeisen kolmen vuosikymmenen aikana on merkinnyt siirtymistä monoliittisista sekä MPA-arkkitehtuuriin pohjautuvista web-sovelluksista selainpainotteisiin ja SPA-arkkitehtuuriin perustuviin frontend-sovelluksiin, joissa enenevissä määrin pyritään toteuttamaan sovelluslogiikka sekä istuntoaikaisten tietojen tallennus selainpuolella, perustuen selaimen JavaScript-ympäristössä suoritettavaan web-sivun JavaScript-koodiin. Frontend-sovelluskehysten synty voidaankin nähdä seurauksena SPA-arkkitehtuurin yleistymiselle web-sovellustuotannossa; kehysten perimmäinen tehtävä on helpottaa monimutkaisen tilanhallinnan sisältävän sovelluksen toteutusta.

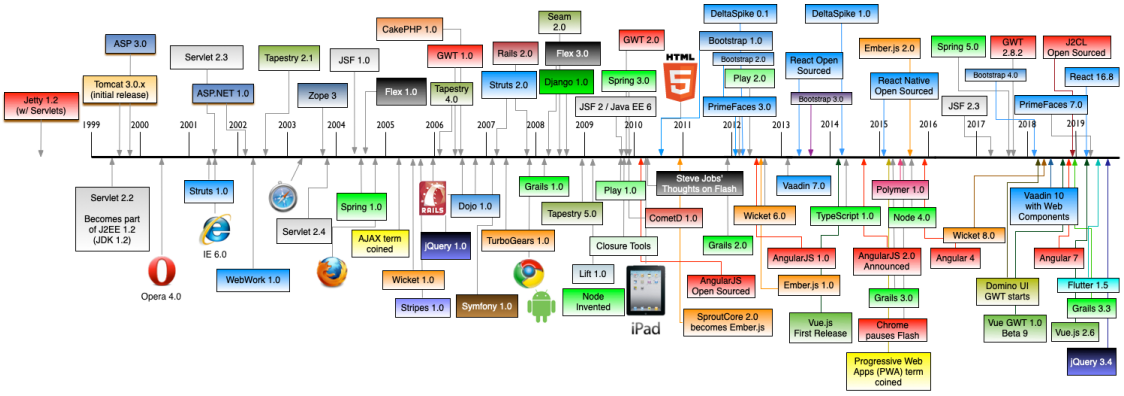
Frontend-sovelluskehys on kokoelma työkaluja ja teknologioita jotka tehostavat web-sivun käyttöliittymän luontiprosessia. Nämä kehykset antavat kehittäjille

valmiit rakennuspalikat ja toimintatavat, jotka mahdollistavat nopeatahtisen dynaamisten ja joustavien web-sivujen luonnin. Frontend-sovelluskehyksen suurin etu on sen säästämä aika kehityksen aikana, uhraamatta lopullisen tuotteen laatua. Frontend-kehykset kuten React, Angular, Vue sekä Ember ovat esimerkkejä suosituimmista vaihtoehdoista [1].

Frontend-kirjastot taas keskittyvät kokoamaan hyödyllisiä ominaisuuksia sisältävää koodia kehittäjiä varten. Liukusäätimet, valikkopalkit sekä modaali-ikkunat ovat esimerkkejä ominaisuuksista, joita voidaan implementoida näiden kirjastojen avulla. Frontend-kehyksen kanssa käytetty frontend-kirjasto voi tarjota ominaisuuksia joita ei muuten olisi valmiiksi saatavilla [1]. Frontend-sovelluskehyksen antamien ”raamien” avulla totetutettuun web-sovellukseen saadaan frontend-kirjastojen avulla hyödyllisiä ominaisuuksia, kuten esimerkiksi autentikoinnissa tarvittavien tokenien luontia varten tai valmiita testikirjastoja yksikkötestien kirjoittamiseen.

Modernin frontend-web-sovelluksen HTML-juuritiedosto sijaitsee yleensä projektihakemiston juuressa ja on ”tynkä”, johon varsinainen komponenttirakenne on määriteltynä, esimerkiksi Reactin tapauksessa, JSX-koodina erillisissä tiedostoissa. Jokaisella kehyksellä on oma formaattinsa koodin määrittelyyn, joista JSX on yksi yleinen merkintäkieli. JSX (JavaScript XML) on alun perin Facebookin Reactia varten kehittämä JavaScriptin laajennus, joka on luotu helpottamaan määrittelyjen kirjoittamista. JSX-koodi tarvitsee toimiakseen kääntäjän (engl. transpiler), joka kääntää lähdekoodin toisenkieliseksi lähdekoodiksi, frontend-kehysten tapauksessa JavaScriptiksi. JSX-laajennusta käytetään edelleen suosituimman frontend-kehyksen, Reactin, merkintään, mutta sen käyttöä koodin merkintään tukevat myös monet vähemmän tunntetut frontend-kehykset, kuten Preact tai Solid.

Ensimmäinen varsinainen frontend-sovelluskirjasto jQuery julkaistiin vuonna 2006, se mahdollisti JavaScriptin kirjoittamisen selainriippumattomasti. Suosituin tämänhetkinen frontend-sovelluskehys, React [6], julkaistiin Facebookin toimesta vuon-



Kuva 2.2: Aikajana täkeimpien web-teknologioiden julkaisuajankohdista

na 2014, ja se toi frontend-kehitykseen ideat komponenteista sekä MVC (Model, View, Controller) -arkkitehtuurista. Kuvassa 2.2 on havainnollistettu tärkeimpien frontend-kehitykseen liittyvien teknologioiden julkaisuajankohtia.

3 Kirjallisuuskatsaus

3.1 Suosio

Sovelluskehysten suosio on tärkeä vertailukohde, sillä se viestii suoraan siitä kuinka paljon kehystä käytetään ohjelmistotuotannossa. GitHub on käytännössä modernien web-sovelluskehysten de facto -julkaisufoorumi, joten GitHub:n tuottama tilastotieto julkisista koodin säilytyspaikoista (engl. ”repository”), on hyödyllistä kehysten vertailussa. Käyttäjät voivat antaa repositorioille tähden merkiksi hyvästä tai hyödyllisestä koodista. GitHubista selviää myös, monessako muussa projektissa on kyseistä sovelluskehystä käytetty. Artikkelissa ”Multi Attribute Decision Making Model for Ranking of Web Frameworks” (Borisova et al., 2021) [7] osana matemaattisen vertailumetodin syötettä toimi sovelluskehyksen suosio, tarkemmin kehyksen GitHub- sekä StackOverflow-pistemäärät. Molemmat pistemäärät ovat suoraan verrannollisia kehysten suosioon kehittäjien keskuudessa.

Stack Overflow on ohjelmointiin, ohjelmistoarkkitehtuuriin sekä tietoturvaan keskittyvä keskustelufoorumi verkossa. Käyttäjien pistemäärä perustuu muiden käyttäjien esittämien kysymysten vastaamiseen sekä annettuihin ääniin, ja kuvastaa käyttäjän ohjelmistokehitykseen liittyvän asiantuntemuksen tasoa. Sivusto myös ylläpitää vuosittaista Stack Overflow Developer Survey -kyselytutkimusta, jossa selvitetään kuluneen vuoden kehittäjien keskuudessa suosituimmat ohjelmistotekniikat.

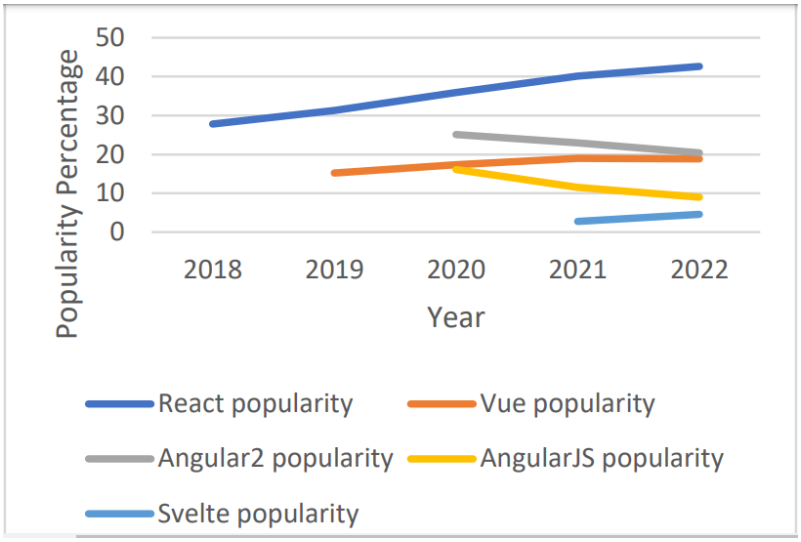
Näiden kahden avoimen tietolähteen lisäksi voidaan suosiota mitata myös sosi-

Taulukko 3.1: Neljän eri frontend-sovelluskehyksen ”läsnäolo” internetissä [8]

	Kehys			
Vertailukriteeri	AngularJS	Backbone	Ember	Knockout
Google-hakutulokset	$\sim$ 105 milj.	$\sim 45,5$ milj.	$\sim$ 6 milj.	$\sim 49,4$ milj.
StackOverflow-hakutulokset	252 317	20 505	22 507	19 316
GitHub projektit/koodi	380 084	24 191	29 912	5 693
Tähtien määrä GitHubissa	36 669	27 192	19 003	8 895
YouTube hakutulokset	$\sim$ 800 t.	$\sim$ 29 t.	$\sim$ 42 t.	$\sim$ 17 t.
Kehyksen koko [kB]	39,5	6,5	90	55
Koko riippuvuuksineen [kB]	39,5	43,5	132,2	55

aalisen median presenssillä, kuten YouTube-hakutulosten määrällä. Lähde [7] onkin ainoa käytetyistä aineistoista, jossa suosio toimii yhtenä syötteenä painotetulle summalle, muut lähteet keskittyvät vain raakadatan vertailuun. Mitään omaa tutkimusdatan keruuta frontend-kehysten suosion suhteen on luonnollisesti hyvin vaikea suorittaa, joten tutkimusdatan lähteet ovat käytännössä aina peräisin web-palveluista. Artikkelissa ”Modern JavaScript frameworks: A Survey Study” (Dalcev et al., 2018) [8] eriteltiin ensin kehysten ”läsnäoloa internetissä” eli Googlen hakutulosten määrä kehyksen nimellä, StackOverflow-hakutulosten määrä, GitHub-pistemäärä sekä YouTube-hakutulosten määrä. Taulukossa 3.1 on artikkelin suosion vertailun tulokset.

Artikkelissa ”Comparison and Analysis of Popular Frontend Frameworks and Libraries: An Evaluation of Parameters for Frontend Web Development” (Kaur, Tiwari, 2023) [1] ensimmäisenä vertailun kohteena toimi suosio, jonka tärkeyttä sovelluskehysten vertailussa perusteltiin sillä että suurempi suosio kertoo sen luotettavuudesta ja lisää internetistä löytyvien resurssien, kuten dokumentaation, tutoriaalien sekä tukifoorumien määrää. Mitä suositumpi käytetty teknologia on, sitä parempia projekteja ja tehokkaampia ohjelmistotuotannon ympäristöjä saadaan aikaan. Vertailun aineistona käytettiin Stack Overflow:n dataa vuosien 2018-2022



Kuva 3.1: Viiden suosituimman frontend-sovelluskehyksen suhteellinen suosio vuosina 2018-2022 [1]

suosituimmista frontend-kehyksistä. Kuvassa 3.1 on artikkelin suosion vertailun tulokset. Kuvaajista näkyy hyvin selkeästi frontend-web-sovelluskehityksessä valloilla ollut trendi, jossa lähestulkoon kaikki frontend-sovellukset toteutetaan yhdellä viidestä suosituimmasta kehyksestä (React, Angular, Vue, Svelte tai Angular2). Samalla uusia fronted-kehyksiä syntyy jatkuvalla tahdilla, mutta niiden käyttöönotto erityisesti ohjelmistoteollisuudessa saattaa lopulta jäädä epärelevantiksi. Kuvaajista myös näkyy Reactin suosion hidas mutta varma kasvu. Angular ja Angular2 taas ovat menettäneet tasaiseen tahtiin suosiota vuoden 2022 loppuun asti, kuvaajat eivät toki kerro tämän ja viime vuoden kehityksestä.

Aineistoissa pelkästään suosiota kriteerinä voidaan mitata ja vertailla monella eri tavalla. Voidaan käyttää vertailussa suoraan verkon ohjelmistotuotantoon liittyvien foorumien tarjoamia analytiikkapalveluja, kuten StackOverflow:ta tai GitHub:ia. Pelkän raakadatan vertailun eli suhteellisen osuuden mittauksen kaikista projekteista tai tähtien määrän tarkastelun lisäksi suosio voi olla yhtenä parametrina laajemmassa vertailumallissa. Tarkastellut aineistot vaikuttaisivatkin tässä mielessä jakaantuvan kahteen pääkategoriaan: suoraan raakadatan vertailuun sekä niihin,

joissa vertailua varten on kehitetty jonkinlainen tilastollinen malli, jonka syöteparametreina toimivat muun muassa suosioon liittyvä data sekä muut tärkeäksi todetut kriteerit.

3.2 Suorituskyky ja tiedostokoko

Suorituskyky ja valmiiden sovellusten ladattavien tiedostojen koko ovat olennaisia vertailukohteita, sillä ne pitkälti määrittävät web-sivun käyttökokemuksen. Web-sivu joka ei lataudu nopeasti tai jonka tiettyjen elementtien latautumiseen menee huomattavasti aikaa, jää myös esimerkiksi SEO-arvioinnissa muiden samankaltaista sisältöä tarjoavien web-sivujen kanssa kilpaillessaan jälkeen, eikä todennäköisesti houkuttele käyttäjiä muutenkaan. Artikkelissa ”Modern Web Frameworks: A Comparison of Rendering Performance” (Ollila et al., 2022) [9] pääasiallisena vertailumetodina oli web-sovellusten renderöintiaika, jota taas vertailtiin sovelluksen eri tiloissa, kuten sivuston renderöinti, komponentin lisäys tai juurikomponentin päivitys. Yksikkönä mittauksissa oli millisekunti. Vertailussa olivat frontend-kehykset Angular, React, Vue, Svelte sekä Blazor, joiden toiminnalle on ominaista virtuaalisen DOM:n ylläpito sovelluksen kehityksen ja suorituksen aikana. Eri vertailuskenaarioita olivat uusien komponenttien renderöinti, uusien komponenttien lisäys binääripuuna, juurikomponentin vaihto, lehtikomponentin vaihto sekä koko komponenttipuun vaihto.

S. Delcev ja D. Draskovic [8] jatkoivat suosion vertailun jälkeen sovelluskehysten suorituskyvyn ja tiedostokoon arvioinnilla. Sovelluskehyksistä arvioitiin sekä kehyksen omaa tiedostokokoa että sen tiedostokokoa kaikkien tarvittavien kirjastojen, eli riippuvuuksien, kanssa. Suorituskykyä arvioitiin suorittamalla jokaisella sovelluskehyksellä testi, joka tulostaa komentokehotteseen 500 kokonaislukuarvoa.

Artikkelissa ”A Comparative Analysis of Modern Frontend Frameworks for Building Large-Scale Web Applications” (Singh et al., 2023) [10] vertailtavien sovellus-

kehysten avulla tuotettujen web-sovellusten vertailukriteereitä olivat käyttäjän tyytyväisyys, ensimmäisen tavun siirtämiseen kuluva viive, ensimmäisen komponentin renderöintiin kuluva aika, sivun latautumiseen kuluva aika, interaktiivisuuden latautumiseen kuluva aika, muistin käyttö pyydettäessä, sovelluskehyksen käynnistykseen kuluva aika sekä eniten aikaa vievien palvelinpyyntöjen käsittelyyn kuluva aika. Vertailudata suorituskyvyn vertailuun saatiin tutkimukseen kolmesta eri analytiikkapalvelusta: JavaScript Framework Benchmark, PerfTrack sekä Startup Metric. Lähteessä [1] jatkettiin suosion vertailun perustelujen jälkeen suorituskyvyn vertailulla, jossa määriteltiin tärkeimmät frontend-sovelluskehysten suorituskykyyn vaikuttavat tekijät: Virtuaalisen DOM:n käyttö käyttöliittymien näkymien päivittämiseen ja uudelleenrenderöintiin, kyky käsitellä massiivisia datamääriä ja kyky toteuttaa monimutkaisia käyttöliittymäelementtejä. Suorituskyvyn vertailu lähteessä[1] oli täysin kvalitatiivista, vertaillen omissa kappaleissaan yksittäisiä frontend-kehyksiä, kuten React ja Angular, Angular ja Svelte, ja niin edelleen. Tässä kyseisessä vertailukohdassa kvalitatiivinen vertailu perustui ilmeisesti kirjoittajan omiin näkemyksiin asiasta.

Frontend-sovelluskehysten avulla tuotettujen valmiiden sovellusten suorituskykyä voidaan mitata yleensä jo kehittäjälähtöisesti selaimesta löytyvien valmiiden työkalujen, kuten Google Chrome -selaimen mukana tulevan kehittäjien työkalujen avulla. Mitattavia kohteita voivat olla muun muassa sivuston osien, kuten HTML:n latautumiseen tai CSS:n piirtymiseen kuluva aika. Valmiiden sovellusten paketoitujen JavaScript-, CSS- sekä HTML-tiedostojen tiedostokoolla on merkitystä niiden verkon yli lähettämiseen kuluvaan aikaan, ja tämän vuoksi se on suorituskyvyn ohella merkittävä vertailun kohde.

Kuten suosion vertailussa, voidaan suorituskyvyn ja tiedostokoon vertailussa arvioida sovelluskehyksiä joko vertailemalla raakadataa tai sisällyttämällä suorituskyvyn ja tiedostokoon vertailu osaksi laajempaa matemaattista vertailumallia,

kuten lähteessä [7]. Kyseisen mallin muodostuksen lähtökohtana on päätösteoria (engl. decision theory), jonka metodeihin kuuluu usean attribuutin hyödyllisyysteoria (engl. Multi-attribute Utility Theory, MAUT), metodi esiteltiin alun perin R. L. Keeney:n ja H. Raiffa:n vuonna 1976 julkaisemassa teoksessa ”Decisions with Multiple Objectives” [11]. Teorian ideana on mahdollisten valintojen hyötyarvojen evaluointi sekä niiden valintaan liittyvien tekijöiden mallintaminen todennäköisyyksinä, perustuen probabilistiseen näkemykseen ihmisen päätöksenteosta. Tiedon lähettämiseen ja renderöintiin kuluvan ajan lisäksi voidaan suorituskykyä mitata esimerkiksi mittaamalla RESTful-sovelluksissa tietopyyntöjen lähettämiseen ja vastaanottamiseen kuluvaa aikaa tai MVC-arkkitehtuuriin pohjautuvissa web-sovelluksissa näkymien päivittämiseen kuluvaa aikaa ja sen vaikutusta käyttöliittymän interaktiivisuuteen.

3.3 Sovelluskehitys ja ohjelmistotuotanto

Sovelluskehyksien abstraktimpia ominaisuuksia ovat sellaiset asiat kuten kehyksen käyttäjäyhteisö, arkkitehtuuri sekä dokumentaatio. Näiden vertailu onkin aikaisempia vertailukohteita monimutkaisempaa, ja keskittyykin kvantitatiivisten metodien sijaan enemmän kvalitatiiviseen tarkasteluun. Artikkelissa ”A model to evaluate front-end frameworks for single page applications written in JavaScript” (S. Abrahamsson, 2023) [12] aloitettiin kriteerien määrittely tekemällä kehittäjille haastatteluja, joissa pyydettiin arvioimaan tiettyjen kriteerien tärkeyttä web-sovelluskehityksen kannalta asteikolla yhdestä neljään. Kriteerejä selvityksessä olivat käytön helppous, hyvä arkkitehtuuri, päivitysten määrä, käyttäjäyhteisö, soveltuvuus, dokumentaatio, aikaisempi käyttökokemus, suorituskyky sekä koodin määrä.

G. Kaur ja R. Gaurang [1] tarkastelivat sovelluskehitykseen ja ohjelmistotuotantoon liittyviä seikkoja kvalitatiivisesti, kiinnittäen huomiota kehysten skaalautuvuuteen, oppimisen vaikeusasteeseen, englanniksi ”learning curve”, yhteisön tukeen

sekä dokumentaatioon. Lisäksi kiinnitettiin huomiota nykyään vallitsevalle mobiilipohjaisten web-sovellusten kysynnälle sekä kehysten koodin luettavuuteen ja ylläpidettävyyteen. Sovelluskehitys ja ohjelmistotuotanto ovat suosioon ja suorituskykyyn verrattuna monimutkaisempia käsitteitä. Käsitteet kuten kehittäjän tuottavuus, koodin ylläpidettävyys sekä tietoturva ovat vaikeampia määritellä formaalisti, joten niiden tarkastelu aineistoissa keskittyykin kvalitatiiviseen eli laadulliseen analyysiin sekä kehittäjille suunnattuihin kysely- ja haastattelututkimuksiin. Toisaalta sovelluskehitykseen ja ohjelmistotuotantoon liittyy käsitteitä, joita voidaan myös kvantitatiivisesti vertailla. Esimerkiksi dokumentaation kattavuutta voitaisiin mitata määrittämällä, kuinka suuri osa frontend-kehyksen lähdekoodin tiedostoista sisältää edes jonkinlaista dokumentaatiota.

3.4 Kooste aineistoissa käytetyistä vertailumetodeista

Talukossa 3.2 on koostettu tässä tutkielmassa käytettyjen aineistojen käyttämät vertailumetodit sekä aineistojen julkaisuvuosi. Kuten taulukosta näkee, on vaihtoehtoja web-sovelluskehysten arviointiin sekä vertailuun paljon, lähtien perustavanlaatuisista eroista kuten kvalitatiivisen ja kvantitatiivisen tarkastelun valinnasta. Myös oikeiden kriteerien valinta vertailuun on olennaista, eli tarkastellaanko vain esimerkiksi kehysten suosiota vai myös teknisiä ja suorituskykyyn liittyviä ominaisuuksia tai jopa kehittäjien omaa kehityskokemusta kehysten käytöstä.

Matemaattisen vertailumallin kehittäminen vertailua varten vaikuttaisi olevan kirjallisuudessa ainakin ideana olemassa, mutta mitään yleiseen käyttöön levinnyttä web-sovelluskehysten matemaattista vertailumallia ei vielä ole ilmaantunut. Painokertoimien määritys eri kriteereille kertoo suoraan siitä, millaisessa tärkeysjärjestyksessä vertailumallin kehittäjät ovat nähneet eri vertailukohteet, kuten esimerkik-

Taulukko 3.2: Kooste aineistoissa käytetyistä arviointimetodologioista

Julkaisu	Julkaisuvuosi	Laadullinen vertailu	Suorituskyvyn vertailu	Matemaattinen vertailumalli	Kvalitatiiviset vertailumetodit	Suosio tai läsnäolo Internetissä
Multi-Attribute Decision-Making Model for Ranking of Web Development Frameworks	2021			✓		
A model to evaluate frontend frameworks for single page applications written in JavaScript	2023			✓		
Modern JavaScript frameworks: A Survey Study	2018	✓	✓			✓
Modern Web Frameworks: A Comparison of Rendering Performance	2022	✓	✓			
A Comparative Analysis of Modern Frameworks for Building Large-scale Web Applications	2023		✓		✓	
Comparison and Analysis of Popular Frontend Frameworks and Libraries: An evaluation of Parameters for Frontend Web Development	2023	✓			✓	✓

si valmiiden sovellusten tiedostokoon, GitHub-tähtien määrän kehykselle tai keskimääräisen ensimmäisen tekstin piirtymiseen kuluvan ajan. Vankan arviointimetodin kehitystä myös vaikeuttaa web-tekniikoiden ja standardien perinteistä ohjelmistotuotantoa nopeampi kehitys ja muutos. Arviointikohteiden määrityksessä olisikin hyvä hyödyntää suoraan web-sovelluskehityksessä työskentelevien kehittäjien käytännön kokemusta, jotta kyettäisiin mahdollisimman tarkasti määrittämään, missä web-sovelluskehykset yleensä epäonnistuvat. Kehittäjille tehdyt haastattelut (kvalitatiivinen analyysi) antavat hyvän pohjan arviointikohteiden määritykselle.

Varsinaisen datan keräys voidaan myös toteuttaa useilla eri tavoilla: pääsääntöisesti voidaan joko tehdä omaa empiiristä tutkimusta tai käyttää netistä löytyviä tilasto- ja analytiikkapalveluja, kuten GitHub:n tai StackOverflow:n tarjoa-

mia tietoja. Omia mittauksia voidaan tehdä esimerkiksi toteuttamalla eri frontend-sovelluskehyksillä sama web-käyttöliittymä, ja vertailemalla sen jälkeen syntyneen sovelluksen main.js- tai index.html -tiedoston kokoa kilotavuissa tai ensimmäisen sisällön piirtymiseen kuluvaa aikaa millisekunneissa. Nykytilanne vaikuttaisi olevan, että frontend-kehyksiä on laajassa yksityisen ja julkisen sektorin käytössä vain neljä, React, Angular, Svelte sekä Vue. Uusia voi tulla tilalle ja suosion järjestys voi tulevaisuudessa muuttua, mutta trendi, jossa frontend-kehitys vielä merkittävästi automatisoituu muun muassa laajojen kielimallien ja No Code -liikkeen vuoksi tulevina vuosina, tuskin tulee katoamaan.

4 Yhteenveto

Tässä tutkielmassa pyrittiin selvittämään, miten web-sovelluskehyksiä on arvioitu. Ensimmäisenä tutkimuskysymyksenä TK1 oli ”Millaisilla lähestymistavoilla web-sovelluskehysten suoriutumista sovelluskehityksessä sekä ohjelmistotuotannossa on lähdetty mallintamaan?”. Katsauksessa selvisi, että eri aineistoissa, sekä osittain myös saman aineiston sisällä, on käytetty useita erilaisia tutkimusmetodologioita sekä mallinnustapoja, kuten kyselytutkimuksia, kvalitatiivista vertailua sekä tilastollisten vertailumallien kehittämistä. Aineistoissa arviontia varten käytetyt metodologiat eriteltiin vielä omaksi taulukokseen 3.4-lukuun. Aineistot saatiin lopulta kategorisoitua lähestymistavoiltaan kolmeen pääkategoriaan: Suosion arviointi, suorituskyvyn ja valmiiden sovellusten tiedostokoon arviointi sekä viimeisenä arvionti ohjelmistotuotannollisesta näkökulmasta.

Toisena tutkimuskysymyksenä TK2 oli ”Mitä arviointimetodeja kehysten arvioinnissa on käytetty?”. Katsauksessa todettiin, että kehysten arviointiin on käytetty monenlaisia arviointimetodeja, kuten raakadatan, eli esimerkiksi GitHub-tähtien tai StackOverflow-survey-kyselyn tulokset, vertailua suosion arvioinnissa tai esimerkiksi renderöintiin kuluvan ajan mittaamista suorituskyvyn arvioinnissa. Kolmantena tutkimuskysymyksenä TK3 oli ”Millaisia tuloksia arvioinneissa on saavutettu?”. Katsauksessa selvisi, että suosion kannalta frontend-sovelluskehykset klusteroituvat neljään suurimpaan kehykseen: React, Angular, Svelte sekä Vue, tämän lisäksi eriteltiin muita aineistoissa saavutettuja tuloksia. Varsinaisia ristiriitoja eri aineistojen

tulosten välille ei vaikuttanut nousevan esiin, osittain siitä syystä, että osa aineistoista keskittyi web-sovelluskehyksiin yleisellä tasolla, käsittäen näin sekä selaimen että palvelimen sovelluksien toteuttamisessa käytettävät kehykset, kun taas osa aineistoista rajasivat itsensä keskittymään ainoastaan frontend-kehysten arviointiin. Mahdollisiksi jatkotutkimuskysymyksiksi nousi muun muassa tarve kehittää jonkinlaiset yhteiset arviointistandardit frontend-kehyksiä varten, koko ohjelmistoteollisuuden käyttöön. Voitaisiin myös tutkia, onko suosio vai tekninen suorituskyky viime kädessä parempi mittari frontend-kehyksen hyödyllisyydelle, sekä sitä, miten paljon moduulien dokumentaation kattavuusaste korreloi suosion kanssa.

Lähdeluettelo

[1] G. Kaur ja R. G. Tiwari, "Comparison and Analysis of Popular Frontend Frameworks and Libraries: An Evaluation of Parameters for Frontend Web Development", teoksessa *2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC)*, 2023, s. 1067–1073. DOI: 10.1109/ICESC57686.2023.10192987.

[2] A. Volle, "Web application", *Encyclopedia Britannica*, 2022. url: https://www.britannica.com/topic/Web-application (viitattu 09. 05. 2024).

[3] I. Hickson, "HTML5 Specification", World Wide Web Consortium, tekninen raportti 1.4802, 2011. url: https://www.w3.org/TR/2011/WD-html5-20110405/ (viitattu 05. 09. 2024).

[4] W3C, "The XMLHttpRequest Object", tekninen raportti, 2006. url: https://www.w3.org/TR/2006/WD-XMLHttpRequest-20060927/ (viitattu 16. 04. 2024).

[5] R. E. Johnson ja B. Foote, "Designing reusable classes", *Journal of object-oriented programming*, vol. 1, nro 2, s. 22–35, 1988.

[6] Stack Overflow. "Stack Overflow Trends". (2023), url: https://insights.stackoverflow.com/trends (viitattu 16. 04. 2024).

[7] D. Borissova, Z. Dimitrova, V. Dimitrov, R. Yoshinov, M. Garvanova ja I. Garvanov, "Multi-Attribute Decision-Making Model for Ranking of Web Development Frameworks", teoksessa *2021 25th International Conference on*

Circuits, Systems, Communications and Computers (CSCC), 2021, s. 3–8. DOI: 10.1109/CSCC53858.2021.00009.

[8] S. Delcev ja D. Draskovic, "Modern JavaScript frameworks: A Survey Study", s. 106–109, 2018. DOI: 10.1109/ZINC.2018.8448444.

[9] R. Ollila, N. Mäkitalo ja T. Mikkonen, "Modern Web Frameworks: A Comparison of Rendering Performance", *Journal of Web Engineering*, vol. 21, nro 3, s. 789–813, 2022. DOI: 10.13052/jwe1540-9589.21311.

[10] P. Singh, M. Srivastava, M. Kansal, A. P. Singh, A. Chauhan ja A. Gaur, "A Comparative Analysis of Modern Frontend Frameworks for Building Large-Scale Web Applications", teoksessa *2023 International Conference on Disruptive Technologies (ICDT)*, 2023, s. 531–535. DOI: 10.1109/ICDT57929.2023.10150911.

[11] R. L. Keeney ja H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*. Cambridge University Press, 1993.

[12] S. Abrahamsson, "A model to evaluate front-end frameworks for single page applications written in JavaScript", 2023.