Tampereen yliopisto

Ville Niemi

WEB-KEHITYKSEN HAASTEET

React-käyttöliittymäkirjaston ongelmat Stack Overflow -sivuston kysymyksissä

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu -tutkielma
Marraskuu 2022

TIIVISTELMÄ

Ville Niemi: Web-kehityksen haasteet: React-käyttöliittymäkirjaston ongelmat Stack Overflow -sivuston kysymyksissä
Pro gradu -tutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Marraskuu 2022

Tässä tutkielmassa käsitellään web-kehityksessä kohdattuja haasteita, ongelmia ja virhetilanteita. Tutkimusosuudessa perehdytään React-käyttöliittymäkirjaston parissa kohdattuihin ongelmiin Stack Overflow -kysymysten perusteella. Kysymyksille suoritetaan LDA-analyysi.

React (myös ReactJS, React.js) on ilmainen avoimen lähdekoodin selainpuolen JavaScript-kirjasto, joka on tarkoitettu käyttöliittymien rakentamiseen. Stack Overflow on ohjelmointiin suuntautunut kysymys-vastaus-sivusto. Latent Dirichlet Allocation -menetelmän (LDA-menetelmä) avulla voidaan luokitella tekstidokumenteille aiheita ja aihemalleille olennaisimpia avainsanoja.

Stack Overflow -sivustolta ladataan vuoden ajalta kaikki *reactjs*-tunnisteella löytyvät kysymykset (96 698 kappaletta). Kysymyksille suoritetaan esikäsittely eli muun muassa siistimistä, sulkusanojen poisto ja stemmaus. Lopuksi käsitellyille kysymyksille suoritetaan LDA-analyysi. LDA-algoritmissa käytetään 11 aihetta ja tuloksia havainnollistetaan taulukko- ja kuvamuodossa.

Aiheet nimettiin manuaalisesti. Aiheiden nimet ja prosenttiosuudet kaikista kysymyksistä ovat seuraavat: komponenttien hahmontaminen (18,5), virheet projekteissa (15,2), data ja taulukot (12,6), kuvat (11,5), ohjelmointirajapinnat ja palvelinpuoli (10,6), komponentit (7,3), tila (6,3), käyttöliittymän ulkoasu (5,8), reititys (5,4), lomakkeet (4,7) ja node-moduulit (2,0). Suurin osa Reactin ongelmista koskivat komponentteja, mutta noin kolmannes liittyi palvelinpuoleen ja yleisiin virheisiin projekteissa.

Aivan samankaltaista LDA-menetelmää ja tietyn käyttöliittymäkirjaston kysymyksiä analysoivaa tutkimusta ei ole aiemmin tehty, mutta pientä yhtäläisyyttä tämän ja aiempien tutkimusten aiheiden prosenttiosuuksien kanssa löytyi. Aikaisempien tutkimusten kysymysten aihealueen ollessa eri, suuria yleistyksiä ei voida kuitenkaan tehdä.

Avainsanat: reactjs, react, web, virhe, lda, Latent Dirichlet Allocation, Stack Overflow

Sisällys

1 Johdanto

Web-kehitys on IT-alan tämän hetken suosituimpia suuntauksia. Siitä kertoo muun muassa JavaScriptin asema GitHubin suosituimpana ohjelmointikielenä ja useat avoimet web-kehittäjän työpaikat. Useat sovellukset siirtyvät käytettäväksi nettiselainten kautta internetin välityksellä, koska silloin vältetään tarve asentaa erillisiä sovelluksia ja tiedon ollessa tallennettu pilveen palveluita voidaan käyttää helposti useiden laitteiden välillä.

Web-kehityksessä on myös omat haasteensa, jotka voivat koskea muun muassa nettisivujen ulkoasun suunnittelua. Kauniiden ja helppokäyttöisten sivujen kanssa on mielekkäämpää vuorovaikuttaa, joten suunnitteluun kannattaa panostaa. Sivujen nopeus ja tehokkuus on myös seikka, johon on hyödyllistä panostaa, sillä käyttäjät eivät jaksa odottaa hitaiden sivujen latautumista. Jos nettisivujen tietoturvaa laiminlyödään, asiakkaille ja ylläpitäjille saattaa koitua ongelmia tietomurron sattuessa, joten tietoturva kannattaa pitää ajan tasalla. Myös oikeiden tekniikoiden valitsemisella ja tulevaisuuden skaalautuvuuden suunnittelulla vähennetään tulevaisuudessa kohdattuja haasteita.

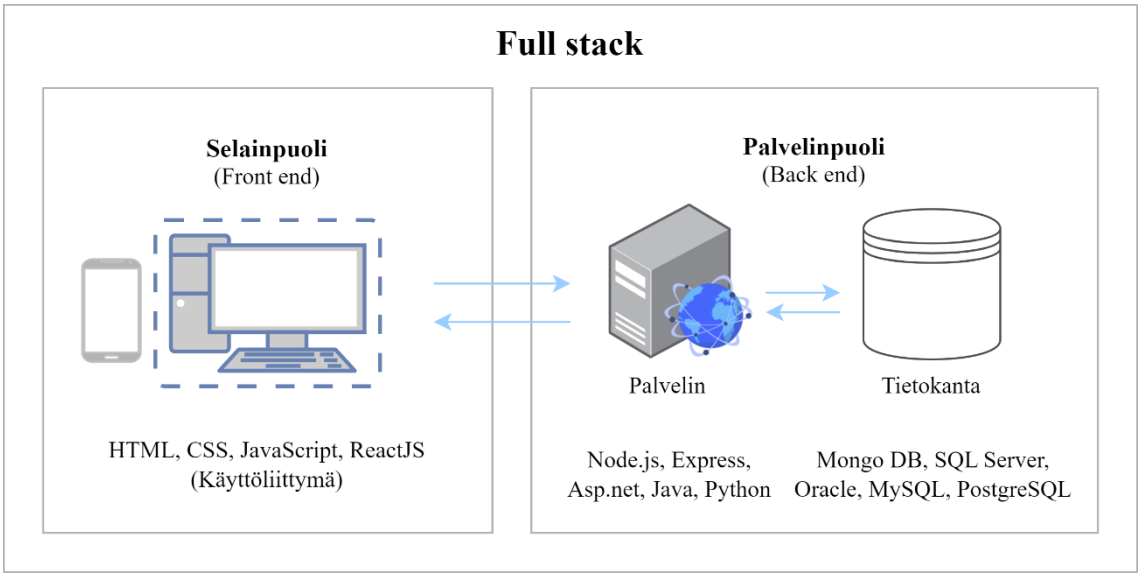
Tämän tutkielman tarkoituksena on selvittää web-kehityksessä ilmeneviä haasteita, ongelmia ja virhetilanteita, joita eri kehittäjät kohtaavat ohjelmoidessaan nettisivuja. Tutkielma on kirjallisuuskatsaus ja käytännön osuutena tutkielma pureutuu tarkemmin Stack Overflow -sivustolla esitettyihin kysymyksiin koskien React-käyttöliittymäkirjastoa. Kysymyksistä pyritään analysoimaan koneellisesti ja selvittämään millaisia ongelmia ihmiset kokevat Reactin parissa.

Tutkielman toisessa luvussa esitellään web-kehitystä yleisellä tasolla, käydään läpi HTML:n, CSS:n ja JavaScriptin perusteita, esitellään palvelinpuolen ohjelmointia ja ohjelmointirajapintoja yleisellä tasolla. Kohdassa 2.4 esitellään aiemmissa tutkimuksissa havaittuja virheitä HTML-, CSS- ja JavaScript-kielten parissa. Kolmannessa luvussa esitellään React-käyttöliittymäkirjastoa. Neljännessä luvussa esitellään Stack Overflow -sivustosta tehtyä tiedonlouhintatukimusta ja etenkin tutkimusta, joka on hyödyntänyt LDA-menetelmää (Latent Dirichlet Allocation). Viidennessä luvussa esitellään React-kysymyksiä Stack Overflow -sivustolla analysoiva tutkimus, sen toteutus, tulokset ja analyysi. Lopussa on yhteenveto tutkielmasta ja viiteluettelo.

2 Web-kehitys

Web-kehityksellä tarkoitetaan kaikkia niitä tehtäviä, jotka liittyvät internet-sivujen kehitykseen ja ylläpitoon. Web-kehitykseen kuuluu muun muassa internet-sivujen suunnittelu, käyttöliittymien toteutus, palvelinpuolen ohjelmointi ja tietokannan hallinta. [Techopedia, 2020]

Web-kehitys jaotellaan yleensä selainpuoleen (front end) ja palvelinpuoleen (back end). Selainpuoleen kuuluvat pääsääntöisesti suoraan käyttäjälle näkyvät asiat, kuten internet-sivujen ulkoasujen toteutus. Palvelinpuoleen kuuluvat muun muassa tiedonvälityksen toteutus käyttöliittymän ja tietokannan välille. On myös kehittäjiä, jotka toteuttavat sekä selainpuolta että palvelinpuolta (full stack). Kuvassa 1 on havainnollistettu tätä jaottelua.



Kuva 1. Web-kehityksen jaottelu ja esimerkkejä käytetyistä kielistä.

2.1 Selainpuoli

Selainpuolen web-kehityksessä suunnitellaan ja toteutetaan internet-sivuihin käyttöliittymiä ja toiminnallisuuksia. Selainpuolen kehityksessä määritellään miten internet-sivu näyttää kuvia, mitä eri painikkeet tekevät, mihin hyperlinkit vievät ja miltä sivusto visuaalisesti näyttää. Selainpuolen web-kehitys hyödyntää muun muassa seuraavia kieliä: HTML (HyperText Markup Language), CSS (Cascading Style Sheets) ja JavaScript.

2.1.1 HTML

HTML on standardoitu merkkauskieli, joka on suunniteltu tiedon esittämiseen internet-selaimissa. HTML5 on tällä hetkellä kielen uusin versio. HTML kuvaa internet-sivun semanttisen rakenteen, jota voidaan myöhemmin muokata lisää CSS-kielellä.

HTML-sivut koostuvat HTML-elementeistä. Elementit koostuvat tunnisteista (tag), jotka koostuvat tunnisteen nimestä ja kulmasulkeista, esimerkiksi <title> on yksi HTML-kielen lukuisista tunnisteista. Elementti koostuu aloitus- ja lopetustunnisteesta, joidenka väliin tulee elementin sisältö. Lopetustunniste on muotoa </title>, eli tunniste sisältää kauttaviivan pienempi kuin -merkin jälkeen.

```
<title>Otsikko</title>
```

Title-elementillä kuvataan internet-sivuston otsikkoa. Internet-sivuston otsikko näytetään yleensä selaimen välilehdessä. [MDN, 2020a]

HTML-elementeille voidaan myös asettaa attribuutteja. Attribuutit sisältävät lisäinformaatiota, joka ei näy suoraan elementin sisällössä.

```
<p class=”huomautus”>Tämä on kappaleen sisältö</p>
```

Yllä olevassa esimerkissä *class* on attribuutin nimi ja *huomautus* on attribuutin arvo. Attribuutin ja tunnisteen välille tulee välilyönti ja useat attribuutit erotetaan toisistaan myös välilyönneillä. Attribuutin nimen ja arvon väliin tulee yhtäsuuruusmerkki. Attribuutin arvo asetetaan lainausmerkkien väliin. Esimerkin kaltaista attribuuttia voidaan hyödyntää CSS-tyylimäärittelyissä. [MDN, 2022a]

HTML-elementtejä voi myös asettaa sisäkkäin. Esimerkiksi <p>-elementin sisälle voi asettaa <b>-elementin, joka lihavoi sisällään olevan tekstin. Alla olevassa esimerkissä todella-sana on lihavoitu:

```
<p>Pihalla on <b>todella</b> kylmä sää</p>
```

Joillain HTML-elementeillä ei ole ollenkaan itse sisältöä, vaan ne ovat niin kutsuttuja tyhjiä elementtejä. Kuvan lisäämiseen tarkoitettu <img>-elementti on esimerkiksi tällainen.

```
<img src="kuvat/logo.jpg" alt="Yrityksen logo" width="300" height="120">
```

Elementtiin tulee src-attribuutiksi kuvan sijainti ja mahdollisesti muita attribuutteja, kuten alt-attribuutti, joka kuvaa tekstimuodossa kuvaa. Alt-attribuutista on hyötyä näkörajoitteisten henkilöiden kohdalla ja silloin kun kuvan latauksessa tapahtuu virhe. Kuvan kokoa voidaan myös säätää height- ja width-attribuuteilla.

HTLM-dokumentti alkaa <!DOCTYPE html>-elementillä, jonka tehtävänä on määritellä, että dokumentti on HTML-koodia [MDN, 2022a]. Tämän jälkeen tulee <html>-elementti, joka sisältää koko dokumentin sisällön. Kyseinen <html>-elementti jakautuu

kahteen osaan <head>- ja <body>-elementteihin. Ensin tulevaan <head>-elementtiin voidaan sisällyttää tietoa, jota ei näytetä suoraan käyttäjälle, kuten dokumentin merkistökoodaus (esim. utf-8), avainsanat, sivun otsikko ja mahdollisesti linkki CSS-tyylitiedostoon. Lopuksi tulevassa <body>-elementissä on itse sivun sisältö, kaikki se sisältö, mitä käyttäjälle halutaan näytettävän: kuvat, videot, tekstit. Koodiesimerkissä 1 on esimerkki yksinkertaisesta HTML-dokumentin sisällöstä.

```
1.  <!DOCTYPE html>
2.  <html>
3.    <head>
4.      <meta charset="utf-8">
5.      <title>Sivun otsikko</title>
6.    </head>
7.    <body>
8.      <img src="kuvat/logo.jpg" alt="Yrityksen logo" width="300" height="120">
9.      <p>Tässä on tekstikappale.</p>
10.   </body>
11. </html>
```

Koodiesimerkki 1. Yksinkertainen HTML-dokumentti.

Muita yleisimpiä HTML-elementtejä ovat otsikko ja erityyppiset listat. Tekstin yhteydessä olevia otsikoita merkitään <h1>-elementeillä. Elementissä esiintyvä numero voi vaihdella riippuen siitä, mitä otsikkotasoa halutaan käyttää. Mitä isompi numero on, sitä pienempi - matalamman tason - otsikko on kyseessä. Matalin mahdollinen otsikko on <h6>. Listojen kuvaamisen kaksi yleisintä tapaa: epäjärjestetty ja järjestetty lista. Epäjärjestettyä listaa kuvataan -elementeillä ja järjestettyä listaa -elementeillä. Listaelementtien sisälle tulevat nimikkeet kuvataan -elementeillä. Koodiesimerkissä 2 on järjestämätön lista kolmella nimikkeellä. [MDN, 2022a]

```
1.  <ul>
2.    <li>ensimmäinen</li>
3.    <li>toinen</li>
4.    <li>kolmas</li>
5.  </ul>
```

Koodiesimerkki 2. Järjestämätön lista kolmella nimikkeellä.

HTML-dokumenttien linkit muodostetaan <a>-elementillä, jonka attribuuttiin *href* tulee linkin osoite. Elementin sisältöön tulee teksti, jota painettaessa href-attribuutin linkki aukeaa. Esimerkiksi näin muodostettaisiin linkki Tampereen yliopiston kotisivuille:

```
<a href=”https://www.tuni.fi/”>Tampereen yliopiston kotisivut</a>
```

Kommentteja HTML-tiedostoihin voidaan kirjoittaa asettamalla haluttu kommenttiteksti <!-- --> merkkien väliin seuraavasti:

```
<!--Tämä on kommentti, jonka selain jättää huomiotta -->
```

2.1.2 CSS

CSS (Cascading Style Sheets) on standardoitu kieli, jolla muokataan internet-sivujen ulkoasua. CSS-kieltä käytetään HTML-elementtien tyylittelyyn. CSS-tyylimäärittelyillä voi esimerkiksi muokata fontin väriä ja asetella tekstiä sivun eri kohtiin. CSS-tyylimäärittelyt voidaan lisätä <style>-elementin sisään tai suoraan HTML-elementtiin, mutta yleisin tapa on kirjoittaa tyylimäärittelyt .css-päätteisiin tiedostoihin ja liittää ne HTML-tiedoston <head>-elementtiin <link>-elementillä seuraavan esimerkin tapaan:

```
<link href="styles/style.css" rel="stylesheet">
```

CSS-sääntöjoukko koostuu valitsimesta (selector), julistuksesta (declaration), joka taas koostuu ominaisuuksista (properties) ja ominaisuuksien arvoista (property values).

```
p {
  color: red;
  width: 400px;
  border: 2px solid black;
}
```

Koodiesimerkki 3. CSS-sääntöjoukko, joka kohdistuu <p>-elementtiin.

Koodiesimerkissä 3 valitsimena on <p>-elementti eli tekstikappale. Tekstikappaleeseen kohdistuu julistusosassa kolme ominaisuutta: tekstin värin vaihto punaiseksi, tekstikappaleen leveyden asetus 400 pikseliä leveäksi ja tekstikappaleen rajaviivan asetus kaksi pikseliä paksuksi mustaksi yhtenäiseksi viivaksi. [MDN, 2022b]

CSS-valintajoukoissa on myös mahdollista käyttää useita eri valitsimia, joihin kaikkiin sovelletaan samoja ominaisuuksia. Valitsimet erotetaan toisistaan pilkuilla, esimerkiksi näin: p, h1, li { color: green }.

Alakohdan 2.1.2 valintajoukot ovat käyttäneet elementtivalitsimia, mutta valitsimia on myös erityyppisiä, kuten ID-valitsimet, luokkavalitsimet, attribuuttivalitsimet, pseudoluokkavalitsimet ja pseudoelementtivalitsimet. ID-valitsimet kohdistavat julistuksena tietyn id:n omaavaan elementtiin, jota on tarkoitus olla vain yksi. CSS-valitsimessa id:tä kuvataan asettamalla ristikkomerkki (#) id:n nimen eteen, esimerkiksi näin: #minun-id. HTML-elementissä id lisätään elementtiin seuraavalla tavalla: <p id=”minun-id”>. Luokkavalitsimet kohdistavat julistuksena kaikkiin niihin elementteihin, joilla on kyseinen luokka-arvo, jota kuvataan CSS-valitsimissa muodossa piste (.) ja luokan nimi, esimerkiksi näin: .minun-luokka. HTML-elementtiin luokka merkitään muodossa: <p

class=”minun-luokka”>. Attribuuttivalitsimet kohdistavat julistuksena tietyn attribuutin omaaviin tiettyihin elementteihin. Attribuuttivalitsinta kuvataan CSS-valitsimissa muodossa valitsin ja hakasulkeissa valittu attribuutti, esimerkiksi näin: img[src], joka kohdistaa valintansa niihin <img>-elementteihin, joilla on src-attribuutti. Tällöin valinta kohdistuisi <img src=”kuva.jpg”>-elementtiin, mutta ei pelkkään <img>-elementtiin. Pseudoluokkavalitsimet kohdistavat julistuksensa tiettyihin elementteihin silloin, kun ne ovat tietyssä tilassa, esimerkiksi tilanne, jossa kursori on linkkielementin päällä, mutta linkistä ei paineta juuri silloin. Pseudoluokkavalitsinta kuvataan CSS-valitsimissa muodossa valitsin, jonka jälkeen tulee kaksoispiste ja haluttu pseudoluokka, esimerkiksi näin: a:hover. Pseudoluokkavalitsimella *hover* voidaan kohdistaa linkkielementtiin <a> tyylejä, silloin kun hiiri leijuu linkin päällä. Pseudoelementtivalitsimilla voidaan kohdistaa tyylejä elementin tiettyyn osaan. Pseudoelementtivalitsinta kuvataan muodossa valitsin, jota seuraa kaksi kaksoispistettä ja haluttu pseudoelementti. Esimerkiksi valitsin p::first-line kohdistaa valintansa <p>-elementin ensimmäiseen tekstiriviin. CSS-valitsimia on myös muita, kuten sisarussuhteisiin liittyvät valitsimet, esimerkiksi p + img, joka kohdistaa valintansa <img>-elementtiin, joka tulee heti <p>-elementin jälkeen. [MDN, 2022b; MDN, 2022j]

HTML-dokumenttien kirjasimien ulkoasua voidaan myös muuttaa käyttämällä fontteja. Fontin voi liittää HTML-dokumenttiin lisäämällä <head>-elementin sisälle linkin johonkin fonttiin, esimerkiksi näin:

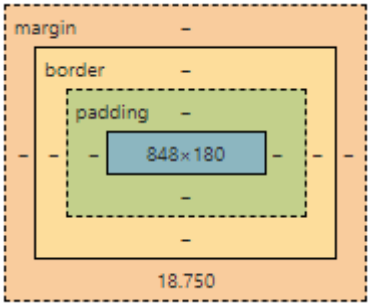
```
<link href="https://fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet">
```

Yllä oleva linkki lataa Open Sans -fonttiperheen samalla kun sen sisältävä HTML-dokumentti avataan selaimessa. HTML-dokumentti saadaan hyödyntämään Open Sans -fontteja lisäämällä CSS-tiedostoon seuraava julistus:

```
html { font-family: ”Open Sans”, sans-serif; }
```

Yllä oleva julistus siis asettaa fontiksi Open Sans -fonttiperheen sans-serif–fontin. Sans-serif fonteilla tarkoitetaan kirjaimia, joissa ei ole pääteviivaa. Tämä tutkielma käyttää taas serif-tyylistä eli pääteviivallista fonttia.

Fontin kokoa voidaan kasvattaa *font-size* ominaisuudella. Arvo 12px kasvattaa fontin kokoa eli korkeutta 12 pikseliä korkeaksi. Riviväliä voidaan kasvattaa *line-height* ominaisuudella ja kirjainten väliä *letter-spacing* ominaisuudella. Teksti voidaan keskittää *text-align* ominaisuudella ja sen arvolla *center*.



Kuva 2. CSS:n laatikkomalli.

CSS:n ulkoasu perustuu laatikkomalliin, jolla tarkoitetaan HTML-elementtejä ympäröiviä osia. Kuvassa 2 on havainnollistettu CSS:n laatikkomallia, jossa elementin sisällön ympärillä on pehmuste (padding), jonka jälkeen tulee reuna (border) ja lopuksi marginaali (margin). Laatikkomallin kaikkia osia voidaan muokata CSS-tyyleillä, niin kokoa kuin väriäkin.

Kommentteja CSS-tiedostoihin voi lisätä asettamalla tekstiä /* ja */ merkkien väliin.

2.1.3 Selainpuolen JavaScript

JavaScript on selaimissa käytettävä komentokieli, jolla voidaan vuorovaikuttaa internetsivun kanssa. JavaScriptiä voidaan hyödyntää sekä selainpuolella että serveripuolella. JavaScript vuorovaikuttaa selaimen dokumentti-objekti-mallin kanssa. JavaScriptillä voidaan muuttaa HTML-dokumentin sisältöä perustuen käyttäjän toimiin, kuten klikkauksiin tai lomakkeella annettuihin tietoihin. [MDN, 2022h]

Eurooppalaisen tieto- ja viestintäjärjestelmien standardointiyhdistyksen Ecma International toimesta JavaScriptistä on luotu standardoitu versio – ECMA Script (ES). JavaScript on standardoitu ECMA:n standardissa ECMA-262. Uusin versio on vuodelta 2021 [ECMA, 2021]. ECMA Scriptistä on olemassa myös ISO standardi ISO/IEC 22275:2018 [ISO, 2018].

JavaScript-koodia voi kirjoittaa joko sisäisesti HTML-dokumenttiin <head>- tai <body>-elementtien sisälle <script>-tunnisteiden väliin, tai ulkoisesti .js päätteiseen tiedostoon, joka linkitetään HTML-dokumentin <head>- tai <body>-elementissä. [MDN, 2022h]

```
1.  <!DOCTYPE html>
2.  <html>
3.  <head>
4.  <script>
5.  function muutaTeksti() {
6.    document.getElementById("kappale").innerHTML = "Muutettu tekstikappale.";
7.  }
8.  </script>
9.  </head>
10. <body>
11.
12. <p id="kappale">Tekstikappale</p>
13.
14. <button type="button" onclick="muutaTeksti()">Paina tästä</button>
15.
16. </body>
17. </html>
```

Koodiesimerkki 4. HTML-dokumentti, joka sisältää upotettua JavaScript-koodia.

Koodiesimerkissä 4 on esimerkki HTML-dokumenttiin upotetusta JavaScript-koodista. Dokumentissa on painike, jota painamalla kutsutaan *muutaTeksti*-funktiota, joka muuttaa kappale id-arvon omaavan tekstikappaleen sisällön muotoon ”Muutettu tekstikappale”. JavaScriptin avulla siis vuorovaikutetaan HTML-dokumentin kanssa, jolloin käyttäjän toimien seurauksena sivuston sisältö muuttuu.

JavaScriptin syntaksissa ja toiminnoissa on paljon samaa muiden ohjelmointikielten kanssa. JavaScript on löysästi tyypitetty dynaaminen kieli, eli muuttujiin voidaan asettaa erityyppisiä arvoja (kuten numeroita ja kirjaimia) ilman, että se ennalta määritellään, ja ohjelman tyyppitarkastus suoritetaan vasta ohjelman ajoaikana. Muuttujia voidaan esitellä avainsanoilla *const*, *let* ja *var*. [MDN, 2022h]

JavaScriptissä on kuusi eri tapaa julistaa funktio:

1. Nimetty funktion julistus: function nimetty(...){}
2. Anonyymi funktion ilmaisu: var anonyymi = function(...){};
3. Nimetty funktion ilmaisu: var anonyymiNimetty = function anNimetty(...){};
4. Välittömästi herätetty ilmaisu: var herätetty = (function(){return function(...){}})();
5. Funktion rakentaja: var rakentaja = new Function();
6. Nuolifunktio: var nuoli = (...) => {};

Nimetty funktion julistus on perinteisin tapa julistaa funktio. Siinä funktioita kuvataan function-avainsanalla, jonka jälkeen tulee funktion nimi ja sulut, jotka sisältävät mahdollisesti funktiolle annettavat parametrit. Funktion sisältö asetetaan aaltosulkeiden väliin. Nuolifunktio on taas kompaktein tapa julistaa funktio. Nuolifunktioiden avulla koodi pysyy siistinä ja helposti luettavana. [Abraham, 2021]

JavaScript tukee myös asynkronista ohjelmointia. Asynkronisessa ohjelmoinnissa on kyse tekniikasta, jossa suoritetaan mahdollisesti pitkään kestävä tehtävä, mutta sen sijasta, että odotettaisiin tehtävän valmistumista, jatketaan muiden tehtävien suorittamista samalla kun kyseistä tehtävää suoritetaan taustalla. Esimerkiksi kun suoritetaan HTTP-

pyyntö fetch-funktiolla, prosessiin kuuluu odottelua, joten on järkevää suorittaa muita tehtäviä odottelun aikana. Kun http-pyyntö saapuu, ohjelma vastaanottaa tulokset. [MDN, 2022d]

Vastakutsut (callbacks) olivat aiemmin JavaScriptin päätapa hyödyntää asynkronista ohjelmointia. Vastakutsuilla tarkoitetaan funktiota, joka siirretään toiselle funktiolle ja suoritetaan tiettynä ajankohtana. Yksi vastakutsujen kategoria on tapahtumakäsittelijät. Tapahtumakäsittelijästä esimerkki on addEventListener, jonka parametriksi voi asettaa muun muassa klikkauksen, jolloin tapahtuma aktivoituu, kun sen sisältämää elementtiä klikataan. Tätä voi hyödyntää esimerkiksi lomakkeen lähetä painikkeessa. Tapahtumakäsittelijä siis koko ajan seuraa klikataanko siitä, ja kun klikkaus tapahtuu, sen sisältä suoritetaan. Klikkausta odotellessa muiden tehtävien suoritus onnistuu taustalla. [MDN, 2022d]

Vuonna 2015 julkaistun ES6:n myötä asynkronista JavaScriptiä alettiin toteuttaa hyödyntämällä lupauksia (promise). Lupaus on asynkronisen funktion palauttama objekti, josta käy ilmi operaation nykytilanne. Lupauksiin voi kiinnittää käsittelijöitä, jotka käsittelevät operaation tuloksen sen valmistuttua. Lupauksella on kolme tilaa: avoin (pending), täytetty (fulfilled) ja hylätty (rejected). Avoin lupaus on luotu, mutta se ei ole vielä täytetty tai hylätty. Kun lupauksen tehtävä on onnistunut, se muuttuu täytetty-tilaan ja tiedot välitetään then-käsittelijälle. Kun lupaus epäonnistuu, tulee tilaksi hylätty ja tiedot välitetään catch-käsittelijälle. [MDN, 2022e]

```
1.  const fetchPromise = fetch('https://mdn.github.io/learning-area/ja-
    vascript/apis/fetching-data/can-store/products.json');
2.
3.  fetchPromise
4.  .then( response => {
5.    console.log(`Vastaanotettu vastaus: ${response.status}`);
6.  })
7.  .catch( error => {
8.    console.error(`Kohdattiin virhe: ${error}`);
9.  });
```

Koodiesimerkki 5. Esimerkki lupauksen käytöstä JavaScript-koodissa [MDN, 2022e].

Koodiesimerkissä 5 tehdään http-pyyntö fetch-funktiolla, joka on lupaus. Lupaukseen liitetään then-käsittelijä, joka tulostaa vastauksen statuskoodin lupauksen täytyttyä. Jos lupaus hylätään, catch-käsittelijä tulostaa virhekoodin. [MDN, 2022e]

Async- ja await-avainsanoilla voidaan myös toteuttaa lupausten tapaan asynkronisia funktioita. Lisäämällä async-avainsanan funktion alkuun tekee siitä asynkronisen funktion. Await-avainsana lisätään koodissa funktiokutsun eteen, joka palauttaa lupauksen. Async- ja await-avainsanojen avulla voidaan siis kirjoittaa synkroniselta koodilta näyttävää asynkronista koodia. Koodiesimerkissä 6 on esimerkki async- ja await-avainsanojen käytöstä. [MDN, 2022e]

```
1.  async function fetchProducts() {
2.      try{
3.          const response = await fetch('https://mdn.github.io/learning-area/ja
    vascript/apis/fetching-data/can-store/products.json');
4.          if (response.ok){
5.            console.log(`Vastaanotettu vastaus: ${response.status}`);
6.          }
7.      }
8.      catch(error) {
9.          console.error(`Kohdattiin virhe: ${error}`);
10.     }
11. };
12. fetchProducts();
```

Koodiesimerkki 6. Lupaus toteutettuna async- ja await-avainsanoilla. [MDN, 2022e]

2.2 Palvelinpuoli

Palvelinpuolella toteutetaan tiedonvälitys käyttöliittymän ja tietokannan välillä. Käyttäjä vuorovaikuttaa käyttöliittymän eli selainpuolen kanssa ja tarvittaessa käyttöliittymä kutsuu palvelinpuolen operaatioita, joilla noudetaan tietokannasta haluttua tietoa ja joka esitetään käyttäjälle käyttöliittymässä helposti luettavassa muodossa. Myös tiedon lähetys ja muokkaus käyttöliittymässä välittyy tietokantaan palvelinpuolen operaatioiden avulla. Palvelinpuoli hoitaa myös tiedonvalidoinnin, eli varmistetaan, että käyttäjän lähettämät tiedot ovat oikein muodostettuja ja paikkansapitäviä. Palvelinpuolella todennetaan käyttäjän identiteetti, eli varmistetaan käyttäjän olevan se kuka hän väittää olevansa ja tarkistetaan käyttäjän valtuutukset käyttää haluttua resurssia. Palvelinpuoli hoitaa myös lokitietojen tallennuksen ja CSRF-hyökkäyksiltä suojautumisen. Palvelinpuolta voidaan hoitaa web-ympäristössä esimerkiksi Node.js:llä ja sille saatavilla olevilla paketeilla, kuten Expressillä.

2.2.1 Node.js

Node.js on JavaScript-kielellä rakennettu alustariippumaton avoimen lähdekoodin ajoympäristö, joka pohjautuu Chromen V8 JavaScript moottoriin. Node.js-koodi suoritetaan palvelimen puolella, jolloin valmis nettisivu lähetetään käyttäjälle [Node.js, 2022].

Node.js:n vahvuuksia ovat hyvä suorituskyky ja se, että Node.js on kirjoitettu JavaScriptillä, jota käytetään myös selainpuolella, jolloin hallittavien kielten määrä ei kasva. Myös monet uudet kielet (TypeScript, CoffeeScript) kääntyvät JavaScriptille, joten niitä voidaan hyödyntää Node.js kehityksessä. Noden pakettimanagerin (NPM) avulla päästään käsiksi satoihin tuhansiin paketteihin, joita voidaan hyödyntää kehityksessä. NPM:ssä on myös hyvä riippuvuuksien hallintatyökalu, jolla voidaan helposti pitää kirjaa tarvittavista paketeista. Node.js on myös saatavilla useille käyttöjärjestelmille ja sitä tuetaan useiden web-isännöintipalveluntarjoajien toimesta. Node.js:llä on myös aktiivinen kehittäjäyhteisö. [MDN, 2022c]

```
1.  const http = require('http');
2.
3.  const hostname = '127.0.0.1';
4.  const port = 3000;
5.
6.  const server = http.createServer((req, res) => {
7.    res.statusCode = 200;
8.    res.setHeader('Content-Type', 'text/plain');
9.    res.end('Hei maailma');
10. });
11.
12. server.listen(port, hostname, () => {
13.   console.log(`Palvelin käynnissä osoitteessa http://${hostname}:${port}/`);
14. });
```

Koodiesimerkki 7. Yksinkertainen ”Hei maailma” Node.js-ohjelma.

Koodiesimerkissä 7 on yksinkertainen Node.js-ohjelma, joka palauttaa ”Hei maailma” -tekstin kun kyseiseen IP-osoitteen ja portin yhdistelmään otetaan yhteyttä internetselaimella. Node.js pystyy käsittelemään useita yhteyksiä samanaikaisesti ja kun tehtäviä ei ole, Node.js on lepotilassa. Node.js:n parissa ei ole riskiä lukkiutumista (deadlock), koska juuri mikään Node.js:ssä ei suoraan suorita syöttö- ja tulostusoperaatioita (input/output, I/O). Ainoastaan Node.js:n standardikirjaston synkroniset syöttö- ja tulostusoperaatiot lukitsevat resursseja. Tämän seurauksena skaalautuvia järjestelmiä on järkevää rakentaa Node.js:llä. [Node.js, 2022]

2.2.2 Express

Express on Node.js:n suosituin web-ohjelmistokehys. Expressin avulla voidaan kirjoittaa hypertekstin siirtoprotokollan (HTTP) pyynnöille käsittelijöitä. Express tukee URL-polkuja (routes) ja HTTP-verbejä (GET, PUT, DELETE…), joita käytetään HTTP-protokollan mukaiseen tiedonvälitykseen. Esimerkiksi GET-metodilla pyydetään informaatiota tietystä resurssista. Express tukee myös malleja, joihin palautettava data voidaan sovittaa ja Expressin avulla voidaan asettaa web-sovelluksille tyypilliset asetukset, kuten portit. Express tukee väliohjelmistojen (middleware) käyttöä. [MDN, 2022c]

```
1.  const express = require('express');
2.  const app = express();
3.  const port = 3000;
4.
5.  app.get('/', function(req, res) {
6.    res.send('Hei maailma')
7.  });
8.
9.  app.listen(port, function() {
10.   console.log(`Esimerkkisovellus, joka kuuntelee porttia ${port}!`)
11. });
```

Koodiesimerkki 8. Yksinkertainen Express-sovellus.

Koodiesimerkissä 8 on esimerkki Express-sovelluksesta. Ensimmäisellä rivillä tuodaan Express-moduuli ja toisella luodaan Express-sovellus. Esimerkin keskivaiheilla oleva app.get-funktio määrittelee vastakutsun, joka suoritetaan, kun GET-muotoinen

http-kutsu kohdistuu polkuun / kyseisellä sivulla. Send-funktio palauttaa ”Hei maailma” -tekstin. Esimerkin lopussa oleva app.listen-funktio käynnistää sovelluksen porttiin 3000 ja tulostaa konsoliin viestin, että serveri on päällä portissa 3000. [MDN, 2022c]

2.3 Rajapinnat ja kehykset

Web-kehityksessä hyödynnetään erilaisia ohjelmointirajapintoja ja ohjelmistokehyksiä. Yksinkertaisesti ilmaistuna ohjelmointirajapinta mahdollistaa vuorovaikutuksen eri ohjelmien välillä ja ohjelmistokehykset tarjoavat valmiit rakennuspalikat ohjelmien luomiselle.

2.3.1 Ohjelmointirajapinnat

Ohjelmointirajapinnoilla tarkoitetaan toimintoja, joilla voidaan luoda vuorovaikutusta ohjelmien välille. Esimerkiksi tiedonvälitystä ja muokkausta. Ohjelmointirajapintojen avulla kehittäjät voivat toteuttaa monimutkaisempia rakennelmia helpommin, kun tarvittavia toimintoja ei tarvitse toteuttaa alusta alkaen.

Selainpuolen JavaScriptiin on saatavilla useita ohjelmointirajapintoja. Ohjelmointirajapinnat eivät ole itse osa JavaScriptiä, mutta ne on toteutettu JavaScriptillä tai jollain toisella ohjelmointikielellä. Ohjelmointirajapinnat voidaan jakaa kahteen pääkatekoriaan: selaimen ohjelmointirajapinnat ja kolmansien osapuolten rajapinnat. Selaimen ohjelmointirajapinnat ovat rakennettu itse internetselaimeen ja ovat kykeneväisiä käyttämään dataa selaimesta ja tietokoneympäristöstä. Esimerkiksi Web Audio API tarjoaa JavaScriptille työkaluja muokata ääntä selainympäristössä. Kolmansien osapuolten ohjelmointirajapinnat täytyy ladata internetin kautta. Esimerkiksi Twitter API:n avulla voi lisätä internetsivulle näkyviin twiittejä eli Twitterin viestejä. Ohjelmointirajapinta tarjoaa funktioita, joilla voi esimerkiksi hakea tietyn käyttäjän twiittejä. [MDN, 2022f]

2.3.2 Ohjelmistokehykset

Ohjelmistokehys tarjoaa pohjan rakennettavalle ohjelmalle. Ohjelmistokehys sisältää valmiita osia, joita voidaan hyödyntää uudestaan rakennettaessa uutta ohjelmaa. Tämä nopeuttaa kehitysprosessia ja antaa ohjelmalle pohja-arkkitehtuurin, josta ohjelmaa on helppo laajentaa ja ylläpitää. Ohjelmistokehykset ovat laajasti käytössä JavaScript kehityksessä. [MDN, 2022g]

JavaScriptin ohjelmistokehykset tarjoavat esimerkiksi apukeinoja sovelluksen tilan ja käyttöliittymän päivittämiseen, joka voi olla työlästä pelkällä HTML:llä ja puhtaalla JavaScriptillä. Ohjelmistokehykset tarjoavat myös työkaluja, joilla voidaan esimerkiksi luoda helposti testejä, virheentarkistusta ja tyylintarkistusta. Monet ohjelmistokehykset suosivat lokerointia, koodin jakamista muokattaviin ja uudelleenkäytettäviin komponentteihin, jotka voivat vuorovaikuttaa keskenään. Yhden komponentin koodi voi olla kaikki samassa tiedostossa, jolloin kehittäjien on helppo löytää kaikki tarvittava koodi ja tehdä

siihen muutoksia. Ohjelmistokehykset tarjoavat myös reititykseen työkaluja. Koska monet nykyajan nettisivut ovat niin kutsuttuja yhden sivun sovelluksia (single page app, SPA), ei varsinaisesti navigoida uusiin ositteisiin, vaan päivitetään vain näkymää. Jotta voitaisiin luoda linkkejä tiettyihin näkymiin sivulla, tarvitaan reititystoimintoja. Yhden sivun sovellusten reititystä kutsutaan selainpuolen reitityksekseksi (client-side routing). [MDN, 2022g]

Suosituimpia selainpuolen JavaScript-ohjelmistokehyksiä tällä hetkellä ovat React, Vue, Angular, jQuery ja Svelte [Shah, 2022].

2.4 Virhetilanteet

Tässä luvussa esitellään muissa tutkimuksissa havaittuja tyypillisimpiä virheitä ja ongelmia JavaScript-, HTML- ja CSS-kielten parissa.

2.4.1 HTML ja CSS

Park ja muut [2013] tutkivat laboratoriossa 20 osallistujaa, jotka suorittivat ohjelmointitehtäviä samalla kun heitä kuvattiin. Tutkijat hyödynsivät taidot-säännöt-tieto-kehystä jaotellakseen osallistujien tekemiä virheitä. Taidot-säännöt-tieto-kehyksen taidot-kategorian virheillä tarkoitettiin huolimattomuus virheitä, säännöt-kategorian virheillä tarkoitettiin huonojen käytäntöjen soveltamista ja tieto-kategorian virheillä tarkoitettiin tilanteen väärinymmärtämisestä aiheutuneita virheitä. Tutkimuksessa tutkittiin HTML- ja CSS-kielten parissa tehtyjä virheitä. Tutkijat teetättivät osallistujilla viisi eri tehtävää, jotka käsittelivät muun muassa tekstikappaleen lisäämistä, hyperlinkin lisäämistä, tekstin värin vaihtoa, bugien korjausta ja elementtien keskittämistä. [Park *et al.*, 2013]

Park ja muut [2013] tutkimuksessa taitoperustaisia virheitä oli eniten. Taitoperustaisiksi virheiksi laskettiin muun muassa typografiset virheet, kuten kirjoitusvirheet tunnisteiden (tag) ja arvojen kanssa. Sääntöperustaisia virheitä oli toiseksi eniten. Sääntöperustaisiksi virheiksi laskettiin muun muassa vanhentuneiden ja virheellisten elementtien ja ominaisuuksien käyttö. Vähiten oli tietoperustaisia virheitä. Tietoperustaisiksi virheiksi laskettiin muun muassa CSS-valitsimien perussyntaksin ymmärtämistä väärin. [Park *et al.*, 2013]

Myöhemmässä tutkimuksessaan Park ja muut [2015] tutkivat opiskelijoiden tekemiä syntaksivirheitä web-kehitykseen suuntautuneella yliopistokurssilla. Tutkijat arvioivat, että 20 % opiskelijoiden tekemistä virheistä liittyivät HTML-elementtien vanhempi-lapsi suhteiden käyttöön ja 35 % tunnisteiden syntaksiin. [Park *et al.*, 2015]

Park ja Wiedenbeck [2011] analysoivat web-kehitykseen suuntautuneet yliopistokurssin tukifoorumin viestejä. Tutkijat analysoivat HTML-, CSS- ja JavaScript-kieliin liittyviä haasteita. Viestit lajiteltiin seuraaviin kategorioihin seuraavilla prosenttiosuuk-

silla: kehitys (34,3 %), ohjeistus (29,9 %), teknologia (24,5 %), sisältö (6,6 %), suunnittelu (4,7 %). Kehitys-kategoriaan kuuluvat kysymykset käsittelivät muun muassa linkkien luontia, kuvia ja taulukoita. Ohjeistus-kategorian kysymykset pääasiassa pyysivät selvennystä kurssin tehtävänantoihin. Teknologia-kategorian kysymykset koskivat ohjelmoinnin ulkopuolisia teknologisia ongelmia, kuten shell-käyttäjien aktivointia ja FTP-ohjelmien (File Transfer Protocol) konfigurointia. Sisältö-kategorian kysymykset käsittelivät esimerkiksi tekijänoikeusasioita nettisivulle tulevien kuvien yhteydessä. Suunnittelu-kategorian kysymykset käsittelivät vinkkejä nettisivujen suunnitteluun liittyen. [Park ja Wiedenbeck, 2011]

2.4.2 JavaScript

Gyumesi ja muut [2021] tutkivat JavaScript-kielessä esiintyviä virheitä luoden BUGSJS-menetelmän, jolla voidaan luokitella JavaScript-ohjelmissa esiintyviä virheitä. Tutkijat luokittelivat JavaScriptissä esiintyvät virheet neljään yläkategoriaan, jotka ovat jaoteltu useisiin alakategorioihin, jotka myös ovat jaoteltu useisiin alakategorioihin. Yläkategoriat olivat seuraavat:

1. Keskeneräinen toiminnon toteutus.
2. Virheellinen toiminnon toteutus.
3. Yleiset ohjelmoinnin virheet.
4. Täydellistävä ylläpito.

Esimerkki keskeneräisestä toiminnon toteutuksesta on puuttuva syötteen vahvistus, kuten puuttuva tyypin tarkistus. Virheellisestä toiminnon toteutuksesta esimerkkitapaus on taas virheellinen syötteen vahvistus, kuten tarpeeton tyypin tarkistus. Yleiseen ohjelmoinnin virheet -kategoriaan kuuluivat muun muassa kirjotusvirheet ja puuttuvat muuttujien alustukset. Täydellistävän ylläpidon kategoriaan kuuluivat ohjelman parannukset ilman, että ohjelmassa olisi ollut erityisiä virhettä, mutta ne esimerkiksi paransivat ohjelman tehokkuutta. [Gyumesi *et al.*, 2021]

Gyumesi ja muut [2021] tutkimuksessa eniten virheitä löytyi virheellisistä toiminnon toteutuksista (162 kpl), toisiksi eniten keskeneräisistä toiminnon toteutuksista (128 kpl), kolmanneksi eniten yleisistä ohjelmoinnin virheistä (29 kpl). Täydellistävän ylläpidon kategoriassa ei eritelty tapauksia. Tutkimuksen esimerkkitapaukset oli saatu GitHub-sivustolta keräämällä. [Gyumesi *et al.*, 2021]

Ocariza ja muut [2017] tutkivat empiirisesti asiakaspuolen JavaScript-virheiden syitä ja seurauksia web-sovelluksissa. Tutkimuksessa analysoitiin 502 JavaScript-virhettä 19:stä eri projektista. Tutkimuksen tuloksissa havaittiin, että 68 % JavaScript-koodissa esiintyvissä virheistä liittyivät dokumenttioliomalliin (Document Object Model, DOM) ja 80 % suurimman vaikutuksen omaavista virheistä liittyivät dokumenttioliomalliin. Tutkijat luokittelivat JavaScript-virheet seuraavasti:

1. Virheellinen syötteen vahvistus.
2. Virhe kirjoittaessa merkkijonoliteraalia.
3. Puuttuva null-arvon tarkistus.
4. Selainten toiminnan erojen laiminlyönti.
5. Syntaksivirhe.

Tutkijat havaitsivat, että 55 % virheistä sopi yllä oleviin kategorioihin ja loput noudattivat sekalaisia malleja. Virheellisiä syötteen vahvistuksia oli 16 % havaituista virheistä. Virheillä kirjoittaessa merkkijonoliteraaleja tarkoitettiin esimerkiksi etuliitteiden puuttumista, kirjoitusvirheitä tai vääriä merkistökoodauksia. Näitä virheitä havaittiin 13 %. Puuttuvia null-arvon tarkistuksesta johtuvia virheitä oli 10 % tapauksista. Selainten toiminnan erojen laiminlyönnistä johtuvilla virheillä tarkoitettiin esimerkiksi sitä miten eri selaimet kohtelevat JavaScriptin eri metodeja. Näitä virheitä oli 9 %. JavaScript-koodin syntaksivirheitä oli 7 % havaituista virheistä. [Ocariza *et al.*, 2017]

Gao ja muut [2017] tutkivat JavaScriptin tyyppijärjestelmään liittyviä ongelmia. Tutkimuksessa analysoitiin 400 eri virheraporttia 398:sta eri projektista. Tutkijat käyttivät staattisia tyyppitarkastajia, kuten Facebookin Flow ja Microsoftin TypeScript, virheellisiin versioihin ohjelmista. Tyyppitarkastajat havaitsivat 15 % kaikista 400 virheestä. Nämä virheet olisi siis pystytty välttämään, jos ohjelman alkuperäiset kehittäjät olisivat hyödyntäneet kyseisiä tyypintarkastajia. [Gao *et al.*, 2017]

Hanam ja muut [2016] tutkivat projektien välisiä virhemalleja palvelinpuolen JavaScript-ohjelmissa. Tutkijat hyödynsivät 134 Node.js-projektia ja esittelivät BugAID-nimellä kutsutun menetelmän, joka koneoppimisen avulla oppii yleisimmät virhemallit JavaScript-koodista. Tutkijat esittelivät 13 toistuvaa virhekuviota, jotka esiintyivät useassa projektissa. Virhekuviot liittyivät muun muassa *null* ja *undefined* arvojen kanssa toimintaan ja käsittelemättömiin poikkeustapauksiin. [Hanam *et al.*, 2016]

Wang ja muut [2017] tutkivat rinnakkaisuusvirheitä Node.js-sovelluksissa. Tutkimus analysoi 57 virhettä 53 Node.js-sovelluksessa. Tutkijat analysoivat virheiden piirteitä, kuten virhemalleja, juurisyitä, virheiden vaikuttavuutta, virheiden ilmentyvyyttä ja korjausstrategioita. Virhemalleja esiteltiin kolme kappaletta: järjestyksen rikkominen, atomisuuden rikkominen ja nälkiintyminen (starvation). [Wang *et al.*, 2017]

Järjestyksen rikkomisella tarkoitettiin tilannetta, jossa kaksi epäsynkronista operaatiota, joidenka oletetaan käyttävän samaa jaettua resurssia, oletetaan suoritettavan tietyssä järjestyksessä, mutta todellisuudessa näin ei käy. Atomisuuden rikkomisella tarkoitettiin tilannetta, jossa joukko epäsynkronisia operaatioita suoritetaan limittäin, mutta atomisuutta ei varmisteta suorituksen aikana, kuten siten, että kaksi operaatiota saavat saman lähtötiedon tietokannasta, mutta toinen operaatioista ehtii muuttaa sitä ensin, toisen operaation olettaessa sen olevan edelleen sama. Nälkiintymiseksi luokiteltiin virheet, jotka veivät kauan aikaa ja estivät toisten tapahtumien suorituksen. Järjestyksen rikkominen

kattoi 30 % virheistä, atomisuuden rikkominen 65 % ja nälkiintyminen 5 %. [Wang *et al.*, 2017]

2.4.3 Virhetilanteiden yhteenveto

HTML:n ja CSS:n virheitä käsittelevissä tutkimuksissa nousivat esiin kirjoitus- ja syntaksivirheet ja HTML-elementit. JavaScriptin virheitä käsittelevissä tutkimuksissa taas nousivat esiin etenkin null-arvojen aiheuttamien virheiden käsittelemättä jättäminen, kirjoitus- ja syntaksivirheet ja tyypin tarkistuksen parissa aiheutuneet virheet.

Park ja Wiedenbeck [2011] tutkimus oli siitä mielenkiintoinen, että siinä analysointiin juuri ihmisten itsensä tukifoorumilla esittämiä kysymyksiä koskien web-kehitystä. Tutkimusasetelma on siten samantyylinen, mitä tässä tutkimuksessa, jossa tullaan analysoimaan ihmisten Stack Overflow'ssa esittämiä kysymyksiä.

3 React

React on ilmainen avoimen lähdekoodin selainpuolen JavaScript-kirjasto, joka on tarkoitettu käyttöliittymien rakentamiseen. Reactilla rakennetaan yhden sivun sovelluksia ja Reactilla rakennettuja käyttöliittymäkomponentteja voidaan uudelleen käyttää muussa koodissa. Reactia ei nähdä yksistään ohjelmistokehyksenä. Reactia käytetään yleensä yhdessä muiden kirjastojen kanssa, kuten React DOM:in kanssa. Reactia ylläpitää Meta (entinen Facebook) ja yksittäiset kehittäjät ja yritykset. [ReactJS, 2022a; MDN, 2022i]

React sisältää JSX-nimellä kutsutun syntaksilaajennuksen JavaScriptille. JSX:n avulla voidaan lisätä JavaScript-koodin sekaan HTML-muotoista koodia.

```
1.  const nimi = 'Matti Meikäläinen';
2.  const elementti = <h1>Hei, {nimi}</h1>;
```

Koodiesimerkki 9. JSX-syntaksilaajennus.

Koodiesimerkissä 9 <h1>-elementin sisään aaltosulkujen paikalle asetetaan nimi-muuttujassa oleva merkkijono. JSX:ssä aaltosulkeiden sisään voidaan asettaa mikä tahansa hyväksyttävä JavaScript-ilmaisu, kuten laskutoimitus (1 + 1), objektin arvo (tilaus.summa) tai funktiokutsu (jarjestaLista(lista)). [ReactJS, 2022b]

Uusi React-sovellus alustetaan Node.js:n ollessa asennettu tietokoneelle komennolla *npx create-react-app nimi*, jossa ”nimi” on sovellusta kuvaavan hakemiston nimi. Komento asentaa tarvittavia npm-paketteja (node package manager), kirjoittaa tarvittavat aloitusskriptit, luo perustiedostoarkkitehtuurin ja alustaa git repositoryn gitin ollessa asennettu tietokoneelle. [MDN, 2020a]

3.1 Komponentit

Reactilla on oma dokumentti-objekti-malli React DOM, joka päivittää selaimen dokumentti-objekti-mallin vastaamaan Reactin elementtejä. React DOM tarvitsee HTML-tiedostossa olevan juurielementin, johon React-elementit hahmonnetaan (render). Juurielementti voi olla esimerkiksi -elementti id:llä root. [ReactJS, 2022c]

```
1.  const element = <h1>Hei maailma</h1>;
2.  const root = ReactDOM.createRoot(
3.    document.getElementById('root')
4.  );
5.  root.render(element);
```

Koodiesimerkki 10. React DOM:in käyttöesimerkki [ReactJS, 2022c].

Koodiesimerkissä 10 hahmonnetaan Reactin <h1>-elementti React DOM:in avulla. Aluksi välitetään juurielementti ReactDOM.createRoot-funktio ja sen jälkeen hahmonnetaan render-funktio <h1>-elementti näkyväksi dokumentti-objekti-malliin. Nyt esimerkkiä X vastaavalla nettisivulla tulisi lukea teksti ”Hei maailma”. React-elementit ovat muuttumattomia. Kun elementti on luotu, sen attribuutteja tai lapsielementtejä ei voi enää muokata. Käyttöliittymää voi kuitenkin päivittää hahmontamalla uuden elementin render-funktiolla tai hyödyntämällä setState-funktioita.

React-komponenttien avulla voidaan pilkkoa käyttöliittymän osat itsenäisiin, uudelleenkäytettäviin osiin. Komponentit muistuttavat JavaScriptin funktioita; ne hyväksyvät syötteekseen sattumanvaraisia ominaisuuksia (properties, props) ja palauttavat React-elementtejä. [ReactJS, 2022]

```
1.  function Welcome(props) {
2.    return <h1>Hei, {props.name}</h1>;
3.  }
4.
5.  function App() {
6.    return (
7.      <div>
8.        <Welcome name="Sara" />
9.        <Welcome name="Cahal" />
10.       <Welcome name="Edite" />
11.     </div>
12.   );
13. }
```

Koodiesimerkki 11. React-komponetti [ReactJS, 2022d].

Koodiesimerkissä 11 Welcome-funktio on yksinkertainen esimerkki React-komponentista. Welcome-komponentti palauttaa <h1>-elementin, joka sisältää tekstin ”Hei, ” ja syötteen props-ominaisuuden name arvon. Reactissa usein käytettävällä App-komponentilla voidaan hahmontaa muita komponentteja. Koodiesimerkissä 11 App-komponentilla hahmonnetaan kolme Welcome-komponenttia name-ominaisuuksilla Sara, Cahal ja Edite. Reactin komponenttien täytyy olla niin kutsuttuja puhtaita funktioita, jotka eivät muuta ominaisuuksiensa arvoja. [ReactJS, 2022d]

3.2 Tila ja elinkaari

Tilan (state) avulla saadaan komponenteista dynaamisia, koska tilan avulla voidaan pitää kirjaa vaihtuvasta informaatiosta hahmontamisien välillä. Tila eroaa ominaisuuksista siten, että tilan sisältämä data voi muuttua ajansaatossa, kun taas ominaisuudet ovat pysyviä. Tilaa voi käyttää vain luokkakomponenteissa ja ominaisuuksia myös funktioissa. Tilan asettaa tapahtumankäsittelijät (event handlers) ja ominaisuudet asettaa komponentin vanhempi. [Robards, 2021]

```
1.  class MyClass extends React.Component {
2.    constructor(props){
3.      super(props);
4.      this.state = { attribute : "value" };
5.    }
6.  }
```

Koodiesimerkki 12. Tilan alustus rakentajalla.

Jotta tilaa voidaan käyttää, täytyy sen olla aina olemassa. Tila alustetaan käyttämällä rakentajaa (constructor), kuten koodiesimerkissä 12 on tehty. Kun halutaan päivittää tilaa, käytetään setState-funktiota, kuten koodiesimerkissä 13. Kun setState-funktiota on kutsuttu, funktio päivittää tilan ja kutsuu hahmontamisfunktiota render.

```
1.  this.setState({attribute: “changed-value”});
```

Koodiesimerkki 13. Tilan päivitys setState-funktiolla.

React-komponentin elinkaari (lifecycle) voidaan jaotella neljään osaan: alustus (initialization), asennus (mounting), päivittäminen (updating) ja irrottaminen (unmounting). Alustuksessa komponentille alustetaan tila ja ominaisuudet, kuten koodiesimerkissä 12. Asennusvaiheessa komponentit luodaan ja lisätään DOM:iin. Funktiota componentWillMount kutsutaan juuri ennen, kun komponentti lisätään DOM:iin ja funktiota componentDidMount kutsutaan lisäyksen jälkeen. Päivittämiseen käytetään funktiota shouldComponentUpdate, joka määrittää päivitetäänkö komponentti. Funktiot componentWillUpdate ja componentDidUpdate toimivat asennusvaiheen metodien tapaan, ennen ja jälkeen päivityksen. Irrottamisen yhteydessä kutsutaan funktiota componentWillUnmount, joka suoritetaan ennen komponentin poistamista DOM:sta. [Robards, 2021]

3.3 Tapahtumankäsittelijät

Reactin tapahtumakäsittelijät muistuttavat pitkälti tavallisia DOM-elementtien tapahtumakäsittelijöitä. Erona tapahtumakäsittelijät nimetään camelCase-tapaa käyttäen ja tapahtumakäsittelijänä välitetään funktio merkkijonon sijasta. Koodiesimerkissä 14 on tavallinen DOM-elementtien tapahtumankäsittelijä ja koodiesimerkissä 15 on Reactin tapahtumakäsittelijä. [ReactJS, 2022]

```
1.  <button onclick="activateLasers()">
2.    Activate Lasers
3.  </button>
```

Koodiesimerkki 14. Tavallinen JavaScript DOM-elementtien tapahtumankäsittelijä.

```
1.  <button onClick={activateLasers}>
2.    Activate Lasers
3.  </button>
```

Koodiesimerkki 15. Reactin tapahtumankäsittelijä.

Jos Reactia käyttäessä halutaan estää oletuskäyttäytyminen, täytyy kutsua erityistä preventDefault-funktiota, kun taas puhtaassa HTML:ssä voitaisiin hyödyntää *return false* -komentoa. [ReactJS, 2022e]

3.4 Hahmontaminen

Reactilla voi myös toteuttaa ehdollista hahmontamista. Ehdollisessa hahmontamisessa hahmonnetaan joukosta komponentteja vain osa kerrallaan riippuen sovelluksen tilasta. Ehdollista hahmontamista voi toteuttaa esimerkiksi tavallisilla if-else-käskyillä tai sitten lyhyemmillä JSX-ilmaisuilla. [ReactJS, 2022f]

```
1.  render() {
2.    const isLoggedIn = this.state.isLoggedIn;
3.    return (
4.      <div>
5.        {isLoggedIn
6.          ? <LogoutButton onClick={this.handleLogoutClick} />
7.          : <LoginButton onClick={this.handleLoginClick} />
8.        }
9.      </div>
10.   );
11. }
```

Koodiesimerkki 16. JSX-ilmaisulla toteutettu if-else-käsky [ReactJS, 2022f].

Koodiesimerkissä 16 on JSX-ilmaisuilla toteutettu if-else-käsky. Käsky noudattaa kaavaa *ehto ? tosi : epätosi*, eli ehdon ollessa tosi, suoritetaan tosi-sanaa vastaava komento ja ehdon ollessa epätosi, suoritetaan epätosi-sanaa vastaava komento. Koodiesimerkissä 16 käyttäjän ollessa kirjautuneena sisään näytetään kirjaudu ulos -painike ja käyttäjän ollessa kirjautuneena ulos, näytetään kirjaudu sisään -painike. [ReactJS, 2022f]

3.5 Listat ja avaimet

Kun halutaan luoda listoja Reactilla, tarvitaan avaimia, joilla yksilöidään elementit. Yksilöinnin seurauksena voidaan tunnistaa mitä elementtejä on muutettu, lisätty tai poistettu. Koodiesimerkissä 17 on funktio, joka luo listan koon verran listaelementtejä <li>. Esimerkissä on käytetty aaltosulkeisiin upotettua map-funktiota, joka jokaisen listan *numbers* alkion kohdalla luo alkiota vastaavan -elementin. Jokaiselle -elementille asetetaan avain, joka on kunkin numeron kohdalla sama numero, joka on vain muunnettu merkkijonoksi. Avaimena suositellaan käytettäväksi ominaisuuden id-arvoa, jos sellainen löytyy. Myös listan indeksi arvoa voidaan käyttää avaimena, mutta se ei ole suositeltavaa. Avaimen täytyy olla uniikki sisarelementtien kesken. [ReactJS, 2022g]

```
function NumberList(props) {
  const numbers = [1, 2, 3, 4, 5];
  return (
    <ul>
      {numbers.map((number) =>
        <li key={number.toString()}>
                  value={number}
        </li>
      )}
    </ul>
  );
}
```

Koodiesimerkki 17. -elementtien luonti map-funktion avulla Reactissa [ReactJS, 2022g].

3.6 Lomakkeet

Lomakkeita voidaan toteuttaa Rectissa myös samalla tavalla kuin perus HTML-koodissa tehtäisiin, mutta usein on kätevää toteuttaa omat JavaScript-funktiot, jotka käsittelevät lomakkeiden tietoja ja lähetystä, jotta tietoihin on helppo päästä käsiksi myöhemmin. Perustapa toteuttaa lomake Reactissa on käyttää kontrolloituja komponentteja (controlled components).

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value} onChange={this.handleChange}
/>
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Koodiesimerkki 18. Lomakkeen toteutus Reactissa [ReactJS, 2022h].

Koodiesimerkissä 18 on kontrolloituja komponentteja apuna käyttäen luotu lomake, joka kysyy nimeä. Syötekentän arvo on aina this.state.value attribuutissa tehden Reactin tilasta (state) oikean. Syötekentän arvo päivittyy koko ajan, kun käyttäjä kirjoittaa, sillä handleChange-funktio suoritetaan joka painalluksella. Reactin kontrolloimaa syötelomaketta kutsutaan kontrolloiduksi komponentiksi. Normaalisti tavallinen HTML <input>-elementti pitää yllä omaa tilaansa. [ReactJS, 2022h]

3.7 Koukut

Version 16.8. myötä Reactiin lisättiin koukut (hooks). Koukkujen avulla voidaan käyttää tilaa ja muita Reactin toiminnallisuuksia pelkän funktion sisällä ilman, että tarvitaan luokkaa (class). Koukut selkeyttävät koodia mahdollistamalla komponenttien pilkkomisen pienempiin osiin. Luokat ovat myös osittuneet hankaliksi ihmisille oppia. Esimerkiksi *this*-avainsana eroaa paljon muiden kielten vastaavista. [ReactJS, 2022i]

```
1.  import React, { useState } from 'react';
2.
3.  function Example() {
4.    // Declare a new state variable, which we'll call "count"
5.    const [count, setCount] = useState(0);
6.
7.    return (
8.      <div>
9.        <p>You clicked {count} times</p>
10.       <button onClick={() => setCount(count + 1)}>
11.         Click me
12.       </button>
13.     </div>
14.   );
15. }
```

Koodiesimerkki 19. Tilakoukun käyttö Reactissa [ReactJS, 2022j].

Koodiesimerkissä 19 on käytetty tilakoukkua (state hook). Funktio palauttaa komponentin, joka ilmaisee klikkausten määrän ja näyttää painikkeen, jolla klikkausten määrää voidaan kasvattaa. Funktion alussa luodaan useState-tyyppinen koukku, joka sisältää attribuutin *count* nykytilalle ja funktion *setCount*, jolla nykytilaa päivitetään. React säilyttää komponentin tilan uudelleenhahmontamisien välillä. Alustusargumentiksi useState-koukulle annetaan koodiesimerkissä 19 arvo 0, koska laskuri alkaa nollasta. Alustusargumenttia käytetään vain ensimmäisen hahmontamisen aikana. [ReactJS, 2022j]

```
1.  useEffect(() => {
2.      document.title = `You clicked ${count} times`;
3.    });
```

Koodiesimerkki 20. Efektikoukun käyttö Reactissa [ReactJS, 2022j].

Toinen Reactissa usein käytetty koukku on efektikoukku (effect hook). Efektikoukun avulla voidaan päivittää DOM:iin muutoksia uudelleenhahmontamisen jälkeen. Koodiesimerkissä 20 on efektikoukku, joka lisättäessä koodiesimerkin 19 koodiin päivittäisi uudelleenhahmontamisen jälkeen dokumentin välilehden otsikkoon klikkausten määrän. [ReactJS, 2022j]

Koukkuja voidaan kutsua vain ylätasolta, ei toistorakenteiden tai ehtolauseiden sisällä. Koukkuja tulee myös kutsua vain Reactin funktiokomponenttien sisältä, ei tavallista JavaScript-funktioista. Reactissa on myös mahdollista toteuttaa omia koukkuja. Omien koukkujen avulla voidaan yhdistää tila- ja efektikoukkujen toiminnallisuus yhteen koukkuun ja hyödyntää sitä useissa eri komponenteissa. Näin saadaan vältettyä toistoa koodissa. [ReactJS, 2022j; Omondi, 2022]

4 Stack Overflow ja tiedonlouhinta

Tässä luvussa esitellään Stack Overlow -sivustosta tehtyä tutkimusta, miten tietoa on louhittu kyseiseltä sivulta ja miten tietoa on analysoitu. Kohdassa 4.1 käsitellään tutkimuksia, jotka hyödynsivät LDA-menetelmää Stack Overflow -kysymysten analysoinnissa.

Stack Overflow on nettisivu, johon voi kuka tahansa lähettää kysymyksiä koskien ohjelmointiin liittyviä ongelmia [StackOverflow, 2022]. Muut käyttäjät voivat vastata kysymyksiin, keskustella ongelmista ja äänestää mielestään parhaimpia vastauksia. Kun etsitään hakukoneella ohjelmointiin liittyviin ongelmiin ratkaisuja, monesti hakutulokset osoittavat Stack Overflow -sivustolle.

4.1 LDA-menetelmä Stack Overflow -tutkimuksessa

Wang ja muut [2013] tutkivat kehittäjien vuorovaikutusta Stack Overflow -sivustolla. Yhtenä tutkimuskysymyksenä tutkijat analysoivat minkälaisia kysymyksiä kehittäjät esittävät ja mitkä ovat näiden kysymyskategorioiden jakaumat. Analysoinnissa tutkijat käyttivät LDA-menetelmää (Latent Dirichlet Allocation). Ennen LDA-menetelmän käyttöä kysymyksille suoritettiin esikäsittely, jossa kysymykset saneistettiin (tokenization), poistettiin sulkusanat (kuten sanat: minä, sinä, olla…) ja sanat stemmattiin eli perusmuotoistettiin. Koodiesimerkeistä poistettiin varatut avainsanat, kuten *if, while* ja myös sulkeet. Esikäsittelyn jälkeen tutkijat suorittivat koneellisesti LDA-menetelmällä aiheenmallinnuksen. Aiheenmallinnuksessa analysoitiin esikäsiteltyjä Stack Overflow -kysymyksiä, joista LDA-menetelmä muodostaa kysymyksistä halutun määrän aiheita. LDA-menetelmä ei luo aiheille nimeä, mutta luo listan kuvaavimmista sanoista kullekin aihekategorialle ja kertoo kategoriaan sijoittuvien kysymysten määrän. LDA-menetelmä antoi tutkijoille seuraavanlaiset kategoriat:

1. view, image, button
2. java, error, org, server
3. code, string, new, object, class
4. href, page, html, php
5. us, can, strong, like, would

Tutkijat nimesivät listan kohdan 1 sanat kategoriaksi *käyttöliittymä*, kohdan 2 kategoriaksi *pinojälki (stack trace)*, kohdan 3 kategoriaksi *suuri koodinpätkä* ja kohdan 4 kategoriaksi *web-dokumentti.* Viimeinen kohta 5 nimettiin *sekalaiseksi*. [Wang *et al.,* 2013]

Rosen ja Shihab [2015] analysoivat myös Stack Overflow kysymyksiä LDA-menetelmällä. Tutkijat keskittyivät mobiilikehitykseen liittyviin kysymyksiin. Tutkijat latasivat Stack Overflow'n tarjoaman datavedoksen (data dump), joka sisälsi kaikki siihenastiset viestit alustalla (yli 13 miljoonaa viestiä). Mobiilikehitykseen liittyvien viestien suodatukseen käytettiin sanalistaa, joka sisälsi muun muassa seuraavat sanat: iPhone, android

ja iOS sdk. Tutkijat havaitsivat, että suosituimpia kysymyksiä ovat sovellusten jakeluun, mobiilityökaluihin ja käyttöliittymäkehitykseen liittyvät kysymykset. Mobiilikysymysten havaittiin myös olevan vaikeammin vastattavia, kuin muihin aiheisiin liittyvien kysymysten Stack Overflow -alustalla. Tutkijat tutkivat myös kysymysten jakaumaa eri mobiilialustojen välillä ja kehittäjien kysymysten kohteita, jotka olivat useassa tapauksessa ohjeita, miten jokin asia tehdään, joka viittasi esimerkkien puutteeseen teknologioiden dokumentaatioissa. [Rosen ja Shihab, 2015]

Georgiou ja muut [2021] analysoivat koronavirustautiin (COVID-19) liittyviä kysymyksiä Stack Overflow -sivustolla. Tutkijat analysoivat kysymyksistä niiden olennaisimpia piirteitä ja piirteiden kehitystä ajan kuluessa, käytetyimpiä teknologioita COVID-19-sovelluksissa ja hankalimpia kehittäjien kohtaamia ongelmia. Kysymysten analysoinnissa tutkijat hyödynsivät kuvailevia tilastoja, assosiaatiosääntökaavioita, selviytymisanalyysia ja LDA-menetelmää. [Georgiou *et al.*, 2021]

Georgiou ja muut [2021] suorittivat koehakuja ja päätyivät lopulta käyttämään seuraavanlaista hakulauseketta etsiessään koronavirustautiin liittyviä kysymyksiä: ‘‘coronavirus’’ OR ‘‘covid*’’ OR ‘‘corona-virus’’ OR ‘‘sars-cov’’ OR ‘‘2019-ncov’. Tutkijat saivat kerättyä 2719 kysymystä, joista poistettiin suljetuiksi (closed) merkityt kysymykset, jonka jälkeen tutkijat kävivät manuaalisesti kaikki kysymykset läpi ja poistivat ne kysymykset, jotka eivät hakeneet apua koronavirustautiin liittyvään sovelluskehitykseen, mutta jotka muuten mainitsivat jonkin hakutermeistä. Suodatuksen jälkeen kysymyksiä oli 2213. Tutkijat käyttivät Pythonille saatavissa olevaa Selenium-kirjastoa, jolla he HTML-koodin seasta keräsivät kysymykset. Stack Overflow -kysymysten lataukseen olisi ollut jo tuolloin käytössä Stack Exhange API [Montrose, 2014]. [Georgiou *et al.*, 2021]

Georgiou ja muut [2021] tallensivat Stack Overflow -kysymyksistä id:n, otsikon, tekstivartalon, tunnisteet (tag), kysymyksen luomispäivän, kysymyksen näyttökerrat, kysymyksen äänet, vastausten lukumäärän, kommenttien määrän, tiedon sisältääkö kysymys koodinpätkän ja ensimmäinen vastauksen luomispäivämäärän. Sen jälkeen otsikoille ja tekstivartaloille suoritettiin esikäsittely, jossa muun muassa poistettiin pilkut, URL-osoitteet ja erikoismerkit. Sen jälkeen suoritettiin saneistus, stemmaus ja sulkusanojen poisto. Tekstin prosessoinnissa käytettiin Pythonille tehtyä NLTK-kirjastoa. Tutkijat analysoivat otsikon ja tekstivartalon lisäksi myös kysymyksille annettuja tunnisteita, koska ne antoivat hyvän yleiskuvan käytetyistä teknologioista liittyen koronavirustaudin parissa tehtyyn ohjelmistokehitykseen. [Georgiou *et al.*, 2021]

4.2 Muut menetelmät

Liu ja muut [2018] kehittivät MFISSO-menetelmän Stack Overflow -kysymysten etsintään. MFISSO-menetelmässä tutkijat loivat kahdeksan yläkategoriaa, jotka käyttivät nimeä facet eli viiste. Viisteet sisälsivät omia alakategorioitaan. Esimerkiksi viisteeseen *database* kuuluivat muun muassa kategoriat *MySQL, Oracle* ja *MongoDB.* Tutkijat kategorisoivat manuaalisesti 1100 kysymystä ja kehittivät näitä kategorisointeja hyödyntävän ohjelman, joka pyrkii löytämään paremmin käyttäjän hakutarpeita vastaavia kysymysvastauksia Stack Overflow'sta. Tutkijat testasivat MFISSO-menetelmää koehenkilöillä ja koehenkilöt onnistuivat ratkaisemaan esimerkkitehtävät nopeammin MFISSO-menetelmää apuna käyttäen, kuin pelkällä Stack Overflow'n omalla tekstihaulla. [Liu *et al.*, 2018]

5 Reactin ongelmat Stack Overflow -kysymysten perusteella

Tässä luvussa on tarkoitus esitellä ja toteuttaa tutkimus, jossa analysoidaan Stack Overflow -sivustolle lähetettyjä kysymyksiä koskien React-käyttöliittymäkirjastoa. Kysymyksistä on tarkoitus selvittää tarkempia ongelmakohtia Reactin sisältä, joihin kehittäjät törmäävät. Stack Overflow'sta ladataan vuoden ajalta React-aiheiset kysymykset ja kysymyksille suoritetaan esikäsittely, jonka jälkeen kysymyksiä analysoidaan LDA-menetelmällä.

5.1 Datan keräys

Stack Exhange -sivusto tarjoaa API:n, jolla voidaan ladata muun muassa Stack Overflow -sivustolta dataa, kuten kysymyksiä, vastauksia, kommentteja ja näihin liittyvää metatietoja [StackAPI, 2022]. Kysymysten lataukseen ja tallennukseen käytettiin Python-ohjelmointikieltä (versio 3.10.4 64-bit) ja Pythonille tehtyä StackAPI-wrapperia (versio 0.2.0), jolla voidaan suorittaa Pythonin kautta StackAPI:n kutsuja [Wegner, 2022].

Stack Overflow -kysymyksiin lisätään yleensä tunnisteita ja Reactiin liittyvissä kysymyksissä käytetään usein ”*reactjs”*-tunnistetta, jota aion myös käyttää Reactiin liittyvien kysymysten valinnassa. Vaikka osasta Reactiin liittyvistä kysymyksistä saattaa puuttua ”*reactjs*”-tunniste, arvioin kuitenkin tunnistemenetelmää hyödyntämällä päästävän hyviin tuloksiin. Kaikkien Stack Overflow -kysymysten läpikäynti ja niiden kysymysvartaloista sisällön analysoiminen olisi myös aikaa vievää ja kuluttaisi paljon resursseja. StackAPI tarjoaa myös hakutoiminnon, jolla voi etsiä kysymysvartaloista ja otsikoista avainsanoja. Toisaalta esimerkiksi ”*react*”-avainsanaa käyttämällä saattaisi löytyä muihin, kuin React-käyttöliittymäkirjastoon liittyviä kysymyksiä, koska ”react” tarkoittaa sanaa ”reagoida” kyseisen käyttöliittymäkirjaston lisäksi. Myös ”*reactjs”*-tunnisteta käyttäessä saattaa mukaan tulla kysymyksiä, jotka eivät oikeasti liity React-käyttöliittymäkirjastoon, mutta koska kysymyksiä ladataan tuhansia, ei muutamat poikkeustapaukset muuta suurta kuvaa.

```
1.  #Funktio hakee 'reactjs' tunisteella löytyviä Stack Overflow -kysymyksiä haluttujen
    päivämäärien väliltä
2.  def noudaKysymyksia(aloitusPvm, lopetusPvm):
3.      #Määritetään lataussivustoksi Stack Overflow
4.      SITE = StackAPI('stackoverflow')
5.
6.      #Määritetään sivun koko 100:n kysymyksen suuruiseksi (maksimi)
7.      SITE.page_size = 100
8.      #Määritetään sivujen määrä 25 (maksimi)
9.      SITE.max_pages = 25
10.
11.     #Ladataan page_size*max_pages suuruinen määrä (2500) kysymyksiä Stack Over-
    flow'sta aloituspäivämäärästä lopetuspäivämäärään olevalta väliltä. Kysymysten täy-
    tyy sisältää tunnsite 'reactjs'. Kysymykset ladataan nousevassa luontijärjestyk-
    sessä. Kysymyksen body eli vartalo sisällytetään mukaan.
12.     kysymykset = SITE.fetch('questions', fromdate=aloitusPvm, todate=lopetusPvm,
    tagged='reactjs', sort='creation', order='asc', filter='withbody')
13.
14.     #Käydään saadut kysymykset läpi ja tallennetaan ne json muodossa tiedostoon.
    Lopuksi palautetaan tieto viimeisen kysymyksen luomispäivämäärä, jotta osataan
    tehdä uusi kutsu uudella aloituspäivämäärällä.
15.     for kysymys in kysymykset["items"]:
16.         if (kysymys["creation_date"] < loppu):
17.             kysymysTiedosto = open(tallennusPolku + str(kysymys["question_id"]) +
    ".json", "w", encoding="utf-8")
18.             json.dump(kysymys, kysymysTiedosto)
19.             kysymysTiedosto.close()
20.         else:
21.             return kysymys["creation_date"]
22.     return kysymykset["items"][-1]["creation_date"]
```

Koodiesimerkki 21. Python kielellä toteutettu funktio, joka noutaa Stack Overflow-sivustolta reactjs-tunnisteella esiintyviä kysymyksiä.

Koodiesimerkissä 21 on esitelty kysymysten noutamiseen käytetty funktio. Funktio *noudaKysymyksiä(aloitusPvm, lopetusPvm)* saa parametreinaan aloitus- ja lopetuspäivämäärät, joiden väliltä funktio lataa kysymyksiä. Funktion alussa olevilla SITE-muuttujilla määritellään sivusto, johon kyselyt kohdistetaan (Stack Overflow), kysymysten määrä sivulla ja sivujen määrä. Sivun maksimikoko on 100 kysymystä ja yhdellä kyselyllä saadaan ladattua 25 sivua kerralla eli 2 500 kysymystä yhteensä. SITE.fetch-funktiolla lähetetään kysely, jolla ladataan aloitus- ja lopetuspäivämäärien väliseltä ajanjaksolta ”reactjs”-tunnisteen omaavia kysymyksiä järjestettynä nousevaan järjestykseen luomispäivän mukaan. Kyselyyn sisällytetään myös kysymyksen vartalo eli itse kysymysteksti. Saadut kysymykset tallennetaan *kysymykset*-muuttujaan, joka käydään läpi for-silmukassa, jossa jokainen kysymys tallennetaan erilliseen tiedostoon JSON-muodossa. Lopuksi funktiosta palautetaan viimeisimmän kysymyksen luontipäivämäärä, jotta osataan aloittaa seuraava funktiokutsu oikeasta ajankohdasta.

Stack Exhange API antaa suorittaa rekisteröimättömässä versiossa 25 000 API kutsua päivässä. Stack Overflowsta ladattiin vuoden ajalta aikaväliltä 1.6.2021-31.5.2022 kaikki ”reactjs”-tunnisteella esiintyvät kysymykset. Kysymysten lataus suoritettiin aika-

välillä 12.-13.6.2022. Kysymyksiä saatiin ladattua yhteensä 96 698 kappaletta. Latauk-seen meni kaksi päivää, kun käytettiin kahta eri IP-osoitetta. Kysymykset yhdistettiin lopuksi yhdeksi isoksi JSON-tiedostoksi käsittelyn nopeuttamiseksi. Koodiesimerkissä 22 on esimerkki yhdestä kysymyksestä JSON-muodossa. Osaa tiedoista ei tulla hyödyn-tämään, mutta varmuuden vuoksi niitä ei poistettu latauksen yhteydessä, jos analyysin myöhemmässä vaiheessa ylimääräisille tiedoille tulisikin käyttöä. Osa tutkimuksessa mukana olleista kysymyksistä ei enää löydy Stack Overflow'sta, koska kysymykset ovat poistuneet käyttäjän, moderoijien tai jonkin muun syyn seurauksena.

```
{'answer_count': 2,
 'body': '<p>How can I style Filebase input button? It looks very ugly and I '
         "don't know how customize it...</p>\n"
         '<p>This is what I have:</p>\n'
         '<pre><code> <FileBase\n'
         '     type="file"\n'
         '     multiple={false}\n'
         '     onDone={({ base64 }) =>\n'
         '     setFormData({ ...formData, avatar: base64 })}\n'
         ' />\n'
         '</code></pre>\n'
         '<p>I tried disabling the button like this: '
         "<code>style={{display:'none'}</code> but it didn't make any "
         'effect.</p>\n',
 'content_license': 'CC BY-SA 4.0',
 'creation_date': 1622516850,
 'is_answered': False,
 'last_activity_date': 1650647201,
 'last_edit_date': 1622516941,
 'link': 'https://stackoverflow.com/questions/67782237/react-filebase64-styling',
 'owner': {'display_name': 'mansur ischanov',
           'link': 'https://stackoverflow.com/users/12941058/mansur-ischanov',
           'profile_image': 'https://www.gravatar.com/ava-
tar/973af4cee668349ce84836f8d4bf41cf?s=256&d=identicon&r=PG&f=1',
           'reputation': 15,
           'user_id': 12941058,
           'user_type': 'registered'},
 'question_id': 67782237,
 'score': 0,
 'tags': ['javascript', 'reactjs'],
 'title': 'React FileBase64 Styling',
 'view_count': 197}
```

Koodiesimerkki 22. JSON-muotoinen Stack Overflow -kysymys kysymysvartalolla (body) [Ischanov, 2021].

5.2 Datan esikäsittely

Ennen kuin dataa alettiin analysoimaan koneellisesti, suoritettiin sille esikäsittely. Aluksi kysymysdata luettiin tiedostosta, jonka jälkeen kunkin kysymyksen tiedoista valittiin ky-symysvartalo ja kysymyksen otsikko, joka liitettiin osaksi kysymysvartaloa. Kysymys-vartaloista poistettiin <pre>- ja <code>-tunnisteiden välissä olevat osuudet, eli koodikat-

kelmat, koska niiden sisältö ei sovellu hyvin LDA-menetelmälle. Seuraavaksi kysymysvartaloista siivottiin pois HTML-tunnisteet ja muut erikoismerkit. Sitten kirjaimet muunnettiin pieniksi kirjaimiksi ja sanat saneistettiin. Sanoista poistettiin myös englanninkieliset sulkusanat, kuten ”a”, ”the”, ”is”, ”are”. Kysymysten oletettiin olevan englanninkielisiä [Atwood, 2009]. Lopuksi sanat stemmattiin ja yhden merkin pituiset sanat poistettiin listasta. Stemmauksessa käytettiin Pythonille tehtyä NLTK-kirjastoa (versio 3.7) [NLTK, 2022].

5.3 LDA-menetelmä

Luonnollisen kielen prosessoinnissa LDA-menetelmä (Latent Dirichlet Allocation) on algoritmi, jolla etsitään aiheita joukolle tekstidokumentteja. Aiheella tarkoitetaan ryhmää sanoja, jotka kuvaavat tiettyyn joukkoon kuuluvia dokumentteja. Dokumentit pyritään kategorisoimaan keskenään samoja aiheita käsittelevien dokumenttien kanssa.

Ennen kuin tekstidokumentit syötetään LDA-algoritmille, tekstidokumentit esikäsitellään. Tekstidokumenteille voidaan suorittaa muun muassa saneistus, sulkusanojen poisto ja stemmaus, kuten kohdassa 5.2 tehtiin. LDA-algoritmissa on myös useita eri hyperparametreja, joilla voidaan säätää algoritmin toimintaa. Oleellisin näistä on aiheiden lukumäärä K (koodiesimerkissä 23 nimellä *num_topics*). Mitä enemmän aiheita on, sitä yksityiskohtaisempia yksittäisistä aiheista tulee. LDA-algoritmi ei nimeä aiheita, mutta nostaa esiin aiheita parhaiten kuvaavat avainsanat ja aiheisiin sijoittuvien kysymysten määrän.

Yksi Python-ohjelmointikielelle LDA-algoritmin tarjoava kirjasto on Gensim [2022a], jota tullaan käyttämään analysoidessa Stack Overflow -sivuston kysymyksiä. Gensim-kirjasto tarjoaa myös moniydintoteutuksen LDA-algoritmista (funktio LdaMulticore), jota tullaan hyödyntämään, koska se on nopeampi, mitä vain yhdellä prosessoriytimellä toimiva versio. Koodiesimerkissä 23 on esitelty LdaMulticore-funktion mahdolliset parametrit ja niiden oletusarvot. Gensim-kirjastosta käytettiin versiota 4.2.0.

```
LdaMulticore(corpus=None, num_topics=100, id2word=None, workers=None, chunksize=2000, passes=1, batch=False, alpha='symmetric', eta=None, decay=0.5, offset=1, eval_every=10, iterations=50, gamma_threshold=0.001, random_state=None, minimum_probability=0.01, minimum_phi_value=0.01, per_word_topics=False, dtype=np.float32)
```

Koodiesimerkki 23. Python-ohjelmointikielen Gensim-kirjaston LdaMulticore-funktio ja sen oletusparametrit oletusarvoineen [Gensim, 2022a].

5.4 LDA-menetelmän hienosäätö

Latent Dirichlet Allocation -menetelmä (LDA-menetelmä) sisältää hyperparametreja, jotka voidaan nähdä koneoppimisalgoritmin asetuksina, jotka sille annetaan ennen algoritmin suoritusta. LDA-menetelmän tapauksessa niitä ovat muun muassa aiheiden lukumäärä (K), asiakirja-aihe tiheys (alfa) ja sana-aihe tiheys (beeta). Edellä mainittujen parametrien hienosäätöä kokeiltiin Kapadian [2019a] artikkelissa esitellyllä menetelmällä. Kapadian [2019a] menetelmässä kokeiltiin eri hyperparametrien *K*, *alfa* ja *beeta* yhdistelmiä LDA-algoritmilla ja vertailtiin niiden antamaa koherenssia.

Koherenssilla tarkoitetaan asioita, jotka ovat keskenään yhdenmukaisia ja aihekoherenssilla tarkoitetaan lukua, joka lasketaan yhdelle aiheelle laskemalla semanttinen samankaltaisuus aiheessa usein esiintyville sanoille [Kapadia, 2019a]. Koherenssin mittaukseen Kapadia [2019a] käytti niin kutsuttua C_v-mittaa, jonka toimintaperiaate on Kapadian [2019a] kuvauksen mukaan seuraava: C_v-mitta perustuu liukuvaan ikkunaan, suosituimpien sanojen yksijoukkoiseen segmentointiin ja epäsuoraan vahvistusmittaan, joka käyttää normalisoitua pistekohtaista keskinäistä tietoa (NPMI) ja kosinin samankaltaisuutta.

Kapadian [2019a] menetelmässä kokeiltiin alfan arvolle seuraavia arvoja: 0,01, 0,31, 0,61, 0,91, *symmetric* ja *asymmetric*. Beetan arvolle kokeiltiin arvoja: 0,01, 0,31, 0,61, 0,91 ja *symmetric*. Arvo symmetric vastaa lukua **1,0 / K**, eli esimerkiksi seitsemällä aiheella luku olisi 1,0 / 7 = 0,14. Arvo asymmetric vastaa lukua **1,0 / (aihe indeksi +** $\sqrt{K}$ **)**. Kutakin beetan arvoa kokeiltiin kunkin alfan arvon kohdalla jokaisella aiheiden lukumäärällä. Aiheiden lukumäärinä toimivat luvut väliltä 2–10.

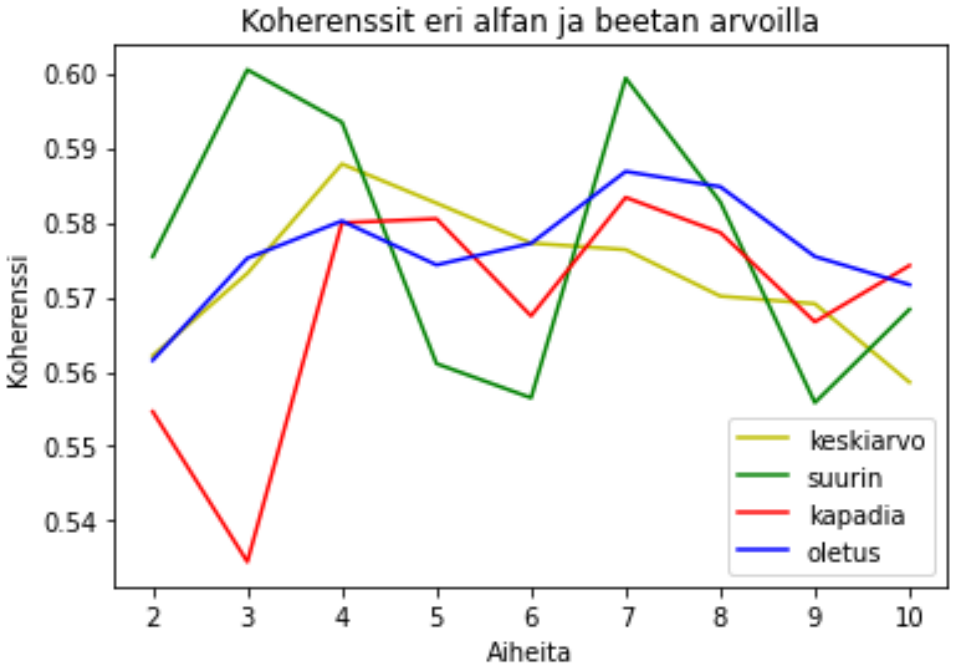
Käytin kysymysvartaloiden ja -otsikoiden muokkaamiseen (sulkusanojen poisto jne.) luvussa 4.2. käyttämiäni metodeja, enkä Kapadian [2019a] esittelemiä, koska omani hoitivat lähestulkoon saman asian. Kapadian [2019a] algoritmista käytettiin ainoastaan 100 % korpusta, jotta aikaa säästyisi.

```
ldaModel = gensim.models.LdaMulticore(corpus=corpus,
                                      id2word=dictionary,
                                      num_topics=k,
                                      random_state=100,
                                      chunksize=100,
                                      passes=10,
                                      alpha=a,
                                      eta=b)
```

Koodiesimerkki 24. Optimaalisten hyperparametrien etsinnässä käytetty LDA-funktio [Kapadia, 2019a].

Koodiesimerkissä 24 on Kapadian [2019a] käyttämä LDA-funktio, jota hyödynnettiin myös tässä tutkimuksessa. Funktiolle syötettiin kaikki mahdolliset aiemmin mainitut K:n (num_topics), alfan (alpha) ja beetan (eta) arvot. Parametri *passes* ilmaisee, montako kertaa korpus käydään läpi koulutuksen aikana, *chunksize* ilmaisee montako dokumenttia

(Stack Overflow -kysymystä) käytetään kussakin koulutuslohkossa ja *random_state* parametrin avulla voidaan luoda toistuvuutta algoritmin tulokseen – samalla arvolla pitäisi tulla suurin piirtein sama tulos uudelleen ajettaessa algoritmi samoilla parametreilla [Gensim, 2022a].



Kuva 3. Koodiesimerkin 24 algoritmin tulokset neljillä eri parametreillä.

Kuvassa 3 on havainnollistettu koodiesimerkin 24 algoritmin koherenssituloksia. Koherenssin laskemiseen LDA-mallista käytettiin Gensim-kirjaston CoherenceModel-funktiota ja koherenssiparametrissa arvoa *coherence='c_v'* [Gensim, 2022b]. Kuvassa 3 on kuvattuna koherenssit aihemäärille väliltä 2–10. Sinisellä oletus-viivalla kuvataan koherenssia LdaMulticore-funktion oletusarvoilla parametreille alfa ja beeta, joka on molempien kohdalla symmetric (beetan oletusarvo *None* tulkitaan ohjelman toimesta arvoksi symmetric). Punaisella kapadia-viivalla kuvataan Kapadian [2019a] artikkelissaan käyttämiä alfan ja beetan arvoa 0,01, jonka pohjalta Kapadia [2019a] valitsi sopivan määrän aiheille siitä kohtaa, joka antoi kyseisillä alfan ja beetan arvoilla suurimman koherenssin. Kuvan 3 tapauksessa se on siis seitsemän (7) aihetta. Lopullisiksi hyperparametreiksi alfa ja beeta Kapadia [2019a] valitsi suurimman koherenssin antaneet arvot sen aihemäärän kohdalta, joka antoi suurimman koherenssin alfan ja beetan arvoilla 0,01. Kuvan 3 aineiston pohjalta aihemäärän seitsemän kohdalla suurimman koherenssin antaneet hyperparametrit olivat alfalle arvo asymmetric ja beetalle arvo 0,91. Kuvassa 3 alfan asymmetric ja beetan 0,91 koherensseja kuvataan vihreällä suurin-viivalla. Keltaisella keskiarvo-viivalla on kuvattu eri hyperparametriyhdistelmien koherenssien yhteenlaskettu keskiarvo.

Kapadian [2019a] menetelmää hyödyntäen saatiin siis oletusarvoilla suoritettuun LDA-menetelmään verrattuna 0,021 % parempi koherenssi. Oletus hyperparametreilla saatiin koherenssiksi noin 0,587 ja suurin hyperparametreilla noin 0,599. Parannus on siis varsin pieni, varsinkin verrattaessa Kapadian [2019a] itse saamaan 17 % parannuksen omassa aineistossaan, mutta se saattaa johtua erityyppisistä dokumenteista (NIPS-konferenssin tutkimusartikkelit).

Kapadian [2019a] menetelmän tulokset eivät olleet kovin vakuuttavia omassa aineistossani. Koherenssi nousi todella vähän ja samaa tulosta ei onnistuttu toistamaan uudestaan samoilla arvoilla, myöhemmin saadut koherenssit olivat noin 0,045 yksikköä pienempiä. Hyperparametrien asymmetric ja 0,91 antamat aiheet olivat osittain varsin spesifejä ja yksittäisten avainsanojen koherenssi oli näissä usein varsin matala. Taulukossa 1 on kuvattu LDA-algoritmin tulokset koodiesimerkin 24 hyperparametreilla num_topics=7, alpha='asymmetric', eta=0.91. Tämä ei kuitenkaan ollut alkuperäinen ajokerta, koherenssi oli taulukossa 1 oli 0,551.

```
[(0,
  '0.024*"compon" + 0.021*"react" + 0.019*"use" + 0.013*"can" + '
  '0.012*"function" + 0.012*"data" + 0.012*"code" + 0.011*"tri" + 0.010*"get" '
  '+ 0.010*"work"'),
 (1,
  '0.031*"react" + 0.019*"js" + 0.018*"use" + 0.018*"error" + 0.018*"app" + '
  '0.014*"file" + 0.011*"tri" + 0.011*"work" + 0.010*"get" + 0.008*"page"'),
 (2,
  '0.017*"video" + 0.011*"socket" + 0.007*"play" + 0.007*"player" + '
  '0.006*"chat" + 0.006*"audio" + 0.005*"queri" + 0.005*"timer" + '
  '0.004*"websocket" + 0.004*"messag"'),
 (3,
  '0.017*"date" + 0.008*"00" + 0.008*"node_modul" + 0.005*"marker" + '
  '0.005*"10" + 0.005*"2022" + 0.004*"2021" + 0.004*"12" + 0.004*"intern" + '
  '0.004*"leaflet"'),
 (4,
  '0.045*"const" + 0.018*"div" + 0.018*"return" + 0.016*"type" + '
  '0.015*"import" + 0.014*"id" + 0.013*"valu" + 0.012*"name" + 0.012*"data" + '
  '0.010*"classnam"'),
 (5,
  '0.009*"movi" + 0.007*"countri" + 0.005*"amplifi" + 0.004*"canva" + '
  '0.004*"camera" + 0.002*"storybook" + 0.002*"paypal" + 0.002*"stori" + '
  '0.002*"region" + 0.001*"genr"'),
 (6,
  '0.029*"style" + 0.021*"div" + 0.017*"css" + 0.015*"color" + 0.011*"imag" + '
  '0.011*"classnam" + 0.010*"import" + 0.009*"materi" + 0.009*"icon" + '
  '0.009*"text"')]
```

Taulukko 1. LDA-algoritmin tulos seitsemällä aiheella alfan arvolla asymmetric ja beetan arvolla 0,91.

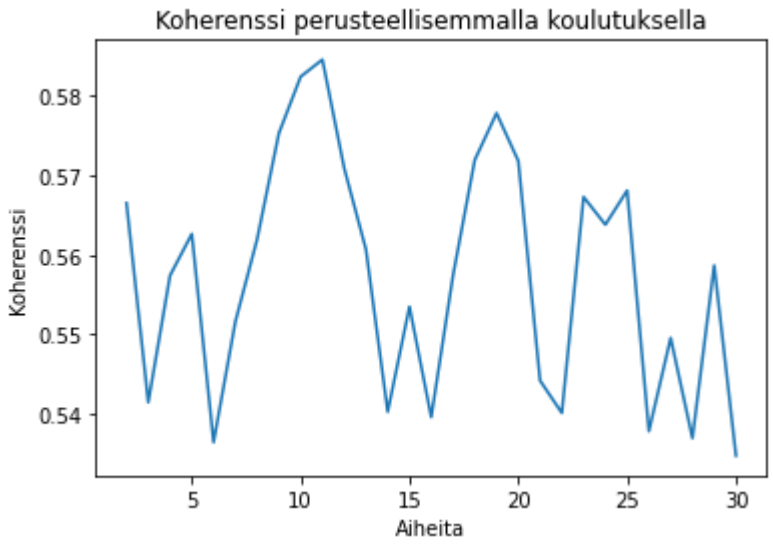
Taulukossa 1 ainakin aiheiden 2, 3 ja 5 yksittäisten avainsanojen koherenssit ovat varsin pieniä, suurimmaksi osaksi alle 0,01. Toisessa artikkelissaan Kapadia [2019b] mainitsee, että mitä suurempi arvo alfa ja beeta hyperparametreissa on, sitä spesifimpiä aiheita ja aiheiden sanoja LDA-algoritmi valikoi. Beetan arvo oli taulukon 1 tilanteessa

0,91 eli suurin testatuista vaihtoehdoista, joka selittää spesifejä avainsanoja pienellä koherenssilla.

Taulukon 1 tulokset eivät kuvanneet kattavasti aineiston aiheita, joten aineiston koherenssia päädyttiin testaamaan LDA-algoritmin hyperparametrien arvojen alfa ja beeta oletusarvoilla symmetric ja suuremmilla hyperparametrien passes, iterations ja chunksize arvoilla, sillä niiden kasvattamisen huomattiin antavan parempia tuloksia. Myös suurempaa aiheiden lukumäärää testattiin.

```
ldaModel = gensim.models.ldamulticore.LdaMulticore(corpus,
                                                   num_topics=k,
                                                   id2word = dictionary,
                                                   passes=20,
                                                   alpha='symmetric',
                                                   eta=None,
                                                   random_state=100,
                                                   chunksize=2000,
                                                   iterations=400)
```

Koodiesimerkki 25. Suurimman koherenssin antavaan aihemäärän etsinnässä käytetty LDA-funktio.



Kuva 4. Koherenssi aiheiden lukumäärillä 2–30 koodiesimerkin 25 algoritmilla.

Kuvassa 4 on koherenssit koodiesimerkin 25 algoritmilla ja aiheiden lukumäärillä 2–30. Suurin koherenssi saatiin aiheiden lukumäärän 11 kohdalla. Koherenssi oli tällöin 0,584. Koherenssin vaihteluväli oli noin 0,05 yksikköä eli ei kovin suuri.

Lopulta päädyttiin käyttämään aiheiden lukumääränä juuri 11:sta ja muita koodiesimerkissä 25 määriteltyjä hyperparametreja. Koodiesimerkin 25 algoritmi antoi suurempia koherensseja yksittäisille avainsanoille ja 11:llä aiheella saadaan tehtyä tarkempi kategorisointi eri Reactin ongelmille, kuin mitä seitsemällä aiheella olisi saatu.

5.5 Tulokset

Kun aineistolle, joka sisälsi yhteensä 96 698 kysymystä, suoritettiin LDA-analyysi koodiesimerkin 25 mukaisella funktiolla ja aihemäärällä 11, saatiin taulukon 2 mukainen tulos. Mitä suurempi avainsanan yhteydessä esiintyvä pisteytys on, sitä olennaisempi avainsana on kyseisen aiheen kannalta.

```
[(1,
  '0.052*"compon" + 0.027*"react" + 0.024*"function" + 0.024*"use" + '
  '0.019*"state" + 0.016*"render" + 0.014*"call" + 0.013*"can" + 0.011*"prop" '
  '+ 0.011*"hook"'),
 (2,
  '0.040*"react" + 0.025*"file" + 0.024*"app" + 0.023*"error" + 0.020*"js" + '
  '0.019*"use" + 0.013*"tri" + 0.011*"work" + 0.011*"run" + 0.011*"project"'),
 (3,
  '0.037*"data" + 0.029*"array" + 0.026*"object" + 0.019*"get" + 0.016*"tri" + '
  '0.015*"use" + 0.014*"react" + 0.014*"can" + 0.013*"code" + 0.013*"map"')
 (4,
  '0.022*"react" + 0.020*"use" + 0.015*"imag" + 0.014*"compon" + 0.014*"can" + '
  '0.014*"work" + 0.013*"style" + 0.012*"tri" + 0.012*"code" + 0.011*"css"'),
 (5,
  '0.029*"user" + 0.021*"api" + 0.019*"request" + 0.018*"get" + 0.018*"use" + '
  '0.016*"error" + 0.015*"post" + 0.013*"server" + 0.012*"react" + '
  '0.012*"data"'),
 (6,
  '0.040*"button" + 0.035*"select" + 0.033*"click" + 0.017*"react" + '
  '0.016*"user" + 0.015*"option" + 0.013*"can" + 0.013*"date" + 0.013*"want" + '
  '0.012*"modal"'),
 (7,
  '0.069*"const" + 0.032*"return" + 0.029*"data" + 0.023*"consol" + '
  '0.023*"log" + 0.019*"import" + 0.018*"state" + 0.018*"usest" + 0.017*"true" '
  '+ 0.016*"fals"'),
 (8,
  '0.060*"div" + 0.034*"classnam" + 0.025*"import" + 0.024*"const" + '
  '0.016*"style" + 0.013*"id" + 0.012*"name" + 0.012*"button" + 0.012*"react" '
  '+ 0.011*"item"'),
 (9,
  '0.077*"page" + 0.046*"js" + 0.037*"rout" + 0.036*"react" + 0.030*"compon" + '
  '0.025*"router" + 0.022*"app" + 0.020*"link" + 0.019*"navig" + 0.015*"url"'),
 (10,
  '0.051*"input" + 0.049*"form" + 0.048*"valu" + 0.033*"type" + 0.028*"field" '
  '+ 0.018*"react" + 0.013*"submit" + 0.012*"use" + 0.012*"string" + '
  '0.011*"valid"'),
 (11,
  '0.026*"node_modul" + 0.019*"10" + 0.018*"00" + 0.015*"12" + 0.012*"11" + '
  '0.011*"15" + 0.011*"17" + 0.011*"lib" + 0.011*"20" + 0.010*"16"')]
```

Taulukko 2. LDA-algoritmin tulokset Stack Overflow -sivuston kysymyksistä. Tulokset on järjestetty aiheen kattavuuden mukaan laskevaan järjestykseen. Indeksointi alkaa numerosta 1.

Taulukossa 3 on kuvattu siistimmässä muodossa taulukossa 2 ilmoitetut LDA-algoritmin antamat tulokset koskien tutkittuja React-aiheisia Stack Overflow -kysymyksiä. Taulukossa 3 on myös annettu aiheille niitä kuvaavat nimet.

Taulukossa 3 useamman aiheen kohdalla avainsanat *react, js* ja *use* toistuvat. Tämä on sikäli oletettavaa, sillä kysymykset käsittelevät Reactia, joka on toteutettu JavaScript-kielellä (lyhennettynä js). Myös sana use eli käyttää toistuu useassa aiheessa, mutta ei

sinällään onnistu kuvaamaan aiheita hyvin, sillä voidaan olettaa, että kysymyksissä puhutaan erilaisten menetelmien käyttämisestä. Avainsanat esiintyvät myös niiden stemmatuissa muodoissa, esimerkiksi avainsana *compon* viittaa sanaan *component* eli komponentti. Avainsana *tri* viittaa sanaan *tried,* perusmuodossa *try,* eli yrittää. Kysyjä on siis luultavasti kertoneensa yrittäneen tehdä jotain. Tämä sana toistui useammassa aiheessa.

Koska LDA-menetelmä on probabilistinen, täysin saman tuloksen saaminen samalla datalla ja parametreilla ei ole täysin varmaa, vaikka käytettäisiin samaa random_state-parametriä. Ajettaessa algoritmia useampia kertoja, avainsanojen järjestys hieman vaihteli aiheiden sisällä, mutta aiheiden prosenttiosuudet pysyivät samoina.

Nro	Aiheen nimi	Avainsanat	Aiheen % osuus
1	Komponenttien hahmontaminen	compon, react, function, use, state, render, call, can, prop, hook	18,5
2	Virheet projekteissa	react, file, app, error, js, use, tri, work, run, project	15,2
3	Data ja taulukot	data, array, object, get, tri, use, react, can, code, map	12,6
4	Kuvat	react, use, imag, compon, can, work, style, tri, code, css	11,5
5	Ohjelmointirajapinnat ja palvelinpuoli	user, api, request, get, use, error, post, server, react, data	10,6
6	Komponentit	button, select, click, react, user, option, can, date, want, modal	7,3
7	Tila	const, return, data, consol, log, import, state, usest, true, fals	6,3
8	Käyttöliittymän ulkoasu	div, classnam, import, const, style, id, name, button, react, item	5,8
9	Reititys	page, js, rout, react, compon, router, app, link, navig, url	5,4
10	Lomakkeet	input, form, valu, type, field, react, submit, use, string, valid	4,7
11	Node-moduulit	node_modul, 10, 00, 12, 11, 15, 17, lib, 20, 16	2,0

Taulukko 3. LDA-algoritmin tulokset nimetyillä aiheilla ja niiden prosenttiosuuksilla kaikista kysymyksistä.

Taulukon 3 aihe 1 *komponenttien hahmontaminen* sisältää komponenttien hahmontamisessa kohdattuja ongelmia ja oli aiheista yleisin 18,5 % osuudella. Avainsanat *compon* ja *render* eli komponentti ja hahmontaa viittaavat suoraan aiheeseen. Avainsanat *state* ja *hook* eli tila ja koukku liittyvät komponenttien tilan päivittämiseen. Avainsana *prop* eli ominaisuus ovat komponenteille välitettävää tietoa.

Taulukon 3 aihe 2 *virheet projekteissa* sisältää erilaisia virheitä, joihin on törmätty suorittaessa projekteja. Projektiin viittaavia avainsanoja ovat *file, app* ja *project* eli tiedosto, sovellus ja projekti. Avainsanat *error* ja *run* eli virhe ja suorittaa viittaavat suorituksen aikana tapahtuneisiin virheisiin. Myös avainsana *work* eli toimia viittaa mahdollisesti epätoivuttuun toimintaan sanan esiintyessä kieltävän sanan kanssa.

Taulukon 3 aihe 3 *data ja taulukot* sisältää datan käsittelyyn ja taulukoihin liittyviä kysymyksiä. Avainsanat *data* ja *array* (taulukko) esiintyvät monissa aiheen kysymyksissä. Avainsana *map* viittaa samannimiseen funktioon, jolla voidaan suorittaa jokaiselle taulukon alkiolle tietty operaatio.

Taulukon 3 aihe 4 *kuvat* sisältää kuviin ja ulkoasuun liittyviä kysymyksiä. Avainsana *imag* viittaa sanaan *image,* joka tarkoittaa kuvaa. Avainsanat *style* ja *css* eli tyyli ja CSS, jolla voidaan määritellä HTML-sivun tyylejä, viittaavat kuvien tapaan sivun visuaaliseen puoleen.

Taulukon 3 aihe 5 *ohjelmointirajapinnat ja palvelinpuoli* sisältää kysymyksiä liittyen ohjelmointirajapintoihin ja muihin palvelinpuolella käytettäviin menetelmiin. Avainsana *api* (application user interface) tarkoittaa ohjelmointirajapintaa. Muita ohjelmointirajapintojen ja palvelinpuolen kanssa usein käytettyjä avainsanoja olivat *request, get, post* ja *server* eli kysely, vastaanottaa, lähettää ja palvelin.

Taulukon 3 aihe 6 *komponentit* sisältää komponentteihin liittyviä kysymyksiä, yleisesti siihen miten tietynlainen komponentti voidaan toteuttaa tietyillä ominaisuuksilla. Avainsanat *button* ja *modal* eli painike ja modaalinen ikkuna (ikkuna, joka ilmestyy sivun pääikkunan päälle jättäen sen näkyviin alle, mutta estäen sen käytön, kunnes modaalinen ikkuna suljetaan) ovat Reactissa esiintyviä komponentteja. Avainsanat *select, click* ja *option* eli valita, klikata ja vaihtoehto viittaavat taas vuorovaikutukseen komponenttien kanssa. Avainsana *date* eli päivämäärä voi esiintyä päivämäärän valintaan liittyvässä komponentissa.

Taulukon 3 aihe 7 *tila* sisältää Reactin tila-ominaisuuteen liittyviä kysymyksiä. Avainsana *state* eli tila viitaa suoraan aiheeseen. Avainsanat *consol (console)* ja *log* eli konsoli ja kirjata ylös viittaavat mahdollisesti virheelliseen ohjelman tulosteeseen. JavaScriptissä on myös funktio nimeltä *console.log*, jolla voidaan tulostaa konsoliin.

Taulukon 3 aihe 8 *käyttöliittymän ulkoasu* sisältää nimensä mukaisesti käyttöliittymän ulkoasua koskevia kysymyksiä. Tähän viittaavat ulkoasuun viittaavat avainsanat

style, name, button ja div eli tyyli, nimi, painike ja div (-elementti). Komponenttikirjasto *Material UI (MUI)* esiintyi myös osassa kysymyksistä ja avainsana *mui* oli myös aiheen 8 avainsanojen joukossa, mutta ei suosituimman 10 sanan joukossa.

Taulukon 3 aihe 9 *reititys* sisältää reititykseen liittyviä kysymyksiä. Reititykseen viittaavia avainsanoja ovat *rout, router, link, navig* ja *url* eli reititin (kaksi kertaa), linkki, navigointi ja URL. Reactiin voi asentaa lisäosan, React Router, jolla sivujen linkitys toisiinsa onnistuu. Aiheeseen parhaiten sopivia kysymyksiä tarkastellessa, moni vaikuttaa myös koskevan juuri React Routerin parissa kohdattuja ongelmia.

Taulukon 3 aihe 10 *lomakkeet* sisältää lomakkeisiin liittyviä kysymyksiä. Avainsana *form* tarkoittaa lomaketta ja avainsanat *input, valu, field* ja *sumbit* eli syöte, arvo, kenttä ja lähettää liittyvät kaikki lomakkeisiin. Myös avainsanat *type* ja *string* eli tyyppi ja merkkijono voivat liittyä lomakkeisiin (esimerkiksi lomakekentän tyyppi ja merkkijonoja hyväksyvä kenttä).

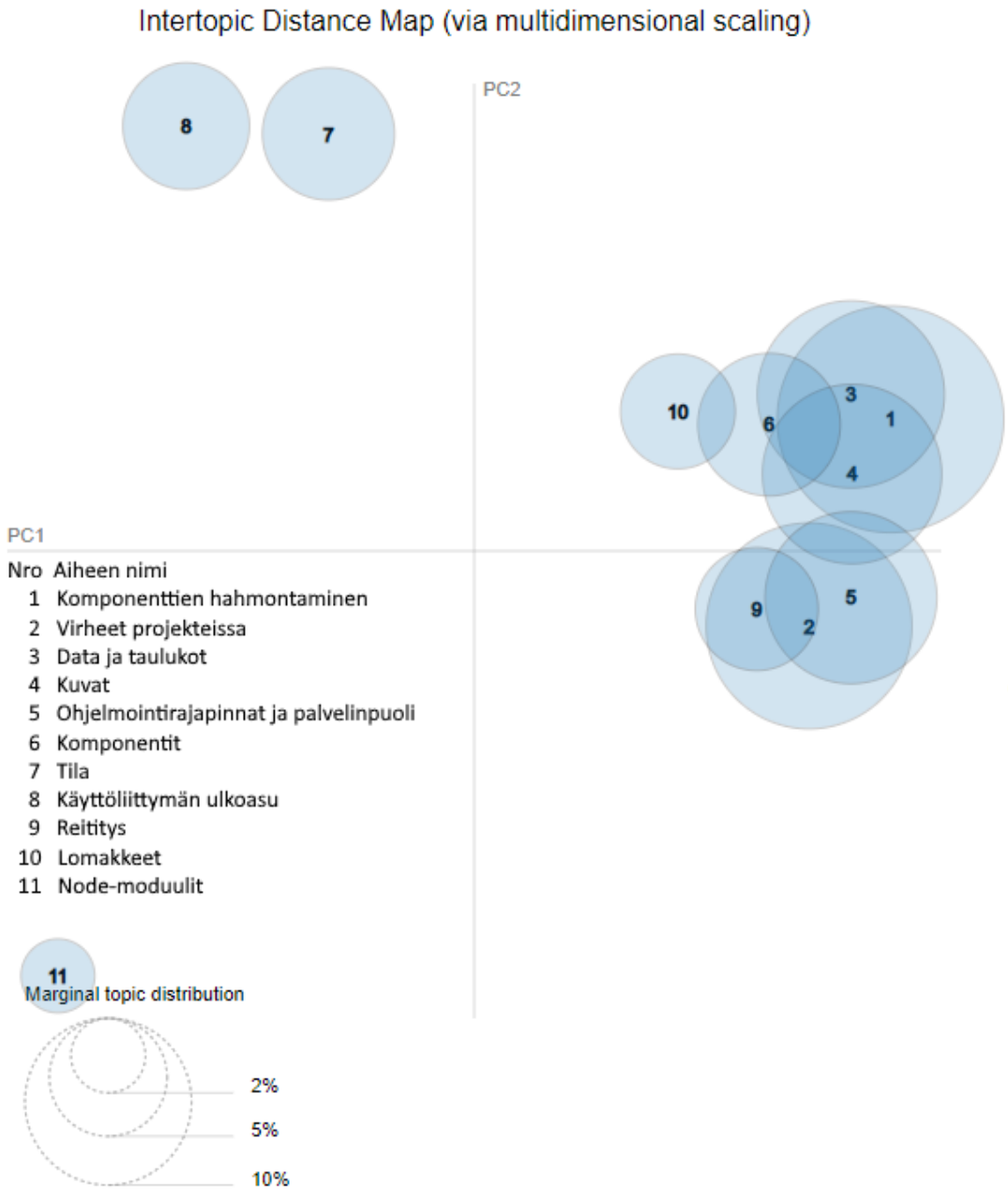
Taulukon 3 aihe 11 *node-moduulit* on hieman epämääräinen, mutta monessa aiheen kysymyksessä esiintyy erilaisia Node-moduulien versionumeroita ja päivämääriä. Osa kysymyksistä käsitteli SVG-kuvia. Kysymyksessä saattoi esiintyä esimerkiksi seuraavanlainen päivämääräesitys:

```
2021-12-31T00:00:00.000Z, 2022-01-12T03:16:30.075Z, 6
```

Tekstin esikäsittelyssä päivämääristä poistettiin viivat ja kaksoispisteet, jolloin numeroista tuli yksittäisiä numeroita. Sama tehtiin erilaisille versionumeroinneille, esimerkiksi versionumerosta 1.19.0 poistettiin pisteet, jolloin siitä tuli kolme eri merkkijonoa.

5.6 Analyysi ja visualisointi

Kuvassa 5 on aiheiden välinen etäisyyskartta, jossa havainnollistetaan taulukossa 3 esiintyneiden aiheiden samankaltaisuutta ja suuruutta. Kuva 5 on toteutettu Pythonille saatavilla olevalla pyLDAvis-kirjaston versiolla 3.3.1 [Mabey, 2022]. Ympyröiden koko kuvastaa kuinka suuri osa kysymyksistä kuului kyseiseen aiheeseen. Aiheen numero on ympyrän keskellä. Mitä lähempänä ympyrät ovat toisiaan, sitä samanlaisempia aiheet ovat. Suurin osa aiheista on melko samanlaisia, suurin osa aiheista sijoittuu kuvan 5 oikeaan reunaan. Kuvan 5 oikeassa reunassa aiheiden 1 ja 2 ympärillä on selvät rykelmät, joissa esiintyy päällekkäisyyttä. Aiheen 1 kanssa samanlaisia ovat aiheet 3, 4 ja 6 ja aiheen 2 kanssa samanlaisia ovat aiheet 5 ja 9. Pienemmällä aihemäärällä (K:n arvolla) nämä voisivat olla kaksi isoa aihetta. Aihe 10 on jo hieman kauempana aiheen 1 rykelmästä. Aiheet 7 ja 8 ovat selvästi muista aiheista poikkeavia sanastonsa puolesta, mutta ovat keskenään melko samanlaisia. Aihe 11 taas eroaa selvästi muista aiheista ja on suuruudeltaan pieni.



Kuva 5. Aiheiden päällekkäisyys ja etäisyys toisistaan.

Kun taulukossa 3 esiteltyjä aiheita katsotaan kokonaisuutena, nousee suurimpana ongelmakohtana esiin komponentteihin liittyvät ongelmat. Selvästi komponentteihin liittyviä aiheita ovat aiheet 1, 3, 4, 6 ja 10. Kaaviosta 3 nähdään myös näiden aiheiden muodostavan oman rykelmänsä. Nämä edellä mainitut aiheet kattavat 54,6 % kaikista kysymyksistä. Tämä on sikäli oletettava tulos, sillä Reactilla pääasiassa luodaan erilaisia komponentteja, jotka muodostavat nettisivujen käyttöliittymään erilaisia elementtejä. Aihe 1 komponenttien hahmontaminen on aiheista suurin ja käsittelee komponentteja ja niihin liittyvää hahmontamista ja tilaa. Aihe 3 data ja taulukot on hyvin samanlainen aiheen 1 kanssa, kuten kaaviosta 3 nähdään, mutta erikoisuutena on taulukot ja niiden hallinta.

Esimerkiksi aiheessa 3 esiintyy avainsana map, jolla tarkoitetaan map-funktiota, jota käytetään usein Reactissa listakomponenttien luonnissa. Aiheen 4 kuvat erikoisuus on kuviin liittyvät komponenttiongelmat, mutta aiheen 4 avainsanoista löytyy myös avainsana compon, mikä korostaa yhteyttä komponentteihin. Aihe 6 komponentit käsittelee suoraan komponentteja, niiden toimintaa ja miten tietyt komponentit saadaan toimimaan tietyllä tavalla. Vaikka aihe 10 lomakkeet on kaaviossa 3 jo hieman enemmän erillään komponentteihin liittyvien kysymysten joukosta, on sen sisältö silti selkeästi komponentteihin liittyvää, tarkemmin ottaen lomakkeisiin ja niiden toimintaan.

Toiseksi suurimman kokonaisuuden muodostavat aiheet 2, 5 ja 9. Nämä aiheet muodostavat kuvassa 5 selvän rykelmän ja kattavat 31,2 % kaikista kysymyksistä. Kysymykset liittyvät enemmän palvelinpuolen asioihin ja virheisiin. Aiheen 5 ohjelmointirajapinnat ja palvelinpuoli kysymykset kattavat vuorovaikutusta palvelimien kanssa, esimerkiksi jonkin internetpalvelun ohjelmointirajapinnan kanssa vuorovaikutusta. Aiheen 9 reititys kysymykset käsittelevät navigointia sivujen välillä, joka osaltaan sivuaa komponentteja, sillä React Router on myös komponentti. Aihe 2 virheet projekteissa on painottunut yleisesti virheisiin projekteissa, virheilmoituksiin ja niiden syyn selvittämiseen, eikä erityisesti keskity suoraan komponenttien parissa törmättyihin virheisiin.

Aiheet 7 tila ja 8 käyttöliittymän ulkoasu olivat kuvassa 5 lähellä toisiaan, mutta eivät kuitenkaan limittäin. Kun taulukosta 3 katsotaan aiheiden avainsanoja, ovat aiheet selvästi kuitenkin erillisiä. Aiheen 7 kysymykset käsittelevät tilaa ja kysymyksissä toistuvat tilaan liittyvät toiminnot, kuten useState, useEffect, Hooks ja myös JavaScriptissä tilanhallintaan käytetty Redux-kirjasto, josta on tehty myös Reactille oma versionsa. Aiheen 8 kysymykset taas painottuvat enemmän Reactin visuaaliseen puoleen, siihen miltä komponentit näyttävät. Kysymyksissä esiintyy myös Material UI -komponenttikirjasto, joka myös vaikuttaisi olevan suosituin Reactille löytyvä erillinen komponenttien ulkoasun muokkaukseen tarkoitettu kirjasto. Aiheet 7 ja 8 kattavat yhteensä 12,1 % kaikista kysymyksistä.

Aihe 11 Node-moduulit erosi muista aiheista huomattavasti. Kuvassa 5 aihe 11 sijoittuu vasempaan alareunaan kauaksi muista aiheista. Kysymyksissä esiintyi paljon numeroita, esimerkiksi Node-moduulien versionumeroiden muodossa ja päivämäärinä. Kysymykset käsittelivät virheitä erilaisten Node-moduulien ja päivämääriin liittyvien operaatioiden kanssa. Kysymyksissä esiintyi myös SVG-kuviin liittyviä ongelmia.

5.7 Tulosten yleistettävyys ja peilaus aiempaan kirjallisuuteen

Kohdassa 2.4 esitellyissä tutkimuksissa virheiden ja haasteiden kategorisointi oli usein yksityiskohtaisempaa, kuin tämän tutkielman tutkimuksessa, jossa saadut kategoriat olivat laveampia. Kohdan 2.4 tutkimuksissa oli myös selvitetty tarkemmin ongelmien syy, kun taas tässä tutkimuksessa perehdyttiin siihen, mitä kysytään. Kohdan 4.1 Wang ja

muut [2013] tutkimuksessa tutkittiin koko Stack Overflow -sivuston kysymysmassaa muun muassa LDA-menetelmällä, mutta koska kysymysjoukko oli eri aiheista, ei sitä voida suoraan verrata tämän tutkimuksen tuloksiin.

Kuitenkin Park ja Wiedenbeckin [2011] tutkimuksen HTML-, CSS- ja JavaScript-kieliin liittyvien kysymysten analysoinnin tulokset olivat jokseenkin tähän tutkimukseen verrattavassa muodossa. Park ja Wiedenbeckin [2011] tutkimuksen kehitys-kategoria käsitteli muun muassa linkkien luontia, kuvia ja taulukoita ja kysymykset kattoivat 34,3 % kaikista kysymyksistä. Tämän tutkimuksen aiheet data ja taulukot, kuvat ja reititys (reititys-aihe sisälsi linkkeihin liittyviä kysymyksiä) sisältävät saman tyylisiä kysymyksiä koskien Reactia ja aiheiden yhteenlaskettu osuus oli 29,5 % eli suurin piirtein samaa suuruusluokkaa, kuin Park ja Wiedenbeckin [2011] tutkimuksessa. Park ja Wiedenbeckin [2011] tutkimuksen teknologia-kategoria käsittelivät ohjelmoinnin ulkopuolisia teknologisia ongelmia, kuten shell-käyttäjien aktivointia ja FTP-ohjelmien (File Transfer Protocol) konfigurointia ja kattoivat 24,5 % kaikista kysymyksistä. Tässä tutkimuksessa ei löytynyt vastaavaa kategoriaa kysymyksille, mutta aihevirheet projekteissa (15,2 %) saattaa sisältää hieman samantyylisiä kysymyksiä, kuten React-sovellusten ylläpitäminen palvelimella. Park ja Wiedenbeckin [2011] tutkimuksen suunnittelu-kategoria käsitteli vinkkejä nettisivujen suunnitteluun liittyen ja kattoi 4,7 % kaikista kysymyksistä. Tässä tutkimuksessa useammatkin aiheet saattavat sisältää juuri suunnitteluvinkkejä kysyviä kysymyksiä, mutta parhaiten niitä saattaa löytää aiheesta käyttöliittymän ulkoasu (5,8 %).

Kohdassa 4.1 esitellyssä Wang ja muut [2013] tutkimuksessa tutkijat tutkivat Stack Overflow -sivuston kaikkia kysymyksiä ja nimesivät yhden LDA-menetelmän antaman kategorian käyttöliittymä-nimiseksi. Käyttöliittymä-kategoria sisälsi ainakin seuraavat avainsanat: view, image ja button. Tässä tutkimuksessa Wang ja muiden [2013] käyttöliittymä-kategoriaa vastaavat parhaiten aiheet käyttöliittymän ulkoasu (5,8 %) ja kuvat (11,5 %). Wang ja muut [2013] tutkimuksessa käyttöliittymä-kategoria sisälsi 10 % kaikista kysymyksistä. Wang ja muiden [2013] tutkimuksen pinojälki-kategoria (stack trace) sisälsi avainsanat java, error, org ja server. Pinojälki-kategoriaa (11 %) tässä tutkimuksessa vastaa hieman virheet projekteissa kategoria (15,2 %). On kuitenkin hyvä muistaa, että tässä tutkimuksessa ja Wang ja muiden [2013] tutkimuksessa oli eri kysymysjoukot kyseessä, joten suuria yleistyksiä tulosten vertailun pohjalta ei voida tehdä.

Tässä tutkimuksessa saaduista havainnoista voi myös saada viitteitä muiden Reactin kanssa samankaltaisten käyttöliittymäkirjastojen – kuten Vue – ongelmista. Tuloksia ei voi kuitenkaan täysin yleistää, sillä eri kirjastoilla on omanlaisensa tavat toteuttaa eri ominaisuudet, joten tietyn kirjaston tapa tehdä tietty asia saattaa olla haastavampi, kuin toisen. Esimerkiksi tämän tutkimuksen taulukon 3 aiheista data ja taulukot, kuvat ja lomakkeet luultavasti toistuvat myös muiden kirjastojen kohdalla, sillä ne ovat vakituinen osa web-kehitystä. Kysymysten aiheiden prosenttiosuudet saattavat kuitenkin vaihdella eri

kirjastojen välillä, johtuen vaihtelevasta haasteellisuudesta toteuttaa tietty ominaisuus, kuten lomake.

6 Yhteenveto

Web-kehitys on tämän hetken suosituimpia ohjelmoinnin suuntauksia ja kuten muissakin ohjelmoinnin haaroissa on web-kehityksessäkin omat haasteensa. Tässä tutkimuksessa perehdyttiin aluksi web-kehityksen perusteisiin, jonka jälkeen tarkasteltiin aiempaa tutkimusta web-kehityksen haasteista. Tutkimusosuudessa tutkittiin React-käyttöliittymäkirjaston käytössä ilmenneitä haasteita.

Stack Overflow kysymys-vastaus-sivustolta ladattiin vuoden ajalta kaikki reactjs-tunnisteella esiintyvä kysymykset, joita oli 96 698 kappaletta lataushetkellä. Kysymyksille suoritettiin muun muassa sulkusanojen poisto, koodikatkelmien poisto ja stemmaus. Tämän jälkeen kysymyksiä analysoitiin LDA-menetelmällä. Aluksi Gensim-kirjaston LDA-algoritmin hyperparametreille suoritettiin hienosäätöä, jossa pyrittiin löytämään suurimman koherenssin antavat hyperparametrit. Lopulta päädyttiin käyttämään oletusarvoja hyperparametreille alfa ja beeta, jolloin saatiin suurin koherenssi aihemäärälle 11. Lopuksi siis suoritettiin LDA-analyysi 11:sta aiheella, joita päädyttiin käyttämään analyysissa.

LDA-analyysilla saadut 11 aihetta nimettiin manuaalisesti avainsanojen ja aiheeseen kuuluvien dokumenttien perusteella. Aiheiden kattavuudelle saatiin myös prosenttiosuudet koko kysymysmassasta. Aiheiden kokoa ja etäisyyttä toisistaan havainnollistettiin myös visuaalisesti. Saadut aiheet ja niiden prosenttiosuudet olivat seuraavat: komponenttien hahmontaminen (18,5), virheet projekteissa (15,2), data ja taulukot (12,6), kuvat (11,5), ohjelmointirajapinnat ja palvelinpuoli (10,6), komponentit (7,3), tila (6,3), käyttöliittymän ulkoasu (5,8), reititys (5,4), lomakkeet (4,7) ja node-moduulit (2,0).

Suurin osa (54,6 %) Reactin ongelmista koskivat suoraan eri komponentteja. Noin kolmannes (31,2 %) liittyi palvelinpuoleen ja yleisiin virheisiin projekteissa. Noin kymmenesosa (12,1 %) kysymyksistä käsitteli tilaa ja käyttöliittymän ulkoasua. Pienin ja muista etäisin aihe oli nimeltään node-moduulit 2,0 % osuudella.

Tutkimustulosten antama 54,6 % osuus komponenteille oli sikäli odotettava lopputulos, että komponentit ovat juuri Reactin pääelementti, uudelleen käytettävien komponenttien luonti. Myös palvelinpuolen esiintyminen kysymyksissä oli odotettavaa, sillä Reactilla ohjelmoitaessa sisällytetään ohjelmiin usein palvelinpuolen toiminnallisuutta, kuten ladattaessa tietoa tietokannasta.

Avain samanlaista LDA-menetelmällä käyttöliittymäkirjastosta esitettyjä kysymyksiä analysoivaa tutkimusta ei aiemmin ole tehty. Esiteltyjen tutkimusten tuloksissa oli kuitenkin pienimuotoisia yhteneväisyyksiä tähän tutkimukseen, mutta aiempien tutkimusten analysoitava aineisto oli kohtalaisen erilainen, joten suuria yleistyksiä ei voida kuitenkaan tehdä.

Tutkimustuloksia on mahdollista hyödyntää Reactin opetuksessa. Aiheiden prosenttiosuuksien pohjalta voisi tehdä painotuksia aiheisiin, jota opetuksessa käsitellään. Esimerkiksi aihe data ja taulukot oli kolmanneksi suurin aihe, joten opetuksessa olisi hyvä käyttää tarpeeksi aikaa siihen, miten dataa käsitellään React-komponenteissa. Reactin kehityksessä voi myös ottaa huomioon tämän tutkimuksen tulokset ja yrittää selkeyttää haasteita tuottaneita aiheita Reactin dokumentaatiossa, virheilmoituksissa ja teknisellä tasolla.

Tutkimukseen valittu metodi – LDA-analyysi Stack Overflow -kysymyksistä – ei antanut erityisen yksityiskohtaisia ja tarkkoja tuloksia Reactin ongelmista. Manuaalisella kysymysten analysoinnilla olisi voinut saada tarkempia tuloksia ongelmista. LDA-analyysia ja manuaalista kysymysten läpikäyntiä olisi voinut myös yhdistää. Tällöin tuloksista olisi saatu tarkempia ja LDA-analyysin tarkkuudesta olisi myös saatu tietoa.

Mahdollisessa jatkotutkimuksessa voisi tutkia kysymyksiä uutena ajankohtana, sillä React ja web-kehityksen menetelmät kehittyvät nopeaa vauhtia – uusia teknologioita ilmestyy ja vanhat jäävät vähemmällä käytölle. Kysymyksiä voisi myös koittaa manuaalisesti kategorisoida, sillä koneellisessa analyysissa tapahtuu helpommin virheanalyyseja, mutta toisaalta volyymi voi olla suurempi. Stack Overflow’ssa käytettyjä tunnisteita voisi myös analysoida, mitä muita tunnisteita käytettiin reactjs-tunnisteen kannssa ja näistä sitten pyrkiä tekemään päätelmiä. Stack Overflow’n kysymysten vastauksista voisi analysoida ja katsoa antavatko ne parempia kuvauksia ongelmista, sillä niissä voi olettaa todennäköisemmin olevan havaittu, kun taas itse kysymyksessä se saattaa olla vielä epäselvä.

7 Viiteluettelo

Abraham Arnold. 2022. 6 Ways to Declare Functions in JavaScript Functions and 1 to Beat Them All. Frontend Weekly. https://medium.com/front-end-weekly/6-ways-to-declare-functions-in-javascript-functions-and-1-to-beat-them-all-66003b6350c0 (Viitattu 22.9.2022)

Atwood Jeff. 2009. Non-English Question Policy. Stack Overflow Blog. URL https://stackoverflow.blog/2009/07/23/non-english-question-policy/ (Viitattu 9.10.2022)

ECMA. 2021. ECMAScript 2021 language specification (ECMA-262). URL https://www.ecma-international.org/publications-and-standards/standards/ecma-262/ (Viitattu 12.3.2022)

Gao Zheng, Bird Christian, and Barr Earl T. 2017. To Type or Not to Type: Quantifying Detectable Bugs in JavaScript. In 2017 *IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, IEEE, 758–769. DOI https://doi.org/10.1109/ICSE.2017.75

Gensim. 2022a. LDA multicore algorithm documentation. URL https://radimrehurek.com/gensim/models/ldamulticore.html (Viitattu 3.9.2022)

Gensim. 2022b. Coherence model algorithm documentation. URL https://radimrehurek.com/gensim/models/coherencemodel.html (Viitattu 9.3.2022)

Georgiou Konstantinos, Mittas Nikolaos, Chatzigeorgiou Alexandros, and Angelis Lefteris. 2021. An empirical study of COVID-19 related posts on Stack Overflow: Topics and technologies. *The Journal of systems and software* 182, (2021), 111089–111089. DOI https://doi.org/10.1016/j.jss.2021.111089

Gyimesi Péter, Vancsics Béla, Stocco Andrea, Mazinanian Davood, Beszédes Árpád, Ferenc Rudolf, and Mesbah Ali. 2021. BUGSJS: a benchmark and taxonomy of JavaScript bugs. *Software testing, verification & reliability* 31, 4 (2021). DOI https://doi.org/10.1002/stvr.1751

Hanam Quinn, De M Brito Fernando S., and Mesbah Ali. 2016. Discovering bug patterns in Javascript. In Proceedings of the *ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 144–156.

Ischanov Mansur. 2021. Stack Overflow question - React FileBase64 Styling. https://stackoverflow.com/q/67782237/7257614 (Viitattu 21.6.2022)

ISO. 2018. ECMAScript Specification Suite (ISO/IEC 22275:2018). URL https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/30/73002.html (Viitattu 12.3.2022)

Kapadia Shashank. 2019a. Evaluate Topic Models: Latent Dirichlet Allocation (LDA). Medium. https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0 (Viitattu 22.6.2022)

Kapadia Shashank. 2019b. Topic Modeling in Python: Latent Dirichlet Allocation (LDA). Medium. URL https://towardsdatascience.com/end-to-end-topic-modeling-in-python-latent-dirichlet-allocation-lda-35ce4ed6b3e0 (Viitattu 3.9.2022)

Liu Mingwei, Peng Xin, Jiang Qingtao, Marcus Andrian, Yang Junwen, and Zhao Wenyun. 2018. Searching StackOverflow Questions with Multi-Faceted Categorization. In *(Internetware '18)*, ACM, 1–10. DOI https://doi.org/10.1145/3275219.3275227

Mabey Ben. 2022. pyLDAvis - Python library for interactive topic model visualization. URL https://github.com/bmabey/pyLDAvis (Viitattu 16.9.2022)

MDN. 2022a. HTML basics - Learn web development. URL https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics (Viitattu 27.2.2022)

MDN. 2022b. CSS basics - Learn web development. URL https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics (Viitattu 2.3.2022)

MDN. 2022c. Express/Node introduction - Learn web development. URL https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction (Viitattu 17.3.2022)

MDN. 2022d. Introducing asynchronous JavaScript - Learn web development. URL https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Introducing (Viitattu 21.3.2022)

MDN. 2022e. How to use promises - Learn web development. URL https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Promises (Viitattu 22.3.2022)

MDN. 2022f. Introduction to web APIs - Learn web development. URL https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction (Viitattu 26.3.2022)

MDN. 2022g. Introduction to client-side frameworks - Learn web development. URL https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction (Viitattu 27.3.2022)

MDN. 2022h. What is JavaScript? - Learn web development. URL https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (Viitattu 20.8.2022)

MDN. 2022i. Getting started with React - Learn web development. URL https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started (Viitattu 29.3.2022).

MDN. 2022j. CSS selectors - Learn web development. URL https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors (Viitattu 18.11.2022)

Montrose Kevin. 2014. Stack Overflow Blog - Stack Exchange API V2.2. URL https://stackoverflow.blog/2014/02/10/stack-exchange-api-v2-2-and-the-demise-of-v1-x/ (Viitattu 29.9.2022)

NLTK. 2022. Natural Language Toolkit. URL https://www.nltk.org/ (Viitattu 26.8.2022)

Node.js. 2022. Node.js - homepage. URL https://nodejs.org/en/ (Viitattu 17.3.2022).

Ocariza Frolin S., Bajaj Kartik, Pattabiraman Karthik, and Mesbah Ali. 2017. A Study of Causes and Consequences of Client-Side JavaScript Bugs. *IEEE transactions on software engineering* 43, 2 (2017), 128–144. DOI https://doi.org/10.1109/TSE.2016.2586066

Omondi Austin R. 2022. How to create your own custom React Hooks. LogRocket Blog. URL https://blog.logrocket.com/create-your-own-custom-react-hooks/ (Viitattu 19.11.2022)

Park Thomas and Wiedenbeck Susan. 2011. Learning web development: challenges at an earlier stage of computing education. In *(ICER '11)*, ACM, 125–132. DOI https://doi.org/10.1145/2016911.2016937

Park Thomas, Dorn Brian, and Forte Andrea. 2015. An Analysis of HTML and CSS Syntax Errors in a Web Development Course. *ACM transactions on computing education* 15, 1 (2015), 1–21. DOI https://doi.org/10.1145/2700514

Park Thomas, Saxena Ankur, Jagannath Swathi, Wiedenbeck Susan, and Forte Andrea. 2013. Towards a taxonomy of errors in HTML and CSS. In *(ICER '13)*, ACM, 75–82. DOI https://doi.org/10.1145/2493394.2493405

ReactJS. 2022a. A JavaScript library for building user interfaces. URL https://reactjs.org/ (Viitattu 29.3.2022)

ReactJS. 2022b. Introducing JSX. URL https://reactjs.org/docs/introducing-jsx.html (Viitattu 30.3.2022).

ReactJS. 2022c. Rendering Elements. URL https://reactjs.org/docs/rendering-elements.html (Viitattu 31.3.2022)

ReactJS. 2022d. Components and Props. URL https://reactjs.org/docs/components-and-props.html (Viitattu 4.4.2022)

ReactJS. 2022e. Handling Events. URL https://reactjs.org/docs/handling-events.html (Viitattu 6.4.2022)

ReactJS. 2022f. Conditional Rendering. URL https://reactjs.org/docs/conditional-rendering.html (Viitattu 7.4.2022).

ReactJS. 2022g. Lists and Keys. URL https://reactjs.org/docs/lists-and-keys.html (Viitattu 8.4.2022)

ReactJS. 2022h. Forms. URL https://reactjs.org/docs/forms.html (Viitattu 9.4.2022)

ReactJS. 2022i. Introducing Hooks. URL https://reactjs.org/docs/hooks-intro.html (Viitattu 10.4.2022)

ReactJS. 2022j. Hooks at a Glance. URL https://reactjs.org/docs/hooks-overview.html (Viitattu 10.4.2022).

Robards Timothy. 2021. React: Understanding State & Lifecycle. Itnext. URL https://itnext.io/react-understanding-state-lifecycle-d45df5d2cf3f (Viitattu 16.4.2022).

Rosen Christoffer and Shihab Emad. 2015. What are mobile developers asking about? A large scale study using stack overflow. *Empirical software engineering: an international journal* 21, 3 (2015), 1192–1223. DOI https://doi.org/10.1007/s10664-015-9379-3

Shah Hardik. 2022. 7 Frontend JavaScript Frameworks Loved by Developers in 2022. Simform - Product Engineering Company.URL https://www.simform.com/blog/javascript-frontend-frameworks/ (Viitattu 22.9.2022)

StackAPI. 2022. Stack Exchange API. URL https://api.stackexchange.com/ (Viitattu 20.6.2022)

StackOverflow. 2022. Programming related question and answer website. URL https://stackoverflow.com/ (Viitattu 6.10.2022)

Techopedia. 2020. What is Web Development? - Definition from Techopedia. URL http://www.techopedia.com/definition/23889/web-development (Viitattu 26.2.2022)

Wang Jie, Dou Wensheng, Gao Yu, Gao Chushu, Qin Feng, Yin Kang, and Wei Jun. 2017. A comprehensive study on real world concurrency bugs in Node.js. IEEE, 520–531. DOI https://doi.org/10.1109/ASE.2017.8115663

Wang Shaowei, Lo David, and Jiang Lingxiao. 2013. An empirical study on developer interactions in StackOverflow. In *(SAC '13)*, ACM, 1019–1024. DOI https://doi.org/10.1145/2480362.2480557

Webopedia. 2020. What is Web Development? – Definition from Webopedia. URL https://www.webopedia.com/definitions/web-development/ (Viitattu 26.2.2022).

Wegner A. 2022. A python wrapper for the Stack Exchange API. URL https://github.com/AWegnerGitHub/stackapi (Viitattu 20.6.2022)