Reactin ja Angularin vertailu web-sovelluksen kehityksessä

Lauri Reis

Haaga-Helia ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
2024

Tiivistelmä

Tekijä
Lauri Reis

Tutkinto
Tradenomi

Opinnäytetyön nimi
Reactin ja Angularin vertailu web-sovelluksen kehityksessä

Sivu- ja liitesivumäärä
40 + 3

Tämän opinnäytetyön tarkoituksena oli vertailla Reactia ja Angularia luomalla niille web-sovellukset. Opinnäytetyö on luonteeltaan toiminnallinen, joka tarkoittaa, että työssä muodostui tuotos. Tuotos oli kaksi yksinkertaista työntekijähallinnan verkkosovellusta sekä sovelluksille jaettu tietokanta ja palvelin. Vertailu toteutettiin testaamalla sovelluksia sekä objektiivisesti että subjektiivisesti. Vertailua avustavat tutkimuskysymykset olivat:

K1 Kumpi teknologia tarjoaa paremman kehityskokemuksen?
K2 Kumpi teknologia tarjoaa parempaa suorituskykyä?

Tuotosta on pohjustettu tietoperustassa kertomalla sekä Reactin että Angularin keskeisimpiä tekniikoita, joita tuotoksessa on tarvittu.

Toteutuksessa ohjelmoitiin paikallinen palvelin sekä React- ja Angular-sovellukset. Menetelmäkuvauksessa havainnollistettiin valittuja toimintatapoja sovellusten ohjelmoinnista.

Lopputuloksissa sovelluksia vertailtiin Googlen Lighthouse-testeillä. Testit vertailivat suorituskykyä, saavutettavuutta, parhaita toimintatapoja ja hakukoneoptimointia. Luvussa myös vertailtiin sovelluksia subjektiivisesti kehittäjäkokemuksen perusteella sekä ulkoisen avun kuten foorumeiden kattavuudella.

Tutkimuskysymysten vastaukset eivät olleet täysin selviä. Tutkimuskysymyksen K1 vastaus riippui useista tekijöistä, kuten projektin vaatimuksista ja web-kehittäjän tottumuksista. Sekä React että Angular tarjosivat miellyttävän kehityskokemuksen. Tutkimuskysymyksen K2 vastaus oli myös epäselvä. React tarjosi parempaa suorituskykyä Lighthouse-testien perusteella, mutta molemmat sovellukset olivat yhtä responsiivisia eikä lopullista päätöstä voitu tehdä tämän mittaluokan sovelluksista.

Asiasanat
Web-kehitys, JavaScript, sovelluskehys, käyttöliittymä

Sisällys

1 Johdanto

Web-kehityksessä tapahtuu jatkuvaa muutosta ja uusien teknologioiden esiintuloa. Web-kehityksessä on tultu pitkälle 1990-luvun alkupuolen staattisista ja tekstipainotteisista sivuista. Etenkin Ajaxin (Asynchronous JavaScript And XML) julkaisun jälkeen 1990-luvun loppupuolella verkkosivuista tuli responsiivisempia. Ennen Ajaxia verkkosivun päivittäminen vaati koko sivun uudelleen lataamisen (W3Schools s.a. a). Responsiivisista verkkosivuista tuli pian uusi normi, jolloin sovelluskehyksiä ja -kirjastoja alettiin kehittämään helpottamaan verkkosivujen kehitystä.

Nykypäivänä on tarjolla monia erilaisia käyttöliittymäsovelluskehyksiä (eng. *front-end-framework*) ja -kirjastoja. Käyttöliittymäsovelluskehykset sisältävät ennalta kirjoitettua koodia, jotka tarjoavat rakenteen verkkosivustojen kehittämiseen. Käyttöliittymäsovelluskehykset yleensä sisältävät kirjastoja, työkaluja ja muita ominaisuuksia käyttöliittymien kehitykseen. Käyttöliittymäkirjastot tarjoavat lähes samoja etuja. Käyttöliittymäkirjastot yleensä vaativat vähemmän alkumäärityksiä ja niissä on vähemmän rajoituksia kuin sovelluskehyksissä. (UXPin 8.11.2023.) Käyttöliittymäkirjastoissa on kuitenkin lähes poikkeuksetta vähemmän toimintoja.

Web-kehitystä opetteleville erilaiset sovelluskehykset tai kirjastot eivät välttämättä ole paras valinta, mutta osaavammalle kehittäjälle sovelluskehyksien ja kirjastojen käyttäminen on lähes itsestäänselvyys. Aloittelevan web-kehittäjän voi olla parempi kirjoittaa raakaa HTML:ää ja JavaScriptiä ettei uuden oppiessa oio kulmia ja hyppää taitotasoon nähden liian hankalaan konseptiin. Taitojen kehittyessä kuitenkin sovelluskehykset ja kirjastot ovat loistava valinta jatkaa opettelua. Sovelluskehykset ja kirjastot tarjoavat useita etuja raakaan koodikieleen verrattuna, kuten työnkulun tehostus, parempi koodin laatu ja nopeampi alustojen välinen kehitys (UXPin 8.11.2023).

Tässä opinnäytetyössä keskitytään Reactiin, joka on käyttöliittymäkirjasto sekä Angulariin, joka on käyttöliittymäsovelluskehys. Molemmat teknologiat pohjautuvat JavaScriptiin, mutta toimivat hieman eri tavalla ja tarjoavat erilaisia etuja.

Sekä React että Angular ovat nykypäivänä hyvin suosittuja. Vuoden 2023 Stack Overflown kehittäjäkyselyyn vastasi 71 802 ohjelmistokehittäjää. React on toiseksi suosituin web-kehityksessä käytettävä teknologia ja Angular viidenneksi suosituin kaikkien vastaajien perusteella. Työelämässä olevien perusteella React on suosituin ja Angular neljänneksi suosituin. React on yleisen mielipiteen mukaan helpompi opetella, kun taas Angularissa oppimiskäyrä on suurempi. Aloittelevien

ohjelmistokehittäjien mukaan Angular on 11. suosituin web-kehityksen teknologia. React yltää taas toiselle sijalle. (Stack Overflow 2023.)

React ja Angular valikoituivat tähän opinnäytetyöhön oman mielenkiintoni sekä yleisen suosion takia. Käymilläni kursseilla opeteltiin Reactia sekä suositeltiin käyttämään sitä myös muissa projekteissa. Angulariin tutustuin työelämässä. Mielestäni Reactin ja Angularin välille saa hyvää vertailua, sillä niiden toimintatavat ja koodin kirjoitustyyli ovat hyvin erilaisia, vaikka molemmat pohjautuvatkin JavaScriptiin.

Tämä opinnäytetyö tarkastelee Reactin ja Angularin välistä vertailua, keskittyen erityisesti niiden käyttökokemukseen ja suorituskykyyn. Vertailu toteutetaan luomalla yksinkertaiset CRUD-sovellukset (eng. *Create*, *Read*, *Update*, *Delete*, suom. Luo, Lue, Päivitä, Poista) sekä Reactille että Angularille ja arvioimalla kehittäjäkokemusta sekä suorituskykyä näiden sovellusten perusteella. Opinnäytetyössä ei mennä pintaa syvemmälle Reactin ja Angularin haastavampiin teknillisyyksiin, sillä kolmannessa luvussa esiteltävä esimerkkisovellus ei vaadi teknisesti vaativaa toteutusta.

Tekstissä on paljon englanninkielisiä termejä, joita on hankala suomentaa. Englanninkielisen termin ensimmäisellä kirjoituskerralla termi on *kursivoitu* helpottamaan lukemista.

1.1 Tutkimuskysymykset

Tässä opinnäytetyössä pyritään vastamaan seuraaviin kysymyksiin:

K1 Kumpi teknologia tarjoaa paremman kehityskokemuksen?

K2 Kumpi teknologia tarjoaa parempaa suorituskykyä?

Kehityskokemusta vertaillaan esimerkiksi tarkastelemalla, kuinka haastavaa koodi on kirjoittaa ja onko opettelu hankalaa. Kehityskokemusta vertaillaan myös molempien teknologioiden tarjoamalla dokumentaatiolla sekä ulkoisella avulla, kuten kysymyksiin vastausten etsiminen foorumeilta.

Suorituskykyvertailu toteutetaan Googlen Lighthouse-työkalulla. Lighthouse on Googlen kehittämä avoimen lähdekoodin automatisoitu työkalu verkkosivujen laadun parantamiseen. Ohjelman voi suorittaa jokaiselle nettisivulle. Ohjelmassa on testejä muun muassa suorituskyvylle ja saavutettavuudelle. Ohjelmaa voi ajaa Chromen kehittäjätyökaluista. (Google 2016.) Lighthouse valikoitui tähän työhön helppokäyttöisyyden, avoimen lähdekoodin ja tunnettavuuden takia.

1.2 Keskeiset käsitteet

1. HTML: **H**yper**T**ext **M**arkup **L**anguage, hypertekstin merkintäkieli. HTML tunnetaan erityisesti merkintäkielenä verkkosivujen valmistuksessa.
2. CSS: **C**ascading **S**tyle **S**heets, porrastetut tyyliarkit. Käytetään yhdessä HTML:n kanssa verkkosivujen tyylimäärityksiä varten.
3. Ajax: **A**synchronous **J**avaScript **A**nd **X**ML. Ajax käyttää selaimeen sisäänrakennettua XMLHttpRequest-objektia pyytääkseen dataa serveriltä sekä JavaScriptiä ja HTML DOMia esittääkseen tai käyttääkseen dataa (W3Schools s. a. a).
4. DOM: **D**ocument **O**bject **M**odel, objektimalli. Objektimalli on dataesitys elementeistä, jotka muodostavat verkkosivun rakenteen ja sisällön (Mozilla 2023).
5. CLI: **C**ommand **L**ine **I**nterface, komentorivityökalu.
6. npm: **n**ode **p**ackage **m**anager, Node paketinhallintatyökalu. npm on vakiona Node.js:n mukana tuleva paketinhallintatyökalu.

2 Tietoperusta

Tässä luvussa perehdytään tarkemmin kolmannessa luvussa esiteltävän esimerkkisovelluksen käyttämiin teknologioihin ja niiden yksityiskohtiin sekä sovelluksen ohjelmointiin tarvittaviin työkaluihin.

2.1 JavaScript

JavaScript on ohjelmointikieli, jota käytetään pääasiassa verkkosivujen tekoon. JavaScriptiä voi myös käyttää selaimen ulkopuolisiin toimiin. (Mozilla 2024.) JavaScript on yksi kolmesta keskeisestä tekniikasta verkkosivujen kehittämisessä HTML:n ja CSS:n ohella. JavaScript mahdollistaa dynaamisten ja interaktiivisten elementtien lisäämisen verkkosivulle, kuten nappien, animaatioiden tai lomakkeiden validoinnin. JavaScriptissä itsessään on hyvin minimaalinen ohjelmointirajapinta eikä se sisällä syöttö- tai tulostustoimintoja. Syöttö- ja tulostustoiminnot jäävät isäntäympäristön vastuulle, johon JavaScript on upotettu, kuten HTML:n. (Flanagan & Vaughan 2023, luku 1.) JavaScriptin olennainen osa on myös asynkronisten kutsujen käyttö, joka mahdollistaa verkkosivujen päivittämisen ilman sivun uudelleenlataamista.

Raa'an JavaScriptin ja HTML:n käyttö nykypäivänä ei ole kovinkaan yleistä. Monet web-kehittäjät valitsevatkin sovelluskehyksen tai kirjaston, kuten Reactin tai Angularin. Nämä teknologiat tukevat uudelleenkäytettävien käyttöliittymäkomponenttien luomista. (Flanagan & Vaughan 2023, luku 15.6.) Uudelleenkäytettävät komponentit nopeuttavat kehitysprosessia huomattavasti.

Tietoperustassa käsiteltävät React ja Angular pohjautuvat molemmat JavaScriptiin. Sekä Reactissa että Angularissa on lisäominaisuuksia, jotka houkuttelevat käyttäjiä, mutta molemmilla teknologioilla voi kirjoittaa myös täysin raakaa JavaScriptiä lisäominaisuuksia hyödyntämättä.

2.2 React

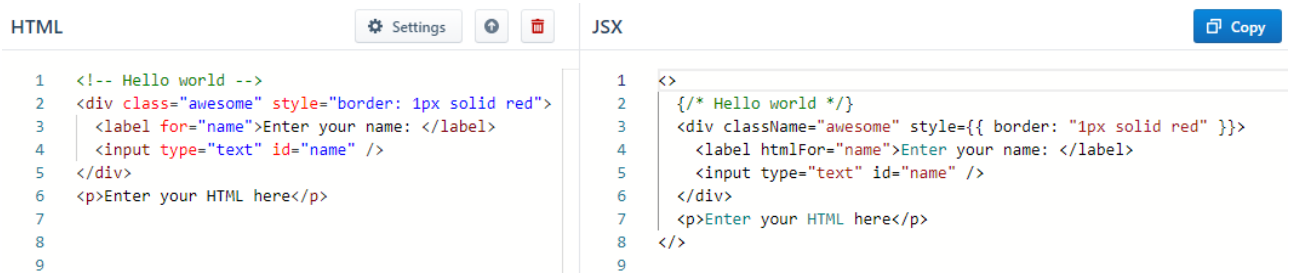
React on Metan kehittämä JavaScript-kirjasto. Reactin toimintaperiaate perustuu komponenttien käyttöön, jotka ovat itsenäisiä koodin palasia. (Meta s.a. a.) React-komponentti on JavaScript-luokka tai -funktio, joka vastaanottaa syötteitä ja tulostaa käyttöliittymäelementin. Yksittäisiä komponentteja yhdistellen voi rakentaa kompleksin käyttöliittymän, jonka React renderöi. (Narayn 2022, luku 1.) Reactia voi kuvailla käyttöliittymäkirjastona. Sen avulla voi koota komponentteja, mutta se ei määrittele, miten reititys ja datan nouto tehdään (Meta s.a. a). Näitä toimia varten voi asentaa paketteja tai käyttää valmiimpaa sovelluskehystä.

2.2.1 JSX

JSX on JavaScriptin syntaksin lisäosa, joka mahdollistaa HTML-tyyppisen koodin kirjoittamisen JavaScript-tiedostoon. JSX-merkinnän kirjoittaminen ei ole pakollista Reactia varten, mutta suurin osa ohjelmistokehittäjistä suosivat sitä. (Meta s.a. b.)

JSX-syntaksissa sekä JavaScript-logiikka että HTML-tyyppinen merkintä ovat samassa tiedostossa verrattuna perinteisempään tyyliin, jossa koodit ovat eri tiedostoissa. Tällainen lähestymistapa saattaa helpottaa kehittäjäkokemusta, sillä ohjelmistokehittäjä näkee sekä yksittäisen komponentin JavaScript-logiikan että HTML-tyyppisen JSX-merkinnän.

JSX on hyvin samannäköistä kuin HTML. JSX-merkinnässä on kuitenkin pieniä eroja. HTML-merkintää voi helposti kääntää käsin JSX-syntaksiin. Jos kääntäessä tulee virheitä, virheviestit yleensä auttavat hyvin korjaamaan ne. Kääntämiseen myös voi käyttää apuna netistä löytyviä kääntäjiä (kuva 1). React-komponentissa JSX-merkintä kirjoitetaan tiedoston loppuun *return*-lausekkeen sisään.



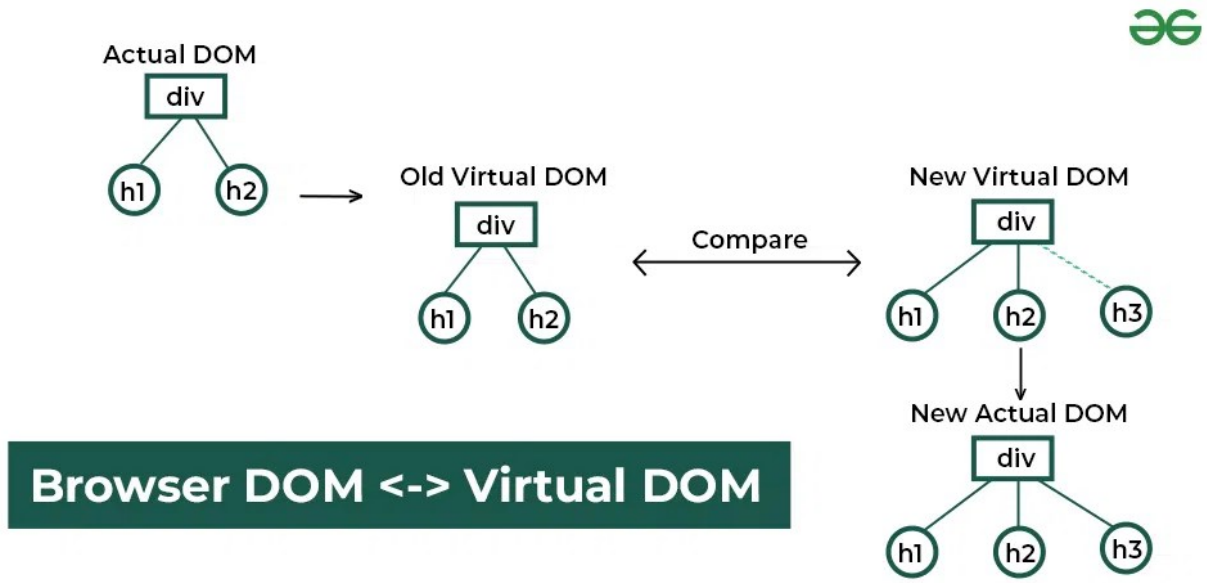
Kuva 1. HTML-JSX kääntäjä (Transform s.a.)

2.2.2 Virtuaalinen objektimalli (VDOM)

Verkkosivun lataamisen yhteydessä selain luo sivusta objektimallin. Objektimallilla esitetään verkkosivulla olevia elementtejä. Objektimalli yleensä kuvataan puun muodossa, jolloin ylhäällä on niin sanottu pääelementti, jonka alapuolella on lapsielementit. Objektimallia päivittäessä päivitetty elementti ja sen lapsielementit täytyy renderöidä uudestaan. (GeeksforGeeks 2023.) Tämä on raskas operaatio, jossa virtuaalinen objektimalli tulee apuun.

React käyttää virtuaalista objektimallia. Virtuaalinen objektimalli on ikään kuin kevyt kopio oikeasta objektimallista. Kaikki objektimallissa olevat elementit sijaitsevat myös virtuaalisessa objektimallissa, mutta virtuaalinen objektimalli ei voi päivittää alkuperäistä objektimallia.

Reactin päivittäessä arvoja se tekee muutoksia virtuaaliseen objektimalliin. Kuvassa 2 koodiin on lisätty uusi alaotsikko ”h3”. React vertaa uutta virtuaalista objektimallia, johon muutos on tehty vanhan virtuaalisen objektimallin kanssa. Vertailun päätyttyä React päivittää alkuperäiseen objektimalliin vain sen muutoksen, joka on tapahtunut.



Kuva 2. Virtuaalisen objektimallin päivitys alkuperäiseen objektimalliin (GeeksforGeeks 2023)

2.2.3 useState- ja useEffect-koukut

Reactissa on käytössä koukkuja (eng. *hook*), joilla voi hyödyntää erilaisia Reactille tyypillisiä ominaisuuksia. Koukut ovat JavaScript-funktioita, joilla voi eristää tilallisen (eng. *stateful*) logiikan sivuvaikutukset React-komponentista. Tilallista logiikkaa on esimerkiksi datan haku ulkoisista palveluista tai tilamuuttujan käyttö. (Adhikary 15.3.2022.) Kolmannessa luvussa esiteltävässä esimerkkisovelluksessa on käytössä kaksi erilaista koukkua: *useState* ja *useEffect*. On myös mahdollista luoda koukkuja itse. Koukut nimetään standardin mukaisesti alkamaan *use*-sanalla.

useState-koukulla komponenttiin voi lisätä tilamuuttujan. Tilamuuttujien tarkoitus on muistaa jokin arvo komponentin sisällä. Yleinen käytäntö tilamuuttujien nimeämiseen on taulukon ”hajottaminen” (eng. *array destructuring*). (Meta s.a. c.) Kuvan 3 esimerkissä tilamuuttuja merkitään syntaksilla *const [age, setAge] = useState(42)*. useState-koukun parametriksi voi syöttää alkutilan. Muuttujan *age* alkuarvo olisi tässä esimerkissä 42. Muuttujan arvoa voi vaihtaa *set*-funktiolla (Meta s.a. c).

Kuvassa 3 *name*-muuttujan arvo vaihdetaan *handleClick*-funktion sisällä, jolloin arvo ”Taylor” vaihtuu arvoksi ”Robin”. Jos handleClick-funktion sisällä tulostaa name-muuttujan arvon, se on edelleen ”Taylor”, sillä useState-koukku päivittää arvon vasta seuraavasta renderöinnistä lähtien (Meta s.a. c). useState-koukkua käytetään esimerkkisovelluksessa muun muassa asettamaan lomakkeen kenttiä sekä tallentamalla tilamuuttujaan tietokannasta tulevaa tietoa taulukkomuodossa.

Kuva 3. useState-koukun käyttöesimerkki (Meta s.a. c)

useEffect-koukku mahdollistaa komponentin synkronoinnin ulkoisen järjestelmän kanssa (Meta s.a. d). Esimerkkisovelluksessa palvelin ja tietokanta eivät ole Reactin kontrolloimia, joten ne määritellään ulkoisiksi. useEffect-koukun sisällä oleva koodi suoritetaan joka kerta, kun komponentti renderöidään. Jos useEffect-koukun loppuun määriteltävät riippuvuudet muuttuvat, useEffect suoritetaan jälleen. (Meta s.a. d.) Esimerkkisovelluksessa useEffect-koukkua käytetään muun muassa tietokannasta työntekijöiden hakuun. useEffect-koukun sisällä on asynkroninen *getRecords*-funktio, joka hakee työntekijät *fetch*-funktiolla. Jos virheitä ei tapahdu, data asetetaan set-funktiolla

records-tilamuuttujaan. useEffect-koukun riippuvuudeksi asetetaan records-tilamuuttujan pituus. (Kuva 4.)

```
useEffect(() => {
  async function getRecords() {
    const response = await fetch(`http://localhost:5200/employees/`)
    if (!response.ok) {
      const message = `An error occurred: ${response.statusText}`
      console.error(message)
      return
    }
    const records = await response.json()
    setRecords(records)
  }
  getRecords()
  return
}, [records.length])
```

Kuva 4. useEffect-koukun käyttöesimerkki

Ennen koukkujen esittelyä Reactin versiossa 16.8 luokkapohjaiset komponentit olivat ainoa tapa seurata muuttujan tilaa ja elinkaarta (W3Schools s.a. b). Reactin nykyversiossa luokkapohjaisia komponentteja ei enää suositella käytettävän (Meta s.a. e). Koukkuja käytetään vain funktiopohjaisissa komponenteissa.

2.2.4 Vite

Reactille on olemassa suuri määrä työkaluja projektin luomiseen. Yksi hyvin suosittu työkalu on *Create React App (CRA)*, joka ajetaan *npx*-komennolla. CRA:n suosio on kuitenkin laskenut, eikä Reactin dokumentaatio enää suosittele sitä (Hinkula 2023, alaluku Creating and running a React app).

Create React App käyttää pakkaajana (eng. *bundler*) Webpackia ja Vite ESbuildia (Madushan A. 26.1.2024). Vaikka esimerkkisovellus on pieni, nopeuseron huomaa välittömästi. Viten ESbuild on nopeampi esimerkiksi asentamaan paketit, käynnistämään sovelluksen ja päivittämään sovelluksen koodin muututtua. Viten käyttö myös helpottaa seuraamaan MongoDB:n tutoriaalia.

2.3 Angular

Angular on Googlen kehittämä JavaScript-sovelluskehys. Reactin ollessa vain JavaScript-kirjasto Angular tarjoaa suuremman määrän ominaisuuksia sen ollessa kokonainen sovelluskehys. Suurimpina eroina Reactiin Angularissa on oma komentorivityökalu ja perinteisen JavaScriptin sijaan käytetään TypeScriptiä. (Google 2023a.)

Angular on Reactin tavoin komponentteihin perustuva, mutta Angularin komponentit kuitenkin erottuvat suuresti. React on nykyversiossaan funktiopohjainen, kun taas Angular on luokkapohjainen. Angularissa komponentin erottaa *@Component*-etuliitteestä luokan edellä. Etuliitteen sisään merkitään esimerkiksi komponentin nimi, HTML-malli (eng. *template*), jota komponentti käyttää sekä tyylimääritykset. (Google 2023a.)

2.3.1 TypeScript

TypeScript on Microsoftin kehittämä JavaScriptin *superset*, joka tarkoittaa JavaScriptin päälle lisättyjä sääntöjä, jolloin JavaScript-syntaksi on oikeanlaista TypeScript-syntaksia. TypeScript on tyypitetty superset. TypeScript lisää erilaisia sääntöjä millaisia arvoja voidaan käyttää. (Microsoft 2024.) Kuvan 5 funktio JavaScriptillä antaisi suorittaa laskutoimituksen, jolloin tulokseksi tulisi ääretön. TypeScriptin tehtävä on estää tällaisten tilanteiden syntyminen jo koodin kirjoitusvaiheessa, jolloin koodieditori näyttää virheilmoituksen.

```
console.log(4 / []);
```

```
The right-hand side of an arithmetic operation must be of type 'any',
'number', 'bigint' or an enum type.
```

Kuva 5. Funktio TypeScriptissä (Microsoft 2024)

Jokainen Angular-sovellus käyttää vakiona TypeScriptiä tarjotakseen parempia työkaluja kehitykseen sekä paranneltua ylläpidettävyyttä paremman kehittäjäkokemuksen vuoksi (Google 2023a).

Reactin kanssa on myös mahdollista käyttää TypeScriptiä, mutta se pitää erikseen lisätä projektin luontikomentoon lipulla.

2.3.2 Angular CLI

React-projektien luontiin käytetään npm-paketinhallintatyökalua. Angular-projektien luontiin käytetään Angularin omaa komentorivityökalua. Komentorivityökalu asennetaan myös npm-paketinhallinnan kautta. Angular komentorivityökalu käyttää *ng*-etuliitettä, jolla voi tehdä Angulariin liittyviä toimintoja. Esimerkiksi uusi projekti luodaan komennolla *ng new* ja projekti ajetaan komennolla *ng serve*. (Google s.a.)

2.3.3 Projektin rakenne

Reactissa koko komponentin koodi löytyy yhdestä tiedostosta. Tiedoston yläreunassa on JavaScript-koodi, jonka alla on return-lauseke, josta JSX-syntaksin web-elementit löytyvät. Angularin rakenne on kuitenkin hyvin erilainen. Generoidessa uuden komponentin komennolla *ng generate component* Angular luo projektin sisään kansion, josta projektin asetuksista riippuen löytyy komponentin TypeScript-, HTML-, CSS- sekä spec- eli testitiedosto. Pienessä sovelluksessa, kuten kolmannen luvun esimerkkisovelluksessa suurista tiedostomääristä saattaa olla haittaa ohjelmistokehittäjästä riippuen. Todellinen hyöty tulee ilmi kymmeniä komponentteja ja satoja rivejä pitkistä tiedostoista koostuvista projekteista, jolloin tiedostorakenne tuo vahvaa struktuuria.

2.3.4 Kaksisuuntainen sidonta

Kaksisuuntainen sidonta on yksi Angularin suurimmista eroista verrattuna Reactiin. Kaksisuuntainen sidonta mahdollistaa komponenttien välisen datan välityksen. Kaksisuuntaista sidontaa voi käyttää tapahtumien kuuntelemiseen ja arvojen päivittämiseen samanaikaisesti pää- ja lapsikomponentin välillä. Kaksisuuntaisen sidonnan syntaksi on ”[()]”, joka yhdistää hakasulut ominaisuussidonnasta (eng. *property binding*) ja sulut tapahtumasidonnasta (eng. *event binding*). (Google 2022.) Muuttuja asetetaan sulkujen väliin ja koko syntaksi asetetaan pääkomponenttiin lapsikomponentin valitsimeen.

Kaksisuuntaista sidontaa hyödyntävät muuttujat merkitään komponentteihin @*Input*- ja @Output-etuliitteillä. @Input antaa pääkomponentin päivittää dataa lapsikomponentissa ja @Output antaa lapsikomponentin päivittää dataa pääkomponenttiin (Google 2023b). Yksisuuntaista sidontaa varten tarvitsee vain käyttää sulkuja, jolloin kyse on tapahtumasidonnasta. Tapahtumasidonnassa lapsikomponenttiin merkitään muuttuja @Input-etuliitteellä.

Reactissa dataa välitetään lapsikomponentille *propsien* avulla. Props tarkoittaa sanaa *properties* (suom. ominaisuudet). Propsit ovat React-komponenteille parametreinä välitettäviä argumentteja. (W3Schools s.a. c.)

ChildComponent-funktion parametriksi asetetaan props ja funktiossa palautetaan teksti. Komponentti renderöidään asettamalla *name*- ja *paper*-muuttujiin halutut arvot, jolloin komponentista palautettava teksti on “You are reading Lauri’s thesis!”. (Kuva 6.)

```
export default function MainComponent() {
    return (
        <>
            <ChildComponent name="Lauri" paper="thesis" />
        </>
    )
}

function ChildComponent(props) {
    return (
        <>
            <p>You are reading {props.name}'s {props.paper}!</p>
        </>
    )
}
```

Kuva 6. React-propsien välitys lapsikomponentille

Angularin tyylinen kaksisuuntainen sidonta ei ole mahdollista Reactissa. Reactissa tilamuuttuja asetetaan pääkomponenttiin, jolloin se välitetään lapsikomponentille propseilla, josta sitä voi kontrolloida. Angularissa muuttuja voi myös sijaita lapsikomponentissa, jolloin sitä voidaan kontrolloida mistä tahansa komponentista käyttäen @Input- ja @Output-etuliitteitä.

Esimerkkisovelluksessa ei tarvita kaksisuuntaista sidontaa. Sovelluksessa käytetään vain ominaisuussidontaa ja @Output-etuliitettä lähettäessä lomakkeen tiedot pääkomponenttiin.

2.4 Käytettävät työkalut

Molempien teknologioiden koodaamiseen käytetään VS Code -tekstieditoria, jossa Prettier-lisäosa kontrolloi koodin indentointia. Angular-sovellukseen tarvitsee lisäksi Angularin oman komentorivityökalun, jolla projektin sekä esimerkiksi uusien komponenttien luominen onnistuu. Sovelluksille

luodaan MongoDB-tietokanta. Tietokantaa kontrolloidaan MongoDB:n omasta Atlas-pilvipalvelusta selaimen avulla. Sovelluksien lopputuloksia vertaillaan Googlen Lighthouse-työkalulla.

3 Toteutus

Tässä luvussa luodaan esimerkkisovellus molemmille teknologioille. Sovellus tulee olemaan yksinkertainen työntekijälista, jossa käyttäjä voi sovelluksen etusivulla katsella, luoda, muokata ja poistaa työntekijöitä. Työntekijät tallennetaan MongoDB-tietokantaan, josta ne voi hakea, kun käyttäjä käynnistää sovelluksen.

Molemmat sovellukset luodaan MongoDB:n omiin tutoriaaleihin perustuen. Sovellukset eivät ole täysin samanlaiset kuin tutoriaaleissa esitetyt, mutta perustuvat siihen tekniikoiltaan. React-sovellus seuraa MongoDB:n MERN-stack-tutoriaalia (MongoDB s.a. a) ja Angular-sovellus MEAN-stack-tutoriaalia (MongoDB s.a. b).

Sekä MERN- että MEAN-tutoriaaleissa luodaan samantyylinen tietokanta. Esimerkkisovelluksen tietokannaksi valikoitui MEAN-tutoriaalin mukainen, sillä siinä käytetään TypeScriptiä. Tutoriaalien tietokannat ovat muuten sisällöltään samanlaisia. Vaikka esimerkkisovelluksia on kaksi, vain yksi tietokanta riittää. Tietokantaan lisätään tutoriaalin mukaisten ohjeiden lisäksi lippu, jolla tietokannasta erottaa kummalla sovelluksella dataa on lisätty.

Sovellukset perustuvat kuvan 7 kaltaiseen arkkitehtuuriin. Sekä Reactilla että Angularilla luodaan frontend-sovellukset, jotka käyttävät paikallista palvelinta, joka yhdistää MongoDB-tietokantaan. Palvelin käyttää Expressiä ja Node.js:ää apunaan tuottaakseen REST-rajapinnan.

Kuva 7. Arkkitehtuurikaavio (mukaillen MongoDB s.a. b)

3.1 Tietokanta

MongoDB-tietokantaa pystyy käyttämään myös paikallisesti omalla tietokoneella, mutta yksinkertaisuuden vuoksi tähän toteutukseen valitsin MongoDB:n Atlas-pilvipalvelun, jolla tietokantaa kontrolloidaan selaimesta.

Tietokanta luodaan Atlas-pilvipalvelussa ennen siihen yhdistämistä. Palvelusta valitsin uuden projektin, jonka tässä tapauksessa nimesin ”ont”. Seuraavaksi luodaan tietokantaklusteri. Valitsin

mallipohjaksi M0:n, joka on ainoa ilmainen vaihtoehto. Klusterin toimittajaksi valitsin AWS:n ja palvelimen sijainniksi Tukholman (eu-north-1). Klusterin nimeksi annoin myös ”ont”. Klusterille luodaan käyttäjä sekä lisätään IP-osoite, jolla tietokantaan saa tehtyä yhteyden. Tietokanta on nyt luotu ja siihen on mahdollista muodostaa yhteys sovelluksesta. Yhteys muodostetaan tarkemmin sovelluskohtaisesti tulevissa alaluvuissa.

Paikallinen palvelin luodaan MEAN-stack tutoriaalin ohjeiden mukaan (MongoDB s.a. b). Aloitin luomalla tyhjän projektin server-kansioon *npm init -y* -komennolla. Komento luo package.json tiedoston. Komennon onnistuessa konsoliin tulostuu kuvan 8 kaltainen tuloste.

```
lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/server
$ npm init -y
Wrote to C:\Users\lauri\Documents\ONT\server\package.json:

{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

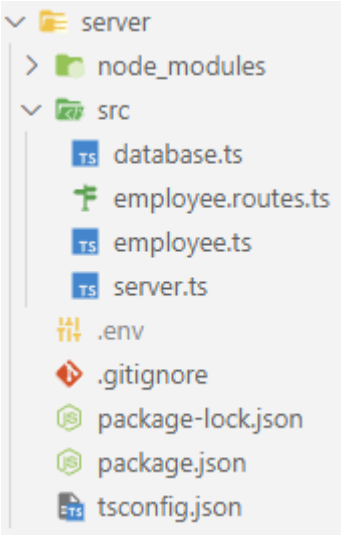
Kuva 8. Server-projektin luonti

Loin server-projektiin *touch*-komennolla tsconfig.json-tiedoston TypeScriptin konfiguraatiota varten, sekä .env-tiedoston MongoDB:n yhdistämisavainta varten, jossa luetellaan tietokantaklusterin nimi, käyttäjänimi sekä salasana. Asensin projektiin samalla tarvittavat paketit tietokantaan yhdistämistä varten. Viimeisimpänä –*save-de*v-lipulla asensin TypeScriptin ja tyyppimäärittelyjä helpottamaan kehittäjäkokemusta. (Kuva 9.) Lisäsin projektiin myös .gitignore-tiedoston, ettei .env-tiedoston salasana joudu GitHub-julkaisun mukana julkiseksi.

```
lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/server
$ touch tsconfig.json .env

lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/server
$ npm install cors dotenv express mongodb

added 79 packages, and audited 80 packages in 5s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/server
$ npm install --save-dev typescript @types/cors @types/express @types/node ts-node

added 31 packages, and audited 111 packages in 3s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Kuva 9. Pakettien asennus

Alkumäärittelyjen jälkeen voi aloittaa itse koodin kirjoittamisen. Loin projektiin src-kansion, jonka sisään loin neljä TypeScript-tiedostoa: *database*, *employee.routes*, *employee* ja *server*. Projektin lopullinen kansiorakenne on kuvan 10 mukainen.



Kuva 10. Server-projektin kansiorakenne

Employee.ts-tiedostoon loin rajapinnan yksittäiselle työntekijälle. Rajapintaan listataan työntekijän nimi (“name”), tehtävänimike (“position”), objekti-id (“_id?”), sovellus, josta työntekijä on lisätty (“app”) sekä asema (“level”), joka on yksi kolmesta: junior, mid tai senior. Rajapintaa käytetään myöhemmin database.ts-tiedostossa varmistaakseen, että tietokantaan menevä data on muodoltaan oikeanlaista, eikä vääränmuotoisen datan syöttäminen ole mahdollista.

Database.ts-tiedostossa loin funktion, jolla yhdistetään Atlakseen. Funktio luo uuden MongoClient-objektin, jolla yhdistäminen tapahtuu. Funktiossa tapahtuu myös skeeman validointia, jossa tarkastetaan tietokantaan syötettävän datan muoto JSON Scheman avulla.

Employee.routes.ts-tiedostossa loin reitit, joilla työntekijä-objekteja voidaan lähettää, muokata ja poistaa tietokannasta. Tiedostossa on myös reitit kaikkien työntekijöiden hakemiselle sekä yksittäisen työntekijän hakemiselle id:n perusteella. Jokaisella reitillä on käytössä myös virheenhallintaa, ettei tietokannassa pääse tapahtumaan virheitä korruptoituneen datan takia.

Server.ts-tiedostossa haetaan Atlaksen yhdistämisavain .env-tiedostosta, jota käytetään *connectToDatabase*-funktiossa (kuva 11). Kun kaikki toimii oletetusti, käynnistäessä palvelimen npx-komennolla saadaan tuloste, joka tulee connectToDatabase-funktiosta. Palvelin on onnistuneesti käynnistynyt portille 5200. (Kuva 12.)

```
connectToDatabase(ATLAS_URI)
  .then(() => {
    const app = express()
    app.use(cors())

    app.use('/employees', employeeRouter)
    app.listen(5200, () => {
      console.log(`Server running at http://localhost:5200/`)
    })
  })
  .catch((error) => console.error(error))
```

Kuva 11. server.ts-tiedoston connectToDatabase-funktio

```
lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/server (master)
$ npx ts-node src/server.ts
Server running at http://localhost:5200/
```

Kuva 12. Palvelimen käynnistys ja tietokantaan yhdistäminen

Valmiin sovelluksen lähdekoodi on saatavilla liitteessä 1.

3.2 React

Tämä luku kuvaa sovelluksen React-version toteutuksen.

3.2.1 Projektin luonti

React-projekti luodaan Vitellä *npm create vite@latest* -komennolla. Projektin nimeksi annoin ”client-react”. Luontiprosessi komentorivissä on interaktiivinen, jolloin nuolinäppäimillä sovelluskehykseksi valitsin ”React” ja variantiksi ”JavaScript”. Vitellä on mahdollista valita luontiprosessissa sovelluskehykseksi useita eri sovelluskehyksiä sekä JavaScriptillä että TypeScriptillä, mutta tässä esimerkissä yksinkertaisuuden vuoksi käytetään Reactia JavaScriptillä.

Komennon valmistuttua konsoliin pitäisi tulostua kuvan 13 mukainen tuloste, jolloin projektin voi ajaa lokaalisti *npm run dev* -komennolla.

```
lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT
$ npm create vite@latest client-react
√ Select a framework: » React
√ Select a variant: » JavaScript

Scaffolding project in C:\Users\lauri\Documents\ONT\client-react...

Done. Now run:

  cd client-react
  npm install
  npm run dev
```

Kuva 13. React-projektin luonti Vitellä

Projektin yksinkertaisuuden vuoksi paketteja tarvitsee vain vähän. Ainoa sovelluksen toimintaan aidosti vaikuttava paketti on reitittimenä toimiva *react-router-dom*, jolla luodaan reitit eri komponenttien välille. Sen lisäksi asensin myös tyylimäärittelyjä, kuten Tailwind CSS:n. Pakettien onnistuneen asennuksen jälkeen Tailwind CSS:lle luodaan konfiguraatiotiedosto. (Kuva 14.) Paketteina asennettavat tyylimäärittelyt eivät ole pakollisia, mutta nopeuttavat kehitysprosessia, sillä tyylimäärittelyyn ei tarvitse käyttää suurta määrää aikaa.

```
lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-react (master)
$ npm install -D tailwindcss postcss autoprefixer react-router-dom

added 364 packages, and audited 365 packages in 36s

126 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-react (master)
$ npx tailwindcss init -p

Created Tailwind CSS config file: tailwind.config.js
Created PostCSS config file: postcss.config.js
```

Kuva 14. Pakettien asennus ja Tailwind CSS:n konfiguraation luominen

Loin sovellukselle seuraavaksi tarvittavat komponentit: *Navbar*, *Record* ja *RecordList* (kuva 15). Projektin luonnin pohjatyö on nyt tehty, jonka jälkeen pääsee kirjoittamaan koodia työntekijöiden listausta varten.

```
lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-react (master)
$ mkdir src/components

lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-react (master)
$ cd src/components

lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-react/src/components (master)
$ touch Navbar.jsx Record.jsx RecordList.jsx
```

Kuva 15. Komponenttien luonti

3.2.2 Työntekijöiden listauksen implementointi

Aikaisemmin mainittu paketti react-router-dom tulee käyttöön *main.jsx*-komponentissa. Loin komponenttiin reitittimen, joka säätelee mihin komponenttiin käyttäjä viedään tietyn polun perusteella. Polku ”/” eli aloitussivu vie käyttäjän RecordList-komponenttiin. Polut ”/edit:id/” sekä ”/create” vievät käyttäjän Record-komponenttiin.

RecordList-komponentissa haen työntekijät asynkronisesti Reactin useEffect-koukulla. Jos tietokannasta tullut data on hyväksyttyä, se asetetaan useState-koukkua käyttävään *records*-tilamuuttujaan.

Yksittäisen työntekijän poisto hoidetaan *deleteRecord*-funktiolla. Tietokantaa kutsutaan useEffect-koukun sisällä JavaScriptin fetch-funktion *delete*-metodilla. Poistaessa työntekijän URL:n polkuun lisätään poistettavan työntekijän id. Poistettu työntekijä poistuu RecordList-komponentista suodattamalla poistettava työntekijä JavaScriptin *filter*-funktiolla. (Kuva 16.)

```
const [records, setRecords] = useState([])

useEffect(() => {
  async function getRecords() {
    const response = await fetch(`http://localhost:5200/employees/`)
    if (!response.ok) {
      const message = `An error occurred: ${response.statusText}`
      console.error(message)
      return
    }
    const records = await response.json()
    setRecords(records)
  }
  getRecords()
  return
}, [records.length])

async function deleteRecord(id) {
  await fetch(`http://localhost:5200/employees/${id}`, {
    method: 'DELETE',
  })
  const newRecords = records.filter((el) => el._id !== id)
  setRecords(newRecords)
}
```

Kuva 16. Työntekijöiden haku ja poisto tietokannasta

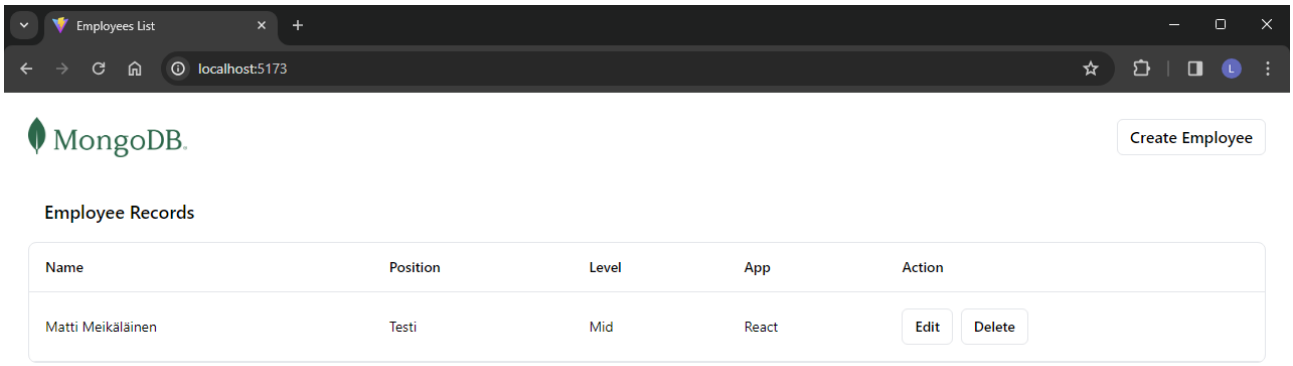
Record-komponentin *onSubmit*-funktiolla joko luodaan uusi työntekijä tai päivitetään jo tietokannassa olevaa. Jos kyseessä on id:n perusteella uusi työntekijä, fetch-funktion kanssa käytetään *POST*-metodia ja lähetyksen *body*-osioon laitetaan uuden työntekijän tiedot. Jos id:n perusteella on kyseessä jo tietokannassa oleva työntekijä, käytetään *PATCH*-metodia, jolloin URL:n perään laitetaan kyseisen työntekijän id ja lähetyksen body-osioon muuttuneet tiedot. (Kuva 17.)

Tietojen lähettämisen jälkeen *try-catch*-lausekkeen *finally*-osio tyhjentää lomakkeen ja ohjaa käyttäjän RecordList-komponenttiin (kuva 17).

```
async function onSubmit(e) {
  e.preventDefault()
  const person = { ...form }
  try {
    let response
    if (isNew) {
      response = await fetch('http://localhost:5200/employees', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(person),
      })
    } else {
      response = await fetch(`http://localhost:5200/employees/${params.id}`, {
        method: 'PATCH',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(person),
      })
    }

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`)
    }
  } catch (error) {
    console.error('A problem occurred with your fetch operation: ', error)
  } finally {
    setForm({ name: '', position: '', level: '' })
    navigate('/')
  }
}
```

Kuva 17. Työntekijöiden lisääminen ja päivitys tietokantaan

Sovelluksen CRUD-päätoiminnot on nyt toteutettu. Projektin voi ajaa npm run dev -komennolla. Kuvassa 18 on React-sovelluksen valmis käyttöliittymä yhdellä tietokantaan lisätyllä testikäyttäjällä. Taulukon *App*-kohdassa näkyy, että työntekijä on lisätty React-sovelluksesta.



Kuva 18. React-sovelluksen käyttöliittymä

Valmiin sovelluksen lähdekoodi on saatavilla liitteessä 2.

3.3 Angular

Tämä luku kuvaa sovelluksen Angular-version toteutuksen.

3.3.1 Projektin luonti

Angular-projekti luodaan käyttämällä Angularin omaa komentorivityökalua. Loin projektin *ng new* -komennolla ja annoin nimeksi ”client-angular”. Viten tavoin luontiprosessi on interaktiivinen: komentorivi kysyy esimerkiksi mitä tyyliarkkimuotoa haluaa käyttää. Valitsin muodoksi CSS:n, sillä sovelluksessa ei tarvitse monimutkaisempia tyyliasetuksia. (Kuva 19.) MongoDB:n tutoriaali käyttää luontikomennossa useaa lippua, jolla sovelluksesta saadaan minimalistisempi. Vertailun vuoksi on kuitenkin parasta tällä kertaa olla käyttämättä lisälippuja ja kokea Angular kuten kehittäjät ovat sen alun perin tarkoittaneet.

```
lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT
$ ng new client-angular
? Which stylesheet format would you like to use? CSS             [
https://developer.mozilla.org/docs/Web/CSS                      ]
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? No
CREATE client-angular/angular.json (2727 bytes)
CREATE client-angular/package.json (1083 bytes)
CREATE client-angular/README.md (1094 bytes)
CREATE client-angular/tsconfig.json (889 bytes)
CREATE client-angular/.editorconfig (290 bytes)
CREATE client-angular/.gitignore (590 bytes)
CREATE client-angular/tsconfig.app.json (277 bytes)
CREATE client-angular/tsconfig.spec.json (287 bytes)
CREATE client-angular/.vscode/extensions.json (134 bytes)
CREATE client-angular/.vscode/launch.json (490 bytes)
CREATE client-angular/.vscode/tasks.json (980 bytes)
CREATE client-angular/src/main.ts (256 bytes)
CREATE client-angular/src/favicon.ico (15086 bytes)
CREATE client-angular/src/index.html (312 bytes)
CREATE client-angular/src/styles.css (81 bytes)
CREATE client-angular/src/app/app.component.html (20239 bytes)
CREATE client-angular/src/app/app.component.spec.ts (969 bytes)
CREATE client-angular/src/app/app.component.ts (323 bytes)
CREATE client-angular/src/app/app.component.css (0 bytes)
CREATE client-angular/src/app/app.config.ts (235 bytes)
CREATE client-angular/src/app/app.routes.ts (80 bytes)
CREATE client-angular/src/assets/.gitkeep (0 bytes)
✓ Packages installed successfully.
    Successfully initialized git.
```

Kuva 19. Angular-projektin luonti

Projektin luonnin jälkeen asennetaan vielä *@angular/material*-tyylipaketti. Asennus kysyy kysymyksiä, joiden vastauksiin voi vaikuttaa valitsemalla haluamansa vaihtoehdon nuolinäppäimillä. Peruskonfiguraatio kuitenkin toimii tähän tapaukseen, joten jokaiseen kysymykseen voi vastata painamalla Enteriä. (Kuva 20.)

```
lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-angular (master)
$ ng add @angular/material
i Using package manager: npm
✓ Found compatible package version: @angular/material@17.3.3.
✓ Package information loaded.

The package @angular/material@17.3.3 will be installed and executed.
Would you like to proceed? Yes
✓ Packages successfully installed.
? Choose a prebuilt theme name, or "custom" for a custom theme: Indigo/Pink        [ Preview:
https://material.angular.io?theme=indigo-pink ]
? Set up global Angular Material typography styles? No
? Include the Angular animations module? Include and enable animations
UPDATE package.json (856 bytes)
✓ Packages installed successfully.
    Your project is not using the default builders for "test". This means that we cannot add the configured theme to the
 "test" target.
UPDATE src/app/app.config.ts (347 bytes)
UPDATE angular.json (3053 bytes)
UPDATE src/index.html (512 bytes)
UPDATE src/styles.css (182 bytes)
```

Kuva 20. Material-kirjaston lisääminen projektiin

Pakettien asennuksen jälkeen generoin kaikki tarvitsemani komponentit, palvelun ja rajapinnan (kuva 21). Komponenteille Angular CLI luo kaikki neljä tiedostoa: HTML-, TypeScript-, CSS- ja testitiedostot. Rajapinnalle tulee vain yksi TypeScript-tiedosto ja palvelulle itse palvelu sekä testitiedosto.

Tiedostojen määrä on valtava verrattuna React-sovellukseen. React-sovelluksessa on kolme itsegeneroitua komponenttia, jolloin tiedostojen määrä on myös kolme. Angular-sovelluksessa on neljä komponenttia, palvelu ja rajapinta, jolloin itsegeneroituja tiedostoja on 19.

```
lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-angular (master)
$ ng generate interface employee
CREATE src/app/employee.ts (32 bytes)

lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-angular (master)
$ ng generate service employee
CREATE src/app/employee.service.spec.ts (383 bytes)
CREATE src/app/employee.service.ts (146 bytes)

lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-angular (master)
$ ng generate component employees-list
CREATE src/app/employees-list/employees-list.component.html (30 bytes)
CREATE src/app/employees-list/employees-list.component.spec.ts (669 bytes)
CREATE src/app/employees-list/employees-list.component.ts (277 bytes)
CREATE src/app/employees-list/employees-list.component.css (0 bytes)

lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-angular (master)
$ ng generate component employee-form
CREATE src/app/employee-form/employee-form.component.html (29 bytes)
CREATE src/app/employee-form/employee-form.component.spec.ts (662 bytes)
CREATE src/app/employee-form/employee-form.component.ts (273 bytes)
CREATE src/app/employee-form/employee-form.component.css (0 bytes)

lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-angular (master)
$ ng generate component add-employee
CREATE src/app/add-employee/add-employee.component.html (28 bytes)
CREATE src/app/add-employee/add-employee.component.spec.ts (655 bytes)
CREATE src/app/add-employee/add-employee.component.ts (269 bytes)
CREATE src/app/add-employee/add-employee.component.css (0 bytes)

lauri@LAPTOP-NCPR5RUQ MINGW64 ~/Documents/ONT/client-angular (master)
$ ng generate component edit-employee
CREATE src/app/edit-employee/edit-employee.component.html (29 bytes)
CREATE src/app/edit-employee/edit-employee.component.spec.ts (662 bytes)
CREATE src/app/edit-employee/edit-employee.component.ts (273 bytes)
CREATE src/app/edit-employee/edit-employee.component.css (0 bytes)
```

Kuva 21. Komponenttien generointi

3.3.2 Työntekijöiden listauksen implementointi

Työntekijäpalveluun (*employee.service.ts*) loin CRUD-elementit tietokantaan yhdistämiseksi. *getEmployees*-metodi hakee kaikki työntekijät tietokannasta. Haku tapahtuu käyttämällä Angularin http-paketin *HttpClient*-ominaisuutta. HttpClientin avulla voi käyttää erilaisia http-metodeja. Kaikkien työntekijöiden sekä id:n perusteella tietyn työntekijän haku onnistuu *get*-metodilla. Työntekijän

luominen tapahtuu post-metodilla, päivittäminen *put*-metodilla ja poistaminen delete-metodilla. Sekä luonti- että päivitysmetodeissa parametrina annetaan aiemmin luotu työntekijäobjekti. (Kuva 22.)

Työntekijäpalvelua käytetään kaikkialla paitsi *employee-form*-komponentissa. Esimerkiksi *add-employee*-komponentissa työntekijäpalvelua kutsutaan komponentin omassa addEmployee-metodissa. Työntekijäpalvelun haluttu metodi tilataan komponentissa. Työntekijäpalvelu lähettää tietokantaan http-kutsun, jonka jälkeen komponentti ohjaa käyttäjän takaisin työntekijälistaukseen tilauksen *next*-kutsulla.

Angularissa hyvä tapa toimia on lopettaa *observable*-muuttujan tilaus komponentin tuhoutuessa. HttpClientin observable-muuttujat kuitenkin lopettavat tilauksen automaattisesti, kun palvelin vastaa joko onnistuneesti tai virheellä, tai jos palvelin aikakatkaisee (Google 2023c). Luonti-, muokkaus- ja poistotoiminnot tilaavat työntekijäpalvelulta tarvitsemansa metodin. Tässä tapauksessa metodin tilausta komponentissa ei tarvitse erikseen lopettaa.

Työntekijälistauksessa työntekijät haetaan työntekijäpalvelun avulla ja esitetään listanäkymässä. Listanäkymässä on myös työntekijäkohtaiset muokkaus- ja poistonapit. Listauksesta pääsee napilla luomaan uuden työntekijän, joka johdattaa käyttäjän employee-form-komponenttiin. Employee-form-komponentti käyttää Angularin *FormBuilder*-luokkaa *@angular/forms*-paketista.

```
private refreshEmployees() {
  this.httpClient
    .get<Employee[]>(`${this.url}/employees`)
    .subscribe((employees) => {
      this.employees$.set(employees);
    });
}

getEmployees() {
  this.refreshEmployees();
  return this.employees$();
}

getEmployee(id: string) {
  this.httpClient
    .get<Employee>(`${this.url}/employees/${id}`)
    .subscribe((employee) => {
      this.employee$.set(employee);
    });
  return this.employee$();
}

createEmployee(employee: Employee) {
  return this.httpClient.post(`${this.url}/employees`, employee, {
    responseType: 'text',
  });
}

updateEmployee(id: string, employee: Employee) {
  return this.httpClient.put(`${this.url}/employees/${id}`, employee, {
    responseType: 'text',
  });
}

deleteEmployee(id: string) {
  return this.httpClient.delete(`${this.url}/employees/${id}`, {
    responseType: 'text',
  });
}
```

Kuva 22. Työntekijäpalvelun CRUD-metodit

Työntekijäpalvelusta haettavat työntekijät esitetään *employees-list*-komponentissa. Työntekijät haetaan komponentin sisällä yksityisellä *fetchEmployees*-metodilla. fetchEmployees-metodi ajetaan komponentin käynnistyessä *ngOnInit*-metodissa. Angularin ngOnInit-metodi on verrattavissa Reactin useEffect-koukkuun, sillä molemmat ajetaan komponentin renderöinnin yhteydessä.

Työntekijät asetetaan fetchEmployees-metodissa *employees$*-muuttujaan. Angularissa yleinen tapa nimeämiseen on asettaa dollarisymboli muuttujan nimen perään, kun muuttuja on observable-muuttuja tai tässä tapauksessa signaali. Lisäksi näytettävät sarakkeet asetetaan *displayedColumns*-muuttujaan, jotta komponentin HTML-malli osaa näyttää oikean määrän sarakkeita oikeilla otsikoilla. (Kuva 23.)

```
export class EmployeesListComponent implements OnInit {
  employees$ = {} as WritableSignal<Employee[]>;
  displayedColumns: string[] = [
    'col-name',
    'col-position',
    'col-level',
    'col-app',
    'col-action',
  ];

  constructor(private employeeService: EmployeeService) {}

  ngOnInit() {
    this.fetchEmployees();
  }

  private fetchEmployees(): void {
    this.employees$ = this.employeeService.employees$;
    this.employeeService.getEmployees();
  }
```

Kuva 23. employees-list-komponentti

Komponentin HTML-malli esittää työntekijäpalvelusta haettavan datan taulukkomuodossa *<table>*-elementillä. Taulukkoelementtiin asetetaan konfiguraatioksi material-paketin *mat-table* sekä datan lähteeksi employees$-muuttuja ominaisuussidonnalla *[dataSource]*-syntaksiin. (Kuva 24.)

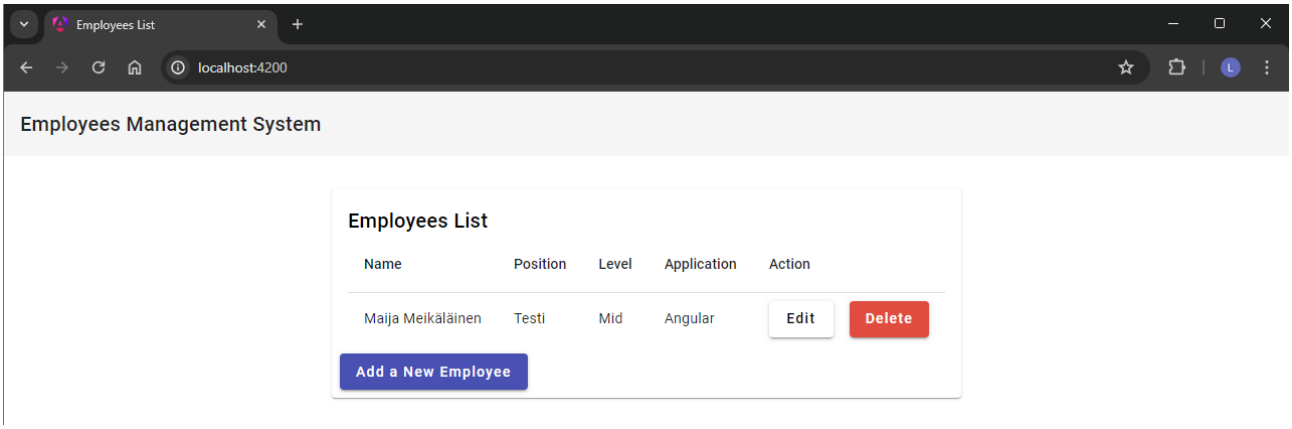
Taulukon *<td>*-elementit esittävät dataa taulukossa. Direktiiviin **matCellDef* asetetaan uusi muuttuja *element*, josta saraketta vastaava data valitaan. Taulukko osaa automaattisesti lisätä rivejä

saman verran kuin employees$-muuttujalla on työntekijäobjekteja. Taulukossa on lisäksi nappi työntekijän muokkaukseen, joka vie käyttäjän reitittimellä *edit*-polkuun sekä työntekijän poisto-nappi, joka poistaa työntekijän Angularin *click*-funktiolla komponentin *deleteEmployee*-metodilla. (Kuva 24.)

```
<mat-card-content>
  <table mat-table [dataSource]="employees$()">
    <ng-container matColumnDef="col-name">
      <th mat-header-cell *matHeaderCellDef>Name</th>
      <td mat-cell *matCellDef="let element">{{ element.name }}</td>
    </ng-container>
    <ng-container matColumnDef="col-position">
      <th mat-header-cell *matHeaderCellDef>Position</th>
      <td mat-cell *matCellDef="let element">{{ element.position }}</td>
    </ng-container>
    <ng-container matColumnDef="col-level">
      <th mat-header-cell *matHeaderCellDef>Level</th>
      <td mat-cell *matCellDef="let element">{{ element.level }}</td>
    </ng-container>
    <ng-container matColumnDef="col-app">
      <th mat-header-cell *matHeaderCellDef>Application</th>
      <td mat-cell *matCellDef="let element">{{ element.app }}</td>
    </ng-container>
    <ng-container matColumnDef="col-action">
      <th mat-header-cell *matHeaderCellDef>Action</th>
      <td mat-cell *matCellDef="let element">
        <button mat-raised-button [routerLink]="['edit/', element._id]">
          Edit
        </button>
        <button
          mat-raised-button
          color="warn"
          (click)="deleteEmployee(element._id || '')"
        >
          Delete
        </button>
      </td>
    </ng-container>

    <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
    <tr mat-row *matRowDef="let row; columns: displayedColumns"></tr>
  </table>
</mat-card-content>
```

Kuva 24. employees-list-HTML-malli

Tutoriaalia löyhästi seuraten sovelluksen käyttöliittymä näyttää kuvan 25 kaltaiselta. Listanäkymään lisäsin kentän React-sovelluksen tavoin, josta ilmenee kummasta sovelluksesta työntekijä on lisätty.



Kuva 25. Angular-sovelluksen käyttöliittymä

Valmiin sovelluksen lähdekoodi on saatavilla liitteessä 3.

4 Sovellusten tulosten vertailu

Tässä luvussa tarkastellaan, kuinka sovellukset lopulta eroavat toisistaan. Tuloksia verrataan sekä objektiivisesti testien avulla että subjektiivisesti kehittäjäkokemuksen perusteella.

4.1 Google Lighthouse

Lighthouse ajettiin sekä React- että Angular-sovelluksille.

React-sovellus pärjäsi testissä hyvin. Sovellus sai suorituskykytestistä tulokseksi 90 (taulukko 1). Suorituskykytestin sisäisistä mittareista kolme viidestä sai täydet pisteet. *First Contentful Paint (FCP)* sai pisteiksi 8/10 ja *Largest Contentful Paint (LCP)* 17/25. Tulokset mittaavat sekunneissa ensimmäisen ja suurimman tekstin tai kuvan renderöimistä. FCP oli 1,1 sekuntia ja LCP 1,9 sekuntia. Tulosta horjuttaa etusivulla oleva MongoDB:n logo, joka ladataan netistä. Ilman kuvaa tulos paranee hieman.

React-sovellus sai täydet pisteet parhaista toimintatavoista ja saavutettavuudesta. Hakukoneoptimointi sai 82 pistettä. Parannettavaa olisi lisätä sovellukseen meta-kuvaus. Testi myös valittaa *robots*-testitiedoston validiteetista, vaikka sellaista tiedostoa ei löydy projektista. Kokonaisuutena testin tulos on kuitenkin erittäin hyvä, etenkin verrattuna Angular-sovelluksen tuloksiin.

Angular-sovellus pärjäsi testissä keskinkertaisesti. Sovellus sai suorituskykytestistä tulokseksi 69, joka on 21 prosenttiyksikköä matalampi kuin React-sovelluksen tulos (taulukko 1). Suorituskykytestin sisäisistä mittareista kaksi viidestä sai täydet pisteet. React-sovelluksessa FCP ja LCP pysyivät epätäydellisistä arvoista huolimatta kohtuullisina. Angular-sovelluksessa arvot putosivat kuitenkin huomattavasti. FCP oli 2,3 sekuntia ja LCP 3,4 sekuntia. Angular-sovellus käyttää Googlen fontteja, jotka ladataan verkosta verkkosivun lataamisen yhteydessä. Tarkempi raportti valittaa fonttien lataamisesta aiheutuneesta hitaudesta, mutta testin suorittaminen uudelleen ilman Googlen fontteja ei kuitenkaan jostain syystä paranna tulosta.

Kuten React-sovellus, Angular-sovellus saa täydet pisteet parhaista käytännöistä ja saavutettavuudesta pois lukien delete-napin matalasta kontrastista. Hakukoneoptimointia varten Angular-sovellukseenkin pitäisi lisätä meta-kuvaus.

Taulukko 1. Sovelluksien Lighthouse-tulokset

Mittari	**React**	**Angular**
Suorituskyky	90	69
Saavutettavuus	100	94
Parhaat toimintatavat	100	100
Hakukoneoptimointi	82	89

Tällaiset tulokset olivat odotettavissa. Näin pienelle projektille Angular on “ylitehokas”. Sovelluksen kasvaessa Angularin hyöty myös kasvaa, mikä ei välttämättä tarkoita, että Reactin hyöty vähenisi. Angular on yleisen mielipiteen mukaan ennemmin suunnattu yrityksille kuin yksityishenkilöille, kun taas React on ennemmin suunnattu kaikille.

Tutkimuskysymyksen K2 tarkoitus oli selvittää, kumpi teknologia tarjoaa parempaa suorituskykyä. Lighthouse-testien perusteella React-sovellus tarjoaa paremman suorituskyvyn. Näin pienessä sovelluksessa tällaista eroa ei kuitenkaan huomaa. Vaikka pisteiden 90 ja 69 välillä on suuri kuilu, molemmat sovellukset ovat erittäin responsiivisia, eikä ilman testitulosten näkemistä voi päätellä, kumpi sovellus on suorituskykyisempi.

4.2 Kehittäjäkokemus

Kokeneelle JavaScript-kehittäjälle sekä Angularin että Reactin opetteleminen on suhteellisen vaivatonta. Kuitenkin omasta näkökulmastani Angular vaikuttaa selvästi vaativammalta. Uuden sovelluskehyksen lisäksi joutuisi opettelemaan TypeScriptin käytön, ellei jo ole tutustunut siihen.

Asensin molempiin sovelluksiin paketteja, jotka olivat lähinnä tyylimäärittelyjä, mutta React-sovellukseen lisäsin reitittimen, joka vaikuttaa suuresti sovelluksen toimintaan. Angularissa reititin tulee sisäänrakennettuna. Angularin sisällyttäessä reitittimen, sitä voisi pitää hieman monimutkaisempana ja valmiimpana pakettina osaavammalle web-kehittäjälle, kun taas React on helpompi oppia ja vaativammat toiminnot voi asentaa myöhemmin paketteina.

Lisäksi Angularin tiedostorakenne saattaa hämmentää aloittelevaa kehittäjää, sillä siinä on paljon tiedostoja ja rivejä. Peruskonfiguraatiolla jokaisella Angular-komponentilla on neljä tiedostoa verrattuna Reactin yhteen. Vaikka Reactin kanssa suositaan erillisten tyyli- ja testitiedostojen luomista,

yleinen käytäntö on sijoittaa tiedostot eri kansioihin, kun taas Angularissa ne ovat samassa kansiossa.

Yksikkötestaaminen Angularissa on yksinkertaista, sillä se käyttää Jasminea testauskehyksenä ja generoi automaattisesti testitiedoston luodessaan komponentin, palvelun tai muun vastaavan. Tämä yksinkertaistaa testien näkemistä ja kontekstin löytämistä. Yksikkötestit ajetaan *ng test* -komennolla.

Reactin mukana testaukseen tulee Jest. Jest tai React ei kuitenkaan itse generoi testitapauksia, joten ne pitää luoda itse. Jasmine on Jestin tavoin vain yksikkötestausta varten, mutta Jestiin voi asentaa *react-test-renderer*-paketin, jolla voi myös testata käyttöliittymää. Angularissa käyttöliittymää voi testata erilaisilla ulkoisesti asennettavilla työkaluilla, kuten Cypressilla. Käyttöliittymätestit ovat Reactin tai Angularin ulkopuolisia ominaisuuksia, jolloin molemmissa tapauksissa testit sijoitetaan erilliseen kansioon, eikä esimerkiksi komponenttikansion sisään.

Yleisellä tasolla sanoisin, että kehittäjäkokemus teknologioiden välillä riippuu web-kehittäjän taitotasosta. Aloittelevalle web-kehittäjälle React sopii paremmin ja Angular sopii paremmin jo hieman harjaantuneemmalle. Oma kokemukseni esimerkkisovelluksen parissa sai minut kallistumaan Angularin puoleen. Angular tuntuu valmiimmalta paketilta verkkosivujen kehitykseen. TypeScript tuo mukanaan tärkeitä sääntöjä, ettei huolimattomuusvirheitä pääse muodostumaan epähuomiossa. Angular on pakatumpi ominaisuuksilla, ja muissa projekteissa olen huomannut, että Reactilla toisinaan joutuu tekemään jonkin teknisesti haastavamman ratkaisun, kun Angularista sama ominaisuus löytyisi sisäänrakennettuna.

Angularin ollessa suunnatumpi suuremille projekteille käyttöönottoon tarvittavia taustatoimia on enemmän verrattuna Reactiin. Loin Angular-sovellukseen rajapinnan, palvelun ja komponentteja, kun taas React-sovelluksessa loin vain kolme komponenttia molempien ajaessa samaa asiaa. Angularin käyttöönotto on monimutkaisempaa, mutta mielestäni vaativan käyttöönoton jälkeen koodin ylläpito helpottuu. Komponentit, palvelut tai muut vastaavat ovat selkeämmin esillä kuin Reactissa. React soveltuu myös suuriin projekteihin, mutta mielestäni todellinen etu on yksityishenkilöiden tai pienten tiimien projekteissa. Sovelluksen saa nopeasti toimintaan eikä web-kehittäjän tekninen osaaminen ole yhtälailla avainasemassa.

Tällaisten asioiden takia Angularin oppimiskäyrä on paljon suurempi kuin Reactin. Angular tarjoaa valtavan määrän ennalta määritettyjä ominaisuuksia, joita ei välttämättä edes tarvitse

yksinkertaista verkkosivua ohjelmoidessa. JavaScriptiä osaavan web-kehittäjän on helppo aloittaa Reactin opettelu. React-sovellusta on helpohko lähteä koodaamaan vain minimaalisella opettelulla, kun taas Angularin toimintamalli on täysin erilainen, joka vaatii ekstensiivistä opettelua.

Kullekin teknologialle löytyy omat käyttökohteensa. Molemmat teknologiat suoriutuivat erinomaisesti esimerkkisovelluksen tekemisestä ja molempien käyttökokemus oli miellyttävä. Mielipiteeni Angularista ei kuitenkaan horjuta mielipidettäni Reactista, jota aion vielä varmasti käyttää tulevaisuudessa.

4.3 Dokumentaatio ja ulkoinen tuki

Sekä Reactilla että Angularilla on erinomaiset dokumentaatiot. Molemmat teknologiat ovat tarpeeksi suosittuja, että jokaisesta aiheesta löytyy kattava dokumentaatio malliesimerkeillä. Dokumentaatioita myös päivitetään hyvin usein. Kuitenkin sanoisin, että yleisilmeellään Angularin dokumentaatio vie voiton. Jokaisen sivun alareunassa lukee, milloin sivua on päivitetty, joka antaa lukijalle tunteen modernin tiedon lukemisesta. Reactin dokumentaatiossa tätä tietoa ei ole näkyvillä. Lisäseikkana Angularin dokumentaatio löytyy yhden linkin alta, kun Reactin dokumentaatio on jaettu kahteen osioon: opetteluun ja viitteeseen. Angularin opetteluosio on laitettu samaisen dokumentaatiolinkin alle.

Molempien teknologioiden ollessa suosittuja myös Google-haun automaattinen täyttö osaa usein auttaa ongelman etsimisessä, sillä monet muutkin ovat tuskailleet saman ongelman kanssa. Esimerkiksi haulla ”react pass state…” löytyy useita ehdotuksia, kuinka tilamuuttujan voi jakaa lapsikomponenttiin, pääkomponenttiin ja niin edelleen. Haulla ”angular observer…” löytyy yhtä paljon hyödyllisiä hakutuloksia. Usein ensimmäinen linkki osoittaa Stack Overflowiin, josta vastaus löytyy vähäisellä lukemisella.

Stack Overflowsta löytyy lähes 500 000 kysymystä React-avainsanalla. Angular-avainsanalla löytyy hieman yli 300 000 kysymystä. React-avainsanan kysymyksistä yli 40 % on vastaamatta ja Angular-avainsanan kysymyksistä noin 38,5 % on vastaamatta. Useat kysymykset myös voivat olla tuplakappaleita, jolloin samantyyppiseen kysymykseen on jo vastattu aikaisemmin. React-avainsanan seuraajia on yli kaksi kertaa enemmän verrattuna Angular-avainsanan seuraajiin kertoen Reactin ylivoimaisemmasta kuuluisuudesta.

Tutkimuskysymyksen K1 tarkoitus oli selvittää, kumpi teknologia tarjoaa paremman kehityskokemuksen. Tästä vertailusta selviää, että sekä React että Angular tarjoavat erinomaisen kehittäjäkokemuksen ja vastaus jää täysin kehitystiimin tai web-kehittäjän oman mielipiteen varaan. Angular tarjoaa vahvan rakenteen ja selkeän ohjeistuksen, joka voi olla hyödyllinen suurten projektien hallinnassa ja tiimityöskentelyssä. React tarjoaa joustavuutta ja kevyempää syntaksia, joka voi olla hyödyllinen pienemmissä projekteissa tai yksilökehittäjille, kuten aiemmin mainitsin. Lopullinen päätös siitä, kumpi teknologia tarjoaa paremman kehityskokemuksen riippuu useista tekijöistä, kuten projektin vaatimuksista sekä kehittäjän tai kehittäjien kokemuksesta ja mieltymyksistä. Päätös täytyy siis tehdä projektikohtaisesti, eikä väärää päätöstä voi välttämättä tehdä, sillä esimerkkisovellus sujui erinomaisesti molemmilla teknologioilla.

5 Pohdinta

Työn tarkoituksena oli luoda yksinkertainen CRUD-sovellus sekä Reactilla että Angularilla ja vertailla niiden suorituskykyä ja kehittäjäkokemusta. Uskon, että onnistuin tavoitteessani hyvin. Sovellusten kehittäminen sujui pääosin suunnitelmien mukaan tutoriaalien avulla, vaikka pieniä haasteita ilmenikin matkan varrella. Lopputulosten vertailuun pystyin suhtautumaan sekä objektiivisesti että subjektiivisesti.

Tulokset eivät kuitenkaan tuottaneet selkeää objektiivista johtopäätöstä, mikä oli odotettavissa. On olemassa monia sovelluskehyksiä ja kirjastoja juuri siksi, ettei yhtä oikeaa ratkaisua välttämättä ole olemassa. Sekä Angular että React soveltuvat esimerkkisovelluksen luomiseen, ja kummallakin on omat vahvuutensa ja käyttötarkoituksensa.

Työn suurin onnistuminen tuli lopputuloksien vertailusta, jotka erottuivat odotettua enemmän erityisesti Lighthouse-testien osalta. Lisäksi kehittäjäkokemuksen vertailu oli antoisaa, sillä se on subjektiivinen kokemus, johon ei ole yhtä oikeaa vastausta.

Tuloksia ja testausta on helppo toistaa, jolloin kuka tahansa voi oppia Reactin ja Angularin, kehittää CRUD-sovellukset ja vertailla niitä. Vaikka lopputulokset voivat vaihdella kehittäjän tekemien valintojen mukaan, kehittäjäkokemus pysyy suunnilleen samanlaisena ainakin teknologioiden nykyisillä versioilla.

Sovelluksia voisi edelleen kehittää monin tavoin. Esimerkiksi Angular-sovelluksen heikkoa testitulosta suorituskyvystä olisi syytä tutkia enemmän. Testituloksen heikkous ei johdu Angularin huonoudesta, vaan siitä, että React käsittelee asioita taustalla eri tavalla. Lisäksi molempiin sovelluksiin voitaisiin lisätä monimutkaisempaa logiikkaa ja komponentteja ja testata niitä, jolloin tulokset voisivat vaihdella.

Vaikka React ja Angular olivat minulle jo ennestään tuttuja, raportin kirjoittaminen opetti paljon ja haastoi minua kyseenalaistamaan asioita. Lopputulosten vertailu opetti minulle objektiivisen tarkastelun tärkeyden, kun pyrin jättämään ennakkoluuloni raportin ulkopuolelle.

Vaikka en kartuttanutkaan koodiosaamistani merkittävästi, kokonaisuudessaan työn tekeminen opetti minulle paljon, ja olen erittäin tyytyväinen aihevalintaani.

Lähteet

Adhikary, T. 15.3.2022. React Hooks Fundamentals for Beginners. freeCodeCamp. Luettavissa: https://www.freecodecamp.org/news/react-hooks-fundamentals/. Luettu: 14.5.2024.

Flanagan D. & Vaughan G. 2023. JavaScript. Ascent Audio. E-kirja.

GeeksforGeeks 2023. ReactJS Virtual DOM. Luettavissa: https://www.geeksforgeeks.org/reactjs-virtual-dom/. Luettu: 5.4.2024.

Google s.a. CLI Overview and Command Reference. Luettavissa: https://angular.io/cli. Luettu: 5.4.2024.

Google 2016. Overview. Luettavissa: https://developer.chrome.com/docs/lighthouse/overview. Luettu: 16.4.2024.

Google 2022. Two-way binding. Luettavissa: https://angular.io/guide/two-way-binding. Luettu: 24.4.2024.

Google 2023a. What is Angular?. Luettavissa: https://angular.io/guide/what-is-angular. Luettu: 10.3.2024.

Google 2023b. Sharing data between child and parent directives and components. Luettavissa: https://angular.io/guide/inputs-outputs. Luettu: 24.4.2024.

Google 2023c. HTTP: Request data from a server. Luettavissa: https://angular.io/guide/http-request-data-from-server. Luettu: 11.4.2024.

Hinkula, J. 2023. Full Stack Development with Spring Boot 3 and React: Build Modern Web Applications Using the Power of Java, React, and TypeScript, luku 7. Packt Publishing. Birmingham. E-kirja.

Madushan A. 26.1.2024. Vite vs Create React App in 2024. Bits and Pieces. Luettavissa: https://blog.bitsrc.io/vite-vs-create-react-app-326e8cc2c46b. Luettu: 9.4.2024.

Meta s.a. a. React. Luettavissa: https://react.dev/. Luettu: 29.2.2024.

Meta s.a. b. Writing Markup with JSX. Luettavissa: https://react.dev/learn/writing-markup-with-jsx. Luettu: 29.2.2024.

Meta s.a. c. useState. Luettavissa: https://react.dev/reference/react/useState. Luettu: 12.4.2024.

Meta s.a. d. useEffect. Luettavissa: https://react.dev/reference/react/useEffect. Luettu: 12.4.2024.

Meta s.a. e. Component. Luettavissa: https://react.dev/reference/react/Component. Luettu: 12.4.2024.

Microsoft 2024. TypeScript for the New Programmer. Luettavissa: https://www.typescript-lang.org/docs/handbook/typescript-from-scratch.html. Luettu: 10.3.2024.

MongoDB s.a. a. How to Use MERN Stack: A Complete Guide. Luettavissa: https://www.mongodb.com/languages/mern-stack-tutorial. Luettu: 29.2.2024.

MongoDB s.a. b. How to Use the MEAN Stack: Build a Web Application From Scratch. Luettavissa: https://www.mongodb.com/languages/mean-stack-tutorial. Luettu: 7.4.2024.

Mozilla 2023. Introduction to the DOM. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. Luettu: 15.4.2024.

Mozilla 2024. JavaScript. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript. Luettu: 5.4.2024.

Narayn, H. 2022. Just React!: Learn React the React Way. Apress L. P. Berkeley, CA. E-kirja.

Stack Overflow 2023. 2023 Developer Survey. Luettavissa: https://survey.stackoverflow.co/2023/#section-most-popular-technologies-web-frameworks-and-technologies. Luettu: 5.4.2024.

Transform s.a. HTML to JSX. Luettavissa: https://transform.tools/html-to-jsx. Luettu: 12.4.2024.

UXPin. 8.11.2023. Why Developers Use Frameworks?. UXPin Blog. Luettavissa: https://www.uxpin.com/studio/blog/why-developers-use-frameworks/. Luettu: 28.4.2024.

Vite s.a. Getting Started. Luettavissa: https://vitejs.dev/guide/. Luettu: 9.4.2024.

W3Schools s.a. a. AJAX Introduction. Luettavissa: https://www.w3schools.com/xml/ajax_intro.asp. Luettu: 7.5.2024.

W3Schools s.a. b. React Class Components. Luettavissa: https://www.w3schools.com/react/react_class.asp. Luettu: 12.4.2024.

W3Schools s.a. c. React Props. Luettavissa: https://www.w3schools.com/react/react_props.asp. Luettu: 14.5.2024.

Liitteet

Liite 1. Server-projekti

https://github.com/laurireis/ont-db

Liite 2: React-projekti

https://github.com/laurireis/ont-mern

Liite 3: Angular-projekti

https://github.com/laurireis/ont-mean