Tampereen ammattikorkeakoulu

Modernit verkkosovelluskehyk-set ja kirjastot

Henri Kemppainen

OPINNÄYTETYÖ
Syyskuu 2022

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

KEMPPAINEN, HENRI
Modernit verkkosovelluskehykset ja kirjastot

Opinnäytetyö 45 sivua, joista liitteitä 1 sivu.
Syyskuu 2022

Opinnäytetyön tarkoituksena oli tutustua kahteen eri käyttöliittymäverkkosovelluskehykseen, Reactiin ja Angulariin. Tarkastelen työssäni näiden kahden verkkosovelluskehyksen perusominaisuuksia sekä niiden oppimiskynnystä uusille sovelluskehittäjille. Lisäksi tarkastelen näiden suosiota maailmanlaajuisesti, sekä niiden työmarkkinoita Suomessa. Vertailen myös näiden kahden käytettävyyttä, ja pohdin niiden suosioiden syitä. Kerron myös lyhyesti Node.js:stä ja sen käyttötarkoituksesta verkkosovellusten kehityksessä.

Sovelluskehyksiin tutustuakseni, kehitin kaksi samanlaista ohjelmistoa, jotka mahdollistavat datan lukemisen, tallennuksen, muokkaamisen ja poistamisen. Palvelinpuolen, reitit ykset, ja tietokanta-kutsut hoitavat tätä työtä varten luomani Node.js-sovellus. Tietokannaksi valitsin dokumenttitietokannan MongoDB:n, josta myös kerron työssäni lyhyesti.

Asiasanat: React, Angular, Node.js, tietokanta, sovelluskehys, työmarkkinat

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT Engineering
Software Engineering

KEMPPAINEN, HENRI
Modern web frameworks and libraries

Bachelor's thesis 45 pages, appendices 1 page
September 2022

The purpose of the thesis was to get familiar with two different web application frameworks, React and Angular. In my work, I look at the basic features of these two web frameworks, and their learning curve for new app developers. I will also look at their popularity worldwide, as well as their job market in Finland. I also compare the usability of these two, and discuss the reasons for their popularity. I will also briefly discuss Node.js and its use in web application development.

To get acquainted with the application frameworks, I developed two identical software applications that allow reading, storing, editing and deleting data. The server-side, routing, and database calls are also handled by the Node.js-application I created for this thesis. The database I chose was MongoDB, which I will also briefly discuss in my thesis.

Key words: React, Angular, Node.js, database, framework, job market

SISÄLLYS

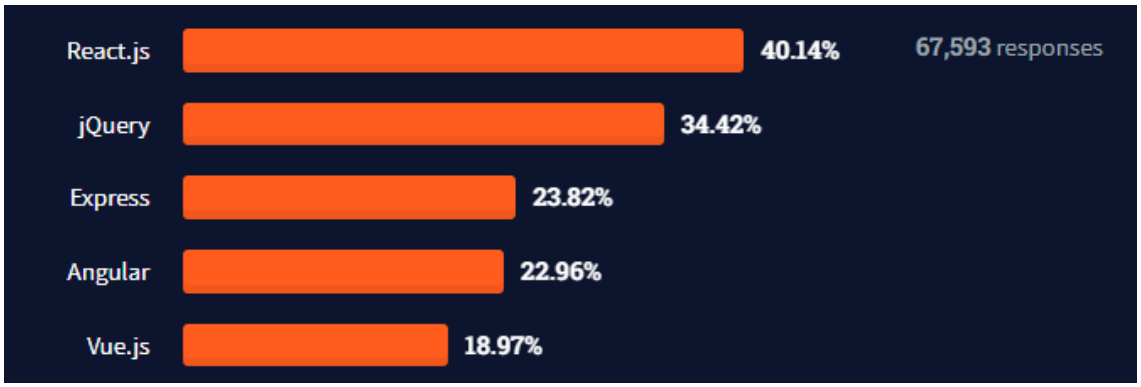
LYHENTEET JA TERMIT

SPA	Single-Page Application, yhden sivun verkkosivusto
JSX	JavaScript Extension Syntax. Laajennus JavaScriptille, joka toimii React sovelluksissa
DOM	Document Object Model. Dokumenttioliomalli. Käytetään tapana esittää verkkosivuston rakenne
JSON	JavaScript Object Notation. Avoimen standardin tiedostomuoto, jonka avulla voidaan välittää tietoa verkossa
API	Application Programming Interface. Rajapinta, jonka avulla eri järjestelmät ja ohjelmat voivat välittää tietoja toistensa välillä
CSS	Cascading Style Sheets. Käytetään verkkosivustojen tyylittelyyn
HTML	HyperText Markup Language. Avoimen standardin merkintäkieli, jonka avulla verkkosivustojen elementit luodaan
JavaScript	Verkkoympäristössä toimiva komentokieli, jonka avulla voidaan lisätä dynaamista toiminnallisuutta verkkosivustoille
TypeScript	JavaScriptin yläjoukko, joka lisää kieleen muun muassa vahvan staattisen tyypityksen
Komponentti	Yksi verkkosivun osa, jonka tehtävä on esittää tiettyä näkymää verkkosivulla

React Native	Reactin pohjalta tehty sovelluskehys, jota käytetään mobiilisovellusten rakentamiseen
Stack Overflow	Verkkosivusto, jonka pääaiheena on ohjelmointi ja ohjelmistoarkkitehtuuri ja niihin liittyvät ongelman ratkaisut. Linkki: https://stackoverflow.com/
CLI	Command-line interface. Komentoliittymä. Käytetään tietokoneen ja ihmisen välisen kommunikointiin, jonka kautta voidaan ajaa tietokoneen ohjelmia sekä muuttaa asetuksia
OOP	Object Oriented Programming. Ohjelmointiparadigma, jonka keskeisenä ajatuksena on objektien välinen yhteistoiminta

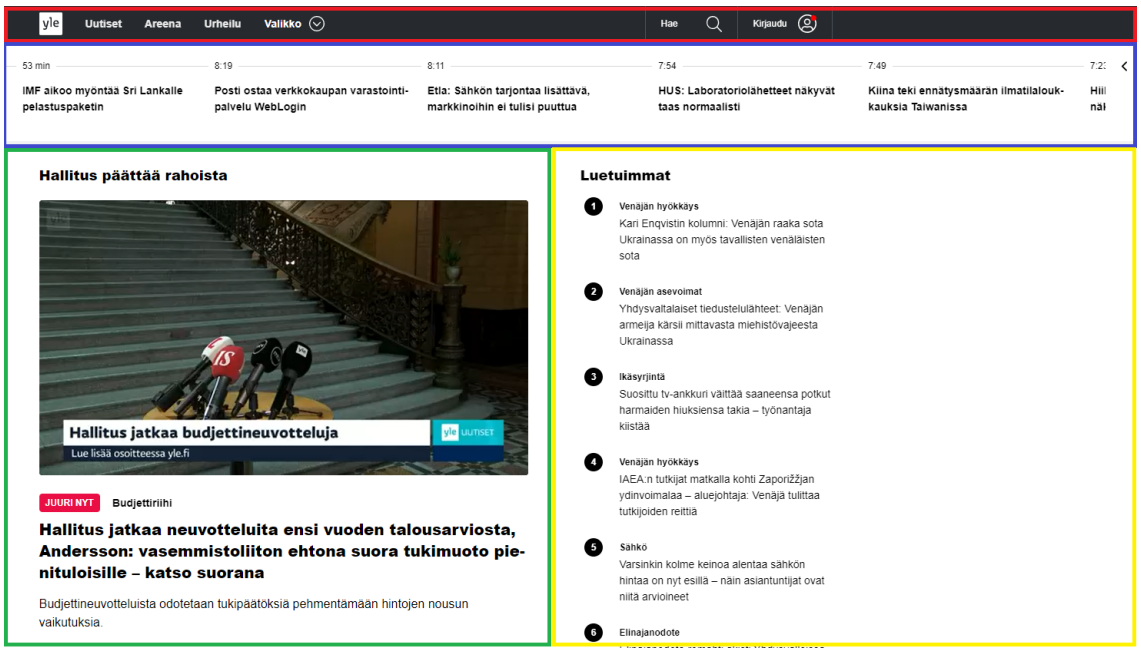
1 JOHDANTO

Verkkosovellusten kehittäminen otti suuria harppauksia nykyaikaan 2010-luvulla, kun verkkojättiläiset Google ja Facebook (nykyään Meta) julkaisivat omat verkkosovelluskehyksensä ja -kirjastonsa julkiseen käyttöön. Siitä lähtien sekä React että Angular ovat kasvattaneet suosiotaan tasaiseen tahtiin. Vuoden 2021 StackOverflow:n 68000 kehittäjän kyselyssä Reactia käytti 40.14% kehittäjistä, kun taas Angularia käytti 22.96% käyttäjistä. (StackOverflow, 2021) Kolmas suosittu käyttöliittymän verkkosovelluskehys on Kuva 1 näkyvä Vue, johon en tässä työssä perehtynyt.



Kuva 1: StackOverflow-kehittäjäkysely vuodelta 2021

Molempien sovelluskehysten ideana on rakentaa käyttöliittymä uudelleenkäytettävistä palasista, eli käyttöliittymä-komponenteista. Komponentti tarkoittaa sitä, että esimerkiksi verkkosivulla näkyvä navigaatiopalkki on oma komponenttinsa, sivun keskellä oleva uutisleike on oma komponenttinsa, ja niin edelleen. Kuva 2 on esitetty esimerkkitilanne, jossa eri värillä laatikoidut alueet ovat omia komponenttejaan.



Kuva 2: Kuvakaappaus Yle:n verkkosivuilta, tarkoituksena esittää komponenttiajattelu

2 SOVELLUSKEHYKSET JA -KIRJASTOT

2.1 Yleistä

Teknisesti ottaen, React ei ole sovelluskehys kuten Angular, vaan sovelluskirjasto. Puhuttaessa Reactista, yleensä sitä kuitenkin nimitetään sovelluskehykseksi. Sovelluskehys ja sovelluskirjasto termejä käytetään yleensä vaihdellen, ja eri konteksteissa ne voivat tarkoittaa monia eri asioita.

Sovelluskirjasto on nimensä mukaisesti kirjasto, joka sisältää erilaisia uudelleenkäytettäviä koodinpalasia, joita kehittäjä voi hyödyntää omassa sovelluksessaan. Esimerkki sovelluskirjastosta on esimerkiksi jokin matematiikan kirjasto, jonka avulla kehittäjä voi kutsua funktiota ilman, että hänen tarvitsee tehdä uudelleen algoritmin toimintaa koskevaa toteutusta.

Sovelluskehys on eräänlainen perusta, jonka päälle kehittäjät voivat rakentaa ohjelmistonsa, esimerkiksi tiettyjä alustoja varten. Se sisältää uudelleenkäytettäviä koodinpätkiä, jotka on kirjoitettu yleisten tehtävien suorittamista varten, ja se käyttää kehittäjän tekemää koodia mukautettuihin toimintoihin. Kehys voi sisältää määriteltyjä ja määrittelemättömiä objekteja ja funktioita, joita kehittäjät voivat käyttää sovellusten luomiseen. Näin järjestelmään voidaan lisätä merkittävää toiminnallisuutta käyttämällä olemassa olevaa koodia sovelluskehyksen ympärillä.

Tekninen ero kehyksen ja kirjaston välillä on termi, jota kutsutaan hallinnan kääntämiseksi. Kun käytät kirjastoa, olet vastuussa sovelluksen kulusta. Sinä päätät, milloin ja missä kirjastoa kutsutaan. Kun käytät sovelluskehystä, se vastaa tapahtumien kulusta. Se tarjoaa joitakin paikkoja, joihin voit liittää koodisi, mutta se kutsuu tekemääsi koodia tarpeen mukaan.

Sekä React että Angular ovat Single-Page -applikaatioita, eli yksisivuisia applikaatioita. Single-Page -applikaatio (lyhyemmin SPA), on verkkosovellus tai verkkosivusto, joka on vuorovaikutuksessa käyttäjän kanssa kirjoittamalla nykyinen verkkosivu dynaamisesti uudelleen uusilla tiedoilla verkkopalvelimelta sen sijaan, että verkkoselain lataisi oletusarvoisesti kokonaisia uusia sivuja. Tavoitteena on nopeammat siirtymät, jotka saavat verkkosivuston tuntumaan enemmän natiivisovellukselta.

Sovelluskehys	Kirjasto
Tarjoaa valmiita työkaluja, standardeja, malleja ja käytäntöjä nopeaan sovelluskehitykseen.	Tarjoaa uudelleenkäytettäviä toimintoja koodille.
Sovelluskehys valvoo kirjastojen kutsumista koodia varten.	Kehittäjän tekemä koodi ohjaa, milloin ja missä kirjastoa kutsutaan.
Sovelluskehyksen hyötyjen hyödyntämiseksi voidaan kehittää uusi sovellus kehyksen ohjeiden mukaisesti.	Kirjasto voidaan lisätä olemassa olevan sovelluksen ominaisuuksien täydentämiseksi.
Kehyksen tarkoituksena on vähentää ohjelmistokehitysprosessin monimutkaisuutta.	Kirjaston tarkoituksena on tarjota uudelleenkäytettäviä ohjelmistotoimintoja.

Taulukko 1: Sovelluskehyksen ja sovelluskirjaston eroavaisuuksia

2.2 React -sovelluskirjasto

React on Metan sekä sen ympärillä olevan yhteisön kehittämä, alun perin vuonna 2013 julkaistu käyttöliittymien tekoon kehitetty JavaScript-sovelluskirjasto. Reactia käytetään perustana kehittäessä Single-Page -applikaatioita tai mobiiliapplikaatioita. Mobiiliapplikaatioita kehitetään käyttäen React Nativea, joka on Reactiin pohjautuva sovelluskehys mobiiliapplikaatioita varten. Seuraavissa kappaleissa puhun Reactin merkittävimmistä ominaisuuksista.

2.2.1 Deklaratiivisuus

Reactin avulla voi luoda helposti interaktiivisia käyttöliittymiä. Jokaista sovelluksen tilaa varten voidaan luoda oma näkymä. React päivittää ja renderöi tehokkaasti vain ne komponentit, joiden tiedot muuttuvat. Tämä tekee React-sovelluksista kevyitä ja nopeita. Deklaratiiviset näkymät tekevät koodista helpommin luettavaa ja korjattavaa.

```
1  import React from 'react'
2  import ReactDOM from 'react-dom/client'
3
4  const App = () => (
5    <div>
6      <p>Hello world</p>
7    </div>
8  )
9
10 ReactDOM.createRoot(document.getElementById('root')).render(<App />)
```

Kuva 3: Esimerkki deklaratiivisesta koodista Reactilla.

2.2.2 Komponenttiajattelu

React-koodi koostuu komponenteista. Komponentit voidaan renderöidä tiettyyn DOM-elementtiin sivustolla React DOM -kirjaston avulla. Komponenttia renderöitäessä komponenttien välisiä arvoja siirretään "props"-ominaisuuksien avulla.

```
import React from 'react'
import ReactDOM from 'react-dom/client'

const App = () => (
  <div>
    <Hello name='Erkki Esimerkki'/>
  </div>
)

const Hello = (props) => {
  return (
    <div>
      <p>Hello {props.name}</p>
    </div>
  )
}
```

Kuva 4: Esimerkki propsien käytöstä React-sovelluksessa

Kuva 4 näytetty esimerkki esittäisi verkkosivulla tekstin ”Hello Erkki Esimerkki”.

2.2.3 Virtuaalinen DOM

Yksi merkittävimmistä ominaisuuksia on virtuaalisen dokumenttiobjektimallin (DOM) eli virtuaalisen DOM:n käyttö. React päivittää selaimen näyttämää DOM:ia tehokkaasti, luomalla muistissa olevan tietorakenteiden välimuistin, ja laskemalla näiden erotuksia. Tätä prosessia kutsutaan yhteensovittamiseksi. Tämän ansiosta ohjelmoija voi kirjoittaa koodia ikään kuin koko sivu renderöitäisiin jokaisen muutoksen yhteydessä, kun taas React-kirjastot renderöivät vain ne osakomponentit, jotka todella muuttuvat. Tämä valikoiva renderöinti tarjoaa merkittävän suorituskykyparannuksen. Se säästää vaivaa CSS-tyylin, sivun asettelun ja koko sivun renderöinnin uudelleenlaskennalta.

2.2.4 JSX-määrittelykieli

React käyttää JSX:ää, eli JavaScript Syntax Extensionia. JSX yhdistää HTML:n ja JavaScriptin yhdeksi kehittäjille tutuksi syntaksiksi, joka tekee komponenttien luomisesta nopeaa ja helppoa. JSX:n avulla vähennetään kokonaistiedostomäärää sovelluksissa, mikä selkeyttää koko sovelluksen tiedostorakennetta. Kun perinteiset sovellukset kasvavat, kasvaa myös tiedostojen määrä. JSX Korvaa perinteisesti tehtävät HTML-, CSS- ja JavaScript-tiedostojen luonnit yhdeksi ainoaksi tiedostoksi, jotka sisältävät nämä kaikki. React ei kuitenkaan vaadi käytettäväksi JSX:ää, mutta kehittäjät suosittelevat sen käyttöä. Esimerkkejä JSX:stä ovat kuvissa 3 ja 4 esitetty koodi.

2.2.5 Oppimiskäyrä

React on helppo oppia ja käyttää. Sen mukana tulee runsaasti dokumentaatiota, opetusohjelmia ja koulutusresursseja. Kuka tahansa JavaScript-taustainen kehittäjä voi helposti ymmärtää ja aloittaa verkkosovellusten luomisen Reactilla muutamassa päivässä.

React sopii myös uusille kehittäjille. Koska React manipuloi itse DOMia, ei kehittäjän itsessään tarvitse käyttää JavaScriptiä DOMin manipuloimiseen. Aloittelija voisi periaatteessa hypätä yli JavaScriptin DOMin manipuloinnin kokonaan, ja hän pärjäisi silti hyvin React maailmassa.

2.2.6 Suosio

Reactin julkaisusta lähtien, sen suosio on kasvanut tasaisesti, ja tänä päivänä se onkin maailman suosituin sovelluskirjasto. Tämä näkyy myös työmarkkinoilla, joista lisää luvussa 3.

Kuvio 1: Google Trends hakutulokset koko maailma 2014-2022

Google Trends –kuvaajissa vaaka-akselilla näkyy aikajana. Pysty-akselilla näkyy prosenttiluku, milloin hakutulokset ovat olleet suurimmillaan. Kuvaajalla näkyvissä kahdessa ”huomioitavaa” kohdassa, Google ilmoittaa, että se on tehnyt parannuksia tietojen keräämiseen. Nämä parannukset tehtiin ensin 1.1.2016 ja toisen kerran 1.1.2022.

2.3 Angular-sovelluskehys

Angular on Googlen kehittämä avoimen lähdekoodin verkkosovelluskehys. Alun perin Angular julkaistiin vuonna 2010, tuolloin nimellä AngularJS. Google julkaisi uuden version Angularista vuonna 2014, jota se kutsui nimellä Angular 2, ja myöhemmin kaikkia Angular 2 ja siitä uudempia versioita kutsutaan pelkästään Angulariksi. Jyrkät muutokset AngularJS:stä Angular 2:een jakoivat kehittäjien mielipiteitä, ja tämä vaikutti myös Angularin suosioon. Lisää Angularin suosiosta luvussa 2.3.7.

1.1.2022 alkaen, Google ei enää päivitä vanhaa AngularJS:ää, ja Angularin tiimi suositteleekin päivittämään kaikki vanhat projektit uuteen versioon.

Seuraavissa kappaleissa käyn läpi Angularin huomioitavia ominaisuuksia.

2.3.1 Toiminnallisuus suoraan laatikosta

Angularin oletusasetukset sisältävät kaiken tarvitseman, jotta sillä voi luoda toimivia ja kokonaisia verkkosovelluksia. Angular sisältää työkalut, jotka huolehtivat reitityksestä, jotta voit helposti hakea sovelluksessasi haluamasi tiedot. Angularin valmiiksi konfiguroitu ympäristö ei ainoastaan auta kehitystyössä, vaan helpottaa myös testausta.

Angular ei vaadi mitään kolmannen osapuolen kirjastoja luodaksesi perustoiminnallisuutta sovellukseesi. Angularin kehittäjätiimin vahvistamia kirjastoja on todella paljon. Kirjastoilta voi odottaa parempaa kyberturvallisuutta ja korkeampaa koodin laatua.

2.3.2 TypeScript-ohjelmointikieli

Angularia täytyy ohjelmoida TypeScript-ohjelmointikielellä. Myös Reactia voi kirjoittaa TypeScriptillä, mutta sitä ei vaadita. TypeScript on Microsoftin kehittämä ohjelmointikieli. Se on JavaScriptin yläjoukko, eli se on kirjoitettu JavaScriptin päälle. Tämän vahvasti tyypitetyn kielen tärkein etu on, että se auttaa kehittäjiä pitämään koodi puhtaana ja ymmärrettävänä. Virheet on helpompi havaita ja poistaa, kun yleiset virheet näkyvät kirjoitettaessa. Tämä nopeuttaa virheenkorjausta ja helpottaa myös suuren koodikannan ylläpitoa - mikä on erityisen hyödyllistä suurissa yritysprojekteissa.

2.3.3 Johdonmukaisuus

Toisin kuin React, Angular on täysin responsiivinen verkkosuunnittelukehys. Keskeinen piirre on, että komponentin, palvelun tai moduulin luomiseen on yksi ehdotettu tapa.

Käytännössä tämä luo suurta johdonmukaisuutta koko koodikantaan - mikä on tärkeä tavoite. Yhdenmukaisuuden lisäämiseksi entisestään Angularin kehittäjätiimi valmisti CLI-työkalun, eli komentorivityökalun, jonka avulla voidaan luoda tiettyjä toistettavia koodilohkoja komentoriviltä. Myös sovelluskehyksen dokumentaatio toimii kehittäjille ajantasaisena viitteenä.

2.3.4 Ylläpidettävyys

Angular tukee koodin erinomaista ylläpidettävyyttä monin tavoin. Ensinnäkin, kun siirrytään yhdestä pääversiosta toiseen, kaikki Angulariin liittyvät paketit päivitetään samaan aikaan.

Päivittäminen on helppoa ja onnistuu yleensä vain yhdellä komennolla, 'ng update'. Tämä tarkoittaa sitä, että kehittäjien ei tarvitse käyttää aikaa miettiäkseen, onko jokin tietty paketti päivitetty tai hajoaako olemassa oleva sovellus uuden päivityksen myötä.

Koska Angularkin toimii komponenttiajattelun mukaan, voidaan komponentit helposti irroittaa ja korvata parannetuilla toteutuksilla.

2.3.5 Angular Material-komponenttikirjasto

Angular Material on kokoelma valmiita, hyvin testattuja käyttöliittymäkomponentteja ja -moduuleja, jotka noudattavat Googlen Material Design -periaatteita. Se sisältää erilaisia käyttöliittymäkomponentteja, kuten navigointikuvioita, lomakeohjaimia, painikkeita ja indikaattoreita. Komponentit on mukautettu eri selaimille sopiviksi, hyvin dokumentoitu ja kirjoitettu uusimpien ohjeiden pohjalta.

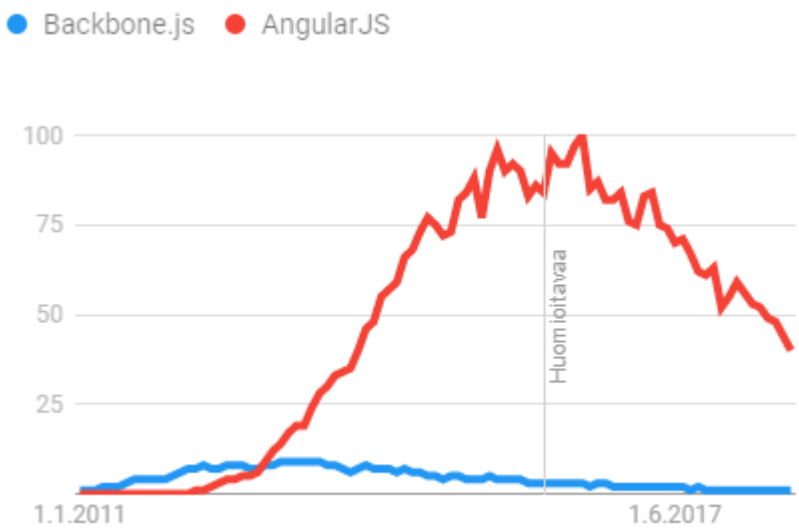
Angular-yhteisön kehittämät moduulit yksinkertaistavat tiimien suunnittelun työnkulkua, jolloin kehittäjät voivat lisätä uusia elementtejä ja kehittää sovelluksia nopeasti ja suorituskykyyn mahdollisimman vähän vaikuttaen.

2.3.6 Oppimiskäyrä

Angular on vaikea oppia. Yksi sen hyvistä puolista on myös yksi sen heikkouksista. Koska Angular sisältää kaiken tarvittavan suoraan asennettuna, se vaatii myös tarkkaa perehtymistä sen syntaksiin. Angular vaikuttaa voimakkaasti siihen, miten projekti täytyy rakentaa. Angularin tarjoaman parhaan mahdollisen hyödyn saaminen tarkoittaa moduulien, komponenttien, direktiivien, palveluiden, putkien jne. erojen opettelua. Nämä ovat Angular-sovelluksen olennaisia osia.

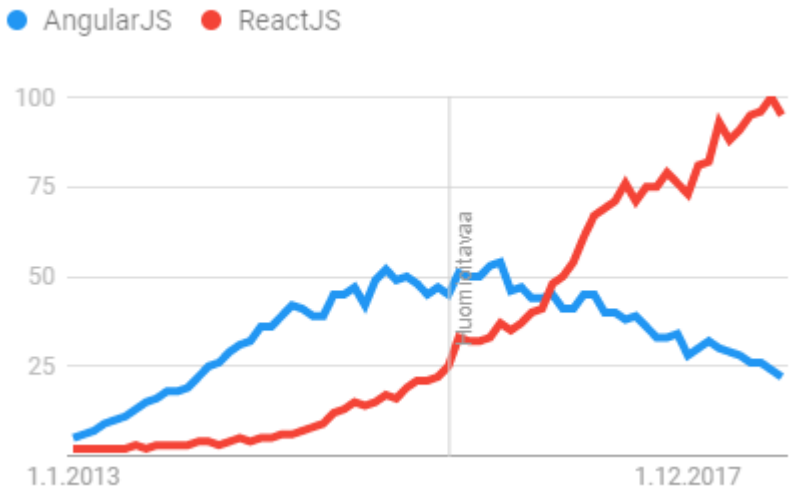
2.3.7 Suosio

Single-page applikaatioiden suosio alkoi nousta rajusti vuonna 2011. Tuolloin aallon harjalla heilui Backbone.js. Vuonna 2012 AngularJS:n saama päivitys 1.0.0 nosti sen kuitenkin suosituimmaksi verkkosovelluskehykseksi.

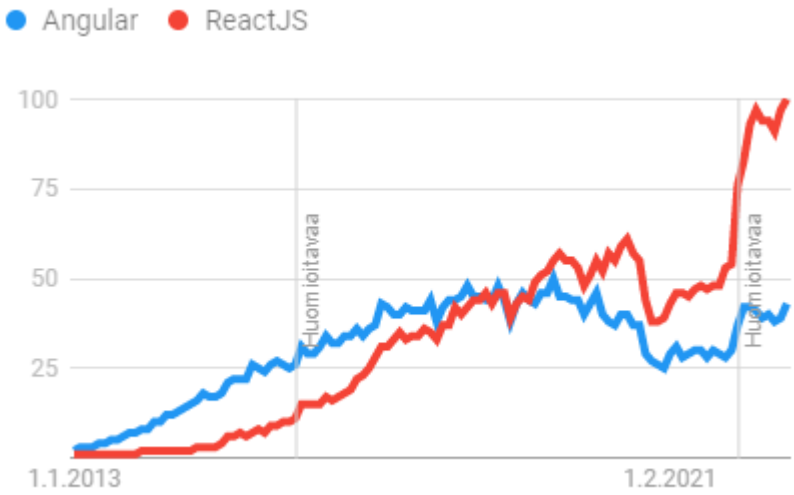


Kuvio 2: Google Trends kuvaaja AngularJS:n suosion kasvusta 2012 alkaen

Luvussa 2.3 mainittu muutos suosiossa tapahtui vuonna 2014, kun AngularJS sai päivityksen, jota kutsuttiin Angular 2:ksi. Muutokset olivat suuria, eivätkä kehittäjät silloin pitäneet suunnasta, johon uusi Angular oli menossa (AgingCoder, 2014). Lisäksi Google linjasi, että vanhat AngularJS sovellukset eivät ole ylöspäin yhteensopivia uusien Angular 2 –sovellusten kanssa. Nämä muutokset, Reactin nousu, sekä myös muiden verkkosovelluskehysten ilmaantuminen vaikuttivat Angulariin niin, että sen suosio ei enää koskaan palautunut aiemmalle tasolleen.



Kuvio 3: Google Trends kuvaaja Angularin ja Reactin suosiosta



Kuvio 4: Google Trends kuvajaa, jossa otetaan huomioon Angularin nimimuutos

Johdannossa nähty Kuva 1 kertoo, että Angular on edelleen suosittu sovelluskehys. Se kilpailee suosiosta Vuen, ja muiden nousevien sovelluskehysten kanssa. Vuonna 2021 React oli ehdoton suosikki, ja vuonna 2022 sen odotetaan olevan entistä suositumpi. Angular saa edelleen jatkuvasti päivityksiä joka pitää sen pinnalla. Lisäksi Google käyttää sitä myös omissa järjestelmissään, mikä auttaa sen näkyvyyttä ja joka vaikuttaa merkittävästi sen suosioon ja myös olemassaoloon.

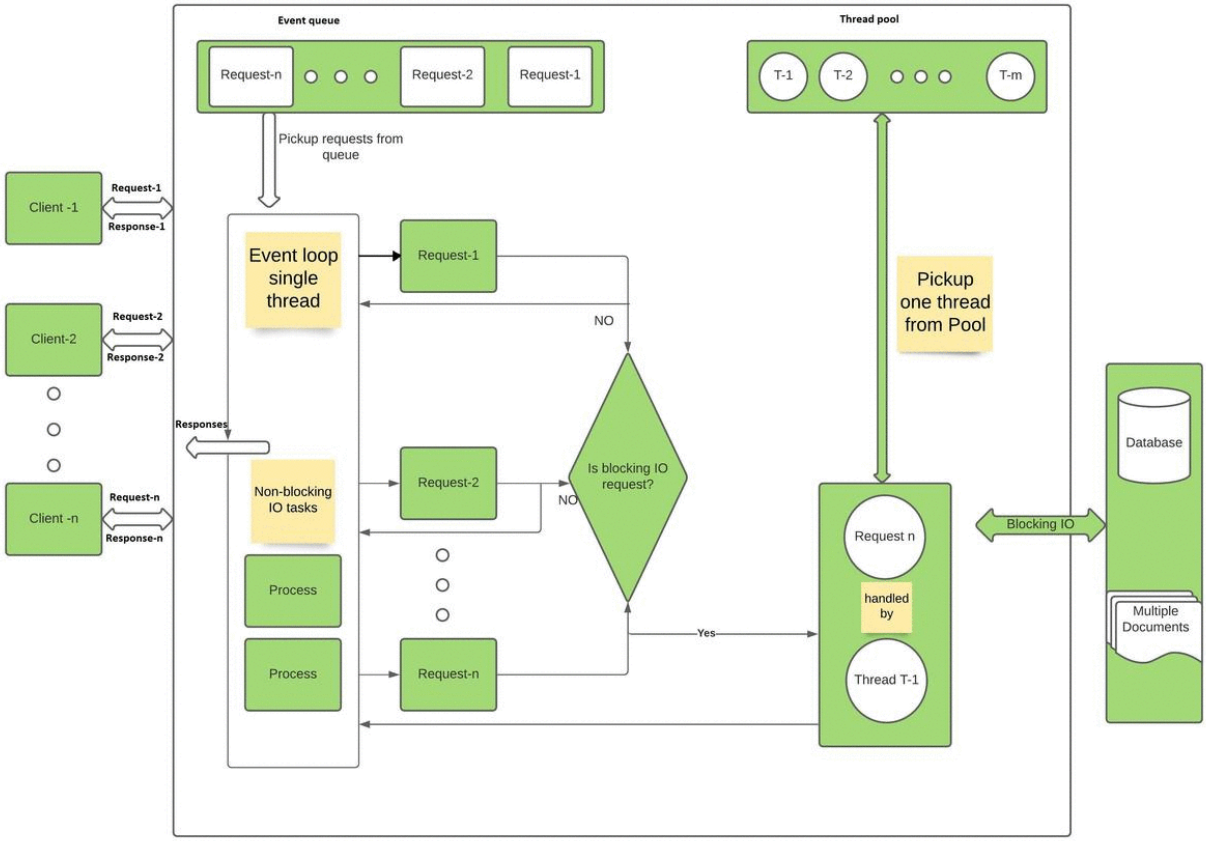
2.4 Muut taustalla toimivat prosessit

Yleensä verkkosovellus tarvitsee myös muita prosesseja toimiakseen. Opinnäytetyötä varten tekemissäni sovelluksissa käytin palvelinpuolen ohjelmoinnissa Node.js:ää, ja tietokantana MongoDB:tä. Seuraavissa kappaleissa kerron lyhyesti Node.js:stä sekä MongoDB:stä.

2.4.1 Node.js-ajoympäristö

Node.js on avoimen lähdekoodin JavaScript-ajoympäristö, joka toimii JavaScriptin V8-moottorilla. Se suorittaa JavaScript-koodia verkkoselaimen ulkopuolella, ja on suunniteltu skaalautuvien verkkosovellusten rakentamiseen. Node.js pohjautuu yksisäikeiseen tapahtumasilmukkamalliin, joka on saanut inspiraationsa mm. Ruby-kielen tapahtumakoneesta (nodejs.dev, N.d). Node.js:n avulla kehittäjät voivat käyttää JavaScriptiä komentorivityökalujen kirjoittamiseen sekä JavaScript-koodin suorittamisen palvelinpuolella dynaamisen verkkosivun sisällön tuottamiseksi, ennen kuin sivu lähetetään käyttäjän verkkoselaimelle. Node.js edustaa "JavaScript everywhere" –paradigmaa (Scott, 2020) joka yhdistää verkkosovellusten kehittämisen yhden ohjelmointikielen ympärille sen sijaan, että palvelin- ja asiakaspuolen ohjelmistoille käytettäisiin eri kieliä.

Node.js:n yksi tärkeimmistä tehtävistä on hoitaa palvelinpuolen API-kutsut, hakea ja siirtää tietoja tietokantaan ja tietoja käyttöliittymälle. Node.js:n vahvuus on se, että se toimii yhdessä prosessissa tapahtumasilmukkamallin mukaan, eikä jokaista pyyntöä varten luoda uutta säiettä. Tämän ansiosta palvelinkone käyttää mahdollisimman vähän muistia ja resursseja.



Kuva 5: Node.js-applikaation yksisäikeinen tapahtumasilmukkamalli (geeksforgeeks, 2021)

Kun Node.js suorittaa I/O-operaation, kuten lukemisen verkosta, pääsyn tietokantaan tai tiedostojärjestelmään, niin se jatkaa operaatiota sen sijaan, että se pysäyttäisi säikeen ja tuhlaisi prosessorijaksoja odottamiseen. Näin Node.js voi käsitellä tuhansia samanaikaisia yhteyksiä yhdellä palvelimella ilman, että säikeiden samanaikaisuuden hallinnasta aiheutuisi taakka, joka voisi merkittävästi hidastaa sovellusta.

2.4.2 MongoDB-tietokanta

MongoDB on dokumenttitietokanta. Kuten avain-arvotietokannat, myös dokumenttivarastot perustuvat arvojen tallentamiseen avainten mukaan. Arvot tai dokumentit, kuten niitä kutsutaan dokumenttivarastojen yhteydessä, voivat kuitenkin olla hyvin monimutkaisia kokonaisuuksia, jotka sisältävät kenttiä, joiden arvot voivat olla joko tavallisia arvoja, kuten numeroita ja merkkijonoja, tai muita kokonaisuuksia. Toisin kuin avain-arvotietokannat, dokumenttivarastot "näkevät" tietokantaan tallennettuihin dokumentteihin ja mahdollistavat kyselyjen tekemisen tallennettujen dokumenttien sisällölle.

MongoDB:ssä käytetään tiedon lähetys- ja esitysmuotona lähes aina JSON:ia. JSON eli JavaScript Object Notation on formaatti, jota käytetään tietojen esittämiseen. Se otettiin käyttöön 2000-luvun alussa osana JavaScriptiä, ja siitä on vähitellen tullut yleisin tekstipohjaisen tiedon kuvaus- ja vaihtomuoto. (Tyson, 2022)

3 TYÖMARKKINAT

Seuraavaksi tarkastelen työmarkkinoita Reactin ja Angularin osalta. Haasteena tarkkojen määrien hakemisessa on se, että usein työnantajat hakevat samaan tehtävään molemmilla osaamisilla varustettua kehittäjää. Tuloksissa on esitettyinä siis myös haut, joissa esiintyy molemmat hakusanat.

Suoritin työpaikkahaut käyttäen useaa eri lähdettä. Lähteinäni käytin Monsterin, LinkedInin, TE-toimiston, Duunitorin sekä Oikotien Työpaikat palveluita. Huomionarvoista on myös, että jotkut työilmoitukset ovat useassa palvelussa, joten sama työpaikka voi esiintyä laskuissa useamman kerran. Haut on tehty 26.8.2022 – 5.9.2022 välisenä aikana

Monster	280
LinkedIn	721
TE-Toimisto	2
Duunitori	332
Oikotie Työpaikat	104
Yhteensä	**1439**

Taulukko 2: Reactin työpaikat Suomessa

Monster	50
LinkedIn	328
TE-Toimisto	1
Duunitori	156
Oikotie Työpaikat	47
Yhteensä	**582**

Taulukko 3: Angularin työpaikat Suomessa

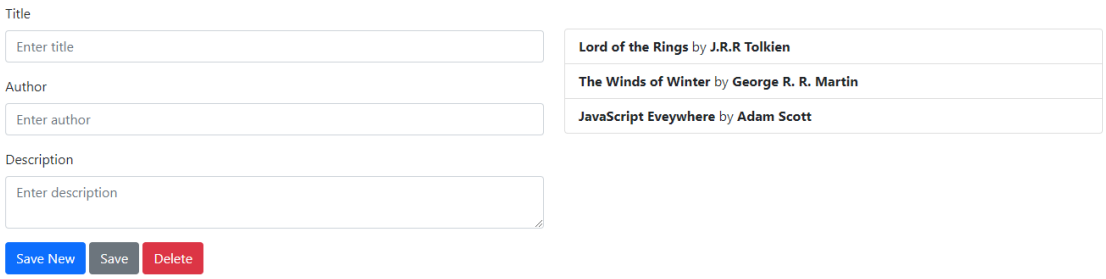
Kuten taulukoista näkyy, esimerkiksi LinkedInissä Reactin työmarkkinat ovat yli kaksinkertaiset Angulariin verrattuna. (LinkedIn, 2022) Myös Duunitorissa Reactin työtehtävämäärä on kaksinkertainen Angulariin verrattuna (Duunitori, 2022).

Yksi osasyy suosioiden välille on niiden oppimiskäyrät. Reactille on helpompi saada kehittäjiä, koska se on nopeampi oppia ja projektiin pääsee helpommin mukaan, kuin esimerkiksi Angular-projekteihin. Tutkiessani eri työpaikkoja, useinkaan henkilöiltä ei vaadittu paljon aiempaa React-osaamista. Angular-tehtäviin taas vaadittiin yleensä useamman vuoden kokemus tästä sovelluskehyksestä.

Useamman tehtävän hakuvaatimuksissa oli yleensä kokemusta toisesta, tai molemmista sovelluskehyksistä. Uskon tämän johtuvan siitä, että molempien sovelluskehysten taustalla on sama, komponenttinen ajattelumalli, jonka tietämys ja osaaminen helpottavat toisen sovelluskehyksen oppimista ja ymmärtämistä.

4 KIRJA-SOVELLUS

Tätä opinnäytetyötä varten tein kaksi samanlaista sovellusta. Toisen käyttäen Reactia, ja toisen käyttäen Angularia. Valitsin aiheekseni kirja-sovelluksen siksi, että siinä pääsen hyödyntämään perinteisiä CRUD (Create, Read, Update, Delete) –operaatioita. Käytin molemmissa sovelluksissa Bootstrap-tyylikirjastoa sivuston tyylittelyä varten. Molempien sovellusten ulkoasu on identtinen, mutta toteutus on tehty käyttäen näitä kahta sovelluskehystä.



Kuva 6: Verkkosivun ulkoasu

4.1 React-toteutus

Kuva 3 tapaan aloitamme React-sovelluksen rakentamisen renderöimällä App-komponentin, joka toimii koko sovelluksen pääsivuna.

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import './index.css';
4  import App from './App';
5  import 'bootstrap/dist/css/bootstrap.min.css'
6
7  const root = ReactDOM.createRoot(document.getElementById('root'));
8  root.render(
9  <React.StrictMode>
10     <App />
11   </React.StrictMode>
12 );
```

Kuva 7: React-sovelluksen alustus

ReactDOM luo pohjaelementin nimeltään ”root”, johon se yhdistää App-komponentin, jonka käyttäjä näkee.

4.1.1 Tila React-sovelluksissa

Tila on sisäänrakennettu React-objekti, jota käytetään sisältämään komponenttia koskevia tietoja. Komponentin tila voi muuttua ajan myötä. Aina kun se muuttuu, komponentti renderöityy uudelleen. Tilan muutos voi tapahtua vastauksena käyttäjän toimintaan tai järjestelmän tuottamiin tapahtumiin, ja nämä muutokset määrittävät komponentin käyttäytymisen ja sen, miten se renderöidään.

```
const App = () => {
  const [bookList, setBookList] = useState([])
  const [title, setTitle] = useState("")
  const [author, setAuthor] = useState("")
  const [description, setDescription] = useState("")
  const [selectedBook, setSelectedBook] = useState([])
  const [loading, setLoading] = useState(false)
```

Kuva 8: Kirja-sovelluksen tilat

Tila voidaan päivittää tapahtumakäsittelijöiden, palvelimen vastausten tai ominaisuuksien muutosten perusteella. Tämä tehdään setState()-metodilla, joka on osa Reactin useState-koukkua. SetState()-metodi asettaa kaikki komponentin tilaan tehdyt päivitykset jonoon ja käskee Reactia renderöimään komponentin ja sen alikomponentit uudelleen päivitetyllä tilalla.

```
15    useEffect(() => {
16      const retrieveBooks = async() => {
17        try {
18          setLoading(true)
19          const response = await bookService.retrieveAllBooks()
20          setBookList(response)
21        } catch(err) {
22          console.log("Error: ", err)
23        }
24      }
25      setLoading(false)
26      retrieveBooks()
27    }, [])
```

Kuva 9: Tilojen muuttaminen

Kuva 9 nähdään esimerkki tilan muutoksesta. Sovelluksen latauduttua sovellus hakee ensin kaikkien kirjojen tiedot palvelimelta rivillä 19 käyttäen Reactin useEffect-koukkua. Sitten vastauksena saadut kirjat asetetaan bookList-tilaan käyttämällä setBookList()-metodia, joka on bookListin setState-metodi. Näin lista kirjoista tulee käyttäjän nähtäville, heti kun sivu latautuu.

Tilan muuttamista käytetään myös uusien kirjojen tallentamiseen. Kuva 10 näytetään, miten luomme HTML-lomakkeen. Jokaisella eri lomakkeen osalla on tilan muutosta vahtiva olio onChange. Kun lomakkeeseen kirjoittaa, käytetään eri lomakkeen, esimerkiksi kirjan tekijän tilaa muokkaavaa onAuthorChange()-metodia, joka kirjoittaa tilaksi kirjan tekijän.

```
89    return (
90      <Form>
91      <Form.Group className="mb-3">
92        <Form.Label>Title</Form.Label>
93        <Form.Control
94        type="text"
95        placeholder="Enter title"
96        value={title ?? ""}
97        onChange={(event) => onTitleChange(event.target.value)}
98        required />
99      </Form.Group>
100
101     <Form.Group className="mb-3">
102       <Form.Label>Author</Form.Label>
103       <Form.Control
104       type="text"
105       placeholder="Enter author"
106       value={author ?? ""}
107       onChange={(event) => onAuthorChange(event.target.value)}
108       required  />
109     </Form.Group>
110
111     <Form.Group className="mb-3">
112       <Form.Label>Description</Form.Label>
113       <Form.Control
114       as="textarea"
115       type="text"
116       placeholder="Enter description"
117       value={description ?? ""}
118       onChange={(event) => onDescriptionChange(event.target.value)}
119       required  />
120     </Form.Group>
```

Kuva 10: Lomakeen esitys React-koodissa

4.1.2 Tietojen haku palvelimelta React-sovelluksissa

Käytän sovelluksessani tietojen hakemiseen Axios-kirjastoa. Axios on lupauksiin perustuva HTTP-palvelinkirjasto JavaScriptille. Se pystyy tekemään HTTP-pyyntöjä selaimesta ja käsittelemään pyyntö- ja vastausdatan muuntamisen.

```
1  import axios from 'axios'
2
3  const baseUrl = "http://localhost:3001/api/books"
4
5  const retrieveAllBooks = async() => {
6      const response = await axios.get(baseUrl)
7      return response.data
8  }
9
10 const addNewBook = async(newBook) => {
11     const response = await axios.post(baseUrl + "/add", newBook)
12     return response.data
13 }
14
15 const updateBook = async(id, updatedBook) => {
16     const response = await axios.put(baseUrl + "/update/" + id, updatedBook)
17     return response.data
18 }
19
20 const deleteBook = async(id) => {
21     const response = await axios.delete(baseUrl + "/delete/" + id)
22     return response.data
23 }
24
25 // eslint-disable-next-line
26 export default {
27     retrieveAllBooks, addNewBook, updateBook, deleteBook
28 }
```

Kuva 11: Palvelinhaut käyttäen Axios-kirjastoa

Rivillä 3 näkyvä osoite on palvelimen osoite, josta tiedot haetaan. Perinteisesti tähän kohtaan tulisi jonkin serverin osoite, mutta koska tässä tapauksessa palvelimena toimii Node.js-sovellus, toimii serverinä localhost, eli paikallisen koneen osoite. Lisää Node.js-toteutuksesta luvussa 4.3.

Rivillä 6 Axios tekee http-get-haun serverin osoitteeseen, joka palauttaa arvonaan kaikki tietokannasta löytyvät kirjat. Näin kirjat saadaan Kuva 8 nähtävään bookList-tilaan, ja jotka sitten näytetään käyttäjälle.

4.1.3 Tietojen lisäys, poisto ja muokkaaminen

Kuva 11 nähdään myös axioksen tekemät http-kutsut jotka liittyvät lisäämiseen, poistamiseen ja muokkaamiseen. Jokainen tapahtuma lähetetään omaan osoitteeseensa. Esimerkiksi kirjan lisäämiseen tarvitsee palvelimen osoitteen perään lisätä parametri ”/add”. Kun palvelin saa post-kutsun tähän osoitteeseen, jonka mukana tulee kirja-olio, tietää palvelin lisätä tietokantaan uuden kirja-olion.

Poistamiseen ja muokkaamiseen toimii sama idea. Nyt lisänä palvelimen osoitteelle täytyy lisätä sen kirjan id, jonka halutaan muutettavan tai poistettavan. Muutettavan kirjan osoitteeseen täytyy lähettää muutetun kirjan data. Poistamiseen riittää pelkkä id.

4.2 Angular-toteutus

Angularin-sovellusten aloittaminen on lähes identtinen React-sovellusten kanssa. Ensin päämoduuli, eli AppModule lisätään sovellukseen, jonka käyttäjä näkee.

```
1  import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2
3  import { AppModule } from './app/app.module';
4
5  platformBrowserDynamic().bootstrapModule(AppModule)
6    .catch(err => console.error(err));
```

Kuva 12: Angular-sovelluksen käynnistys

```
import { CUSTOM_ELEMENTS_SCHEMA, NgModule, NO_ERRORS_SCHEMA } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { HttpClientModule } from '@angular/common/http';

import { AppRoutingModule } from './app-routing.module';
import { RootComponent } from './root.component';
import { BookComponent } from './book/book.component';

@NgModule({
  declarations: [
    RootComponent,
    BookComponent,
  ],

  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    FormsModule,
    ReactiveFormsModule,

  ],
  bootstrap: [RootComponent],
  schemas: [CUSTOM_ELEMENTS_SCHEMA, NO_ERRORS_SCHEMA]
})
export class AppModule { }
```

Kuva 13: AppModule-moduulin toteutus

AppModule sisältää kaikki komponentit, jotka ovat siihen yhteydessä. Declarations-osioon lisätään ne komponentit jotka ovat osa AppModulea. Lisäksi AppModuleen tulee lisätä sen imports-osioon kaikki sovelluksessa käytettävät Angularin sisäiset kirjastot, kuten HttpClientModule, FormsModule jne. Bootstrap-osuuteen lisätään se komponentti, johon AppModulen tulee kiinnittyä. Tässä tapauksessa AppModule kiinnitetään RootComponent-komponenttiin, joka on se mikä näytetään käyttäjälle.

4.2.1 Tietojen käsittely Angular-sovelluksissa

Kuten kappaleessa 2.3.1 mainittiin, Angularin mukana tulee valmiina monia kirjastoja, joita kehittäjät voivat käyttää. Yksi näistä on HttpClient.

Muutoin haku on lähes identtinen Axioksen kanssa. HttpClient tekee http-operaatiot palvelimen osoitteeseen, joka palauttaa tietokannasta saadut tiedot. Määriteltyjä metodeja voidaan nyt kutsua Angular-koodissa, missä ikinä sitä tarvitaankaan.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';


@Injectable({
  providedIn: 'root'
})
export class BookService   {
  private readonly URL_CREATE_BOOK: string = 'http://localhost:3001/api/books/add';
  private readonly URL_FETCH_BOOKS: string = 'http://localhost:3001/api/books';
  private readonly URL_UPDATE_BOOK: string = 'http://localhost:3001/api/books/update/';
  private readonly URL_DELETE_BOOK: string = 'http://localhost:3001/api/books/delete/';

  constructor(private http: HttpClient, private router: Router) { }

  public async fetchBooks() {
    return this.http.get(this.URL_FETCH_BOOKS, {});
  }

  public async saveBook(book: any) {
    return this.http.post(this.URL_CREATE_BOOK, book);
  }

  public async updateBook(book: any) {
    return this.http.put(this.URL_UPDATE_BOOK + book.id, book);
  }

  public async deleteBook(bookId: any) {
    return this.http.delete(this.URL_DELETE_BOOK + bookId);
  }
}
```

Kuva 14: HttpClient tekee http-kutsut palvelimelle

4.2.2 Näkymien esittäminen

Yksi tapa käyttää Bootstrappia on Kuva 10 näkyvään tapaan, eli lisäämällä React-komponentteja, joille annetaan erilaisia parametreja. Angularissa näkymien teko on hoidettu perinteisempään tapaan, käyttämällä html-elementtejä.

```
1  <div class="modal-header">
2    <h4 class="modal-title">Book</h4>
3  </div>
4  <div class="modal-body" *ngIf="ready">
5    <form [formGroup]="bookForm" name="book_form">
6      <div class="row">
7        <div class="col-6 mt-3">
8          <div class="list-group">
9            <div (click)="setForm(book,selected)"  *ngFor="let book of books" id="demo" class="list-group-item"   style="cursor: pointer;" #selected>
10             <b>{{book?.title}}</b> by <b>{{book?.author}}</b>
11           </div>
12         </div>
13       </div>
14       <div class="col-6">
15         <div class="form-group" hidden>
16           <label for="id" class="label">id</label>
17           <input type="text" class="form-control" id="id" aria-describedby="emailHelp" placeholder="id"
18             formControlName="id">
19         </div>
20         <div class="form-group">
21           <label for="title" class="label">Title</label>
22           <input type="text" class="form-control" id="title" aria-describedby="emailHelp" placeholder="Title"
23             formControlName="title">
24         </div>
25         <div class="form-group">
26           <label for="author" class="label">Author</label>
27           <input type="text" class="form-control" id="author" placeholder="Author" formControlName="author">
28         </div>
29         <div class="form-group">
30           <label for="description" class="label">Description</label>
31           <textarea type="text" class="form-control" id="description" rows="5" placeholder="Description"
32             formControlName="description"></textarea>
33         </div>
34         <button type="submit" class="btn btn-danger shadow-0 mt-2" (click)="delete()">Delete</button>
35         <button type="submit" class="btn btn-secondary mx-1 shadow-0 mt-2" (click)="update()">Save</button>
36         <button type="submit" class="btn btn-primary shadow-0 mt-2" (click)="saveNew()">Save New</button>
37       </div>
38     </div>
39   </form>
40 </div>
```

Kuva 15: Näkymien esittäminen Angularissa

Tässä Bootstrap toimii niin, että jokaiselle diville, eli osanäkymälle, annetaan luokan nimeksi erilaisia bootstrap parametreja, jotka muokkaavat osanäkymää halutunlaiseksi. Tässä kohtaa näemme myös yhden osan Angularia. Rivillä 4 näkyvä ngIf=”ready” direktiivi kertoo, että mikäli hipsujen sisään asetettu muuttuja ready saa arvon true, silloin tämä osanäkymä näytetään.

Rivillä 9 näkyvä direktiivi ngFor käy läpi listan nimeltä books, jolloin se saa käyttöönsä yksittäisen objektin nimeltä book. Näin jokaisen book-objektin tiedot voidaan esittää näkymässä. Riveillä 18, 23, 27 ja 32 näkyvät formControlName sarakkeet ovat osa Angularin FormsModulea, jolla hallitaan käyttöliittymien lomakkeita.

4.2.3 Angular-komponenttien rakentaminen

Angular-komponenttien valmistus muistuttaa perinteistä OOP-mallia. Ensin luodaan kompenttia vastaava objekti, jolla on muuttujia sekä constructor, eli alustusoperaatio. Lisäksi Angular-sovelluksissa tulee jokaisella komponentilla olla sisäänrakennettu metodi, ngOnInit(), joka ajetaan kun komponentti rakentuu ensimäisen kerran.

```
1  import {Component, OnInit} from "@angular/core";
2  import { FormBuilder } from "@angular/forms";
3  import { Book } from "../services/book.model";
4  import { BookService } from "../services/book.service";
5
6  @Component({
7    selector: 'app-book',
8    templateUrl: './book.component.html',
9    styleUrls: ['./book.component.css'],
10 })
11 export class BookComponent implements OnInit  {
12
13   public bookForm: any;
14   public books: Book[];
15   public ready: boolean = false;
16    constructor(
17        private formBuilder:FormBuilder,
18        private bookService:BookService
19    ) {}
20
21    ngOnInit() {
22        this.initForm();
23        this.fetchBook();
24    }
```

Kuva 16: Angular-komponentin rakentaminen

Tämän jälkeen komponentille voidaan rakentaa eri metodeja, joita se käyttää. Esimerkiksi täällä voimme kutsua aiemmin luotuja HttpClientin metodeja. Angular-komponenttien metodit luodaan kuten perinteiset metodit.

```
35      saveNew() {
36          const book: Book = this.bookForm.value;
37
38          const response: any =  this.bookService.saveBook(book);
39          response.then((data: any) => {
40             data.subscribe((bookData: any) => {
41               this.books.push(bookData)
42               });
43           });
44          this.setForm("", null)
45        }
```

Kuva 17: Metodien luominen Angular-komponentille

4.3 Palvelimen toteutus Node.js:llä

Palvelinpuoti toteutetaan siis Node.js:llä. Jotta palvelimen rakentaminen olisi nopeampaa ja helpompaa, käytän Node.js:n lisäksi Express.js-sovelluskehystä.

Express.js on sovelluskehys, joka on tehty Node.js:ää varten. Se on avoimen lähdekoodin sovelluskehys, joka tarjoaa vankat ominaisuudet ja työkalut skaalautuvien backend-sovellusten kehittämiseen. Se tarjoaa reititysjärjestelmän ja yksinkertaistettuja ominaisuuksia, joiden avulla voidaan kehittää tehokkaampia komponentteja ja osia sovelluksen käyttötarpeiden mukaan.

4.3.1 Node.js-sovellusten käynnistäminen

```
1  const express = require("express")
2  const app = express()
3
4  app.use(express.json())
5
6  // Start the server
7  app.listen(process.env.PORT, () => {
8      console.log("Server running on port", process.env.PORT)
9  })
```

Kuva 18: Node.js-sovelluksen käynnistys

Kaikessa yksinkertaisuudessaan, Node.js-sovellus käynnistetään Kuva 18 mukaisella toteutuksella. Ensin luodaan express-objekti, ja josta taas edelleen luokaan app-objekti. App-objekti asetetaan käyttämään expressin json-työkaluja, ja jonka jälkeen sovellus käynnistetään kuuntelemalla tiettyä verkkoporttia. Nyt sovellukseen voidaan ottaa yhteys esimerkiksi selaimella osoitteessa http://localhost:<porttinumero>.

4.3.2 Yhteys tietokantaan

Yhteys tietokantaan muodostetaan Node.js-sovelluksessa.

```
1  const express = require("express")
2  const mongoose = require("mongoose")
3  const cors = require("cors")
4  require("dotenv").config()
5
6  // Routes
7  const bookRoute = require("./routes/books")
8  const MONGO_URL = "mongodb+srv://[redacted]luster0.eruyl.mongodb.net/?retryWrites=true&w=majority"
9  const app = express()
10 app.use(express.json())
11
12 // Connect to MongoDB
13 mongoose
14 .connect(MONGO_URL, {
15     useNewUrlParser: true,
16     useUnifiedTopology: true
17 })
18 .then(() => {
19     console.log("MongoDB connected")
20 })
21 .catch((err) => {
22     console.log("Error: " + err)
23 })
24
25 app.use(cors())
26 app.use("/api/books", bookRoute)
27
28 // Start the server
29 app.listen(process.env.PORT, () => {
30     console.log("Server running on port", process.env.PORT)
31 })
```

Kuva 19: Tietokantayhteyden muodostaminen

Rivillä 2 käytetään Mongoosea, JavaScript-kirjastoa joka auttaa meitä käyttämään MongoDB-tietokantaa. Ensin alustetaan mongoose-olio, jonka jälkeen rivillä 8 alustetaan muuttuja joka sisältä tietokannan osoitteen. Rivillä 13 muodostetaan yhteys tietokantaan, jonka jälkeen tietokannasta voidaan lukea tai sinne voidaan kirjoittaa dataa.

4.3.3 Kutsut tietokantaan

```
1  const router = require("express").Router()
2  const { body, validationResult } = require("express-validator")
3  const Book = require("../models/Book")
4
5  // Get all Books
6  router.get("/", async (req, res) => {
7      try {
8          const books = await Book.find()
9          res.status(200).json(books)
10     } catch(err) {
11         res.status(500).json(err)
12     }
13 })
```

Kuva 20: Toteutus palvelimen get-pyynnölle

Kuva 19 rivillä 26 nähdään, kuinka palvelimelle lisätään reitti /api/books. Kuva 20 näytetään, miten ensin luodaan Router-objekti, joka on osa Express-sovelluskehystä. Nyt kun palvelin saa get-pyynnön osoitteeseen http://localhost:3001/api/books, se tietää palauttaa käyttäjälle kaikki tietokannasta löytyvät kirjat.

4.3.4 Mongoose-skeemat

Jotta sovellus osaa etsiä Kuva 20 tapaan tietokannasta kirja-objektit, täytyy Mongoosea varten luoda skeema. Tämä skeema on eräänlainen malli kaikille kirja-objekteille. Kaikki kirjat täytyy luodan saman kaavan mukaan.

```
1  const mongoose = require("mongoose")
2
3  const BookSchema = new mongoose.Schema(
4      {
5          title: {
6              type: String,
7              require: true,
8          },
9          author: {
10             type: String,
11             require: true
12         },
13         description: {
14             type: String,
15             require: true
16         },
17     },
18     { timestamps: true }
19 )
20
21 module.exports = mongoose.model("Book", BookSchema)
```

Kuva 21: Mongoose-skeeman toteutus

Skeemalle annetaan ensin kaikki objektin ominaisuudet, eli ne asiat mitä se sisältää. Näille ominaisuuksille voidaan vaihtoehtoisesti antaa erilaisia vaatimuksia, kuten esimerkiksi Kuva 21 kaikille annetaan ”require: true” –parametri, joka määrittää sen, että kirja-oliota luodessa nämä kentät ovat pakollisia.

Kun skeema on luotu, osaa Mongoose hakea tietokannasta kaikki tästä skeemaa vastaavat oliot, sekä lisätä ne oikeisiin dokumentteihin.

4.4 Toteutusten vertailu ja päätelmät

Kahden identtisen sovelluksen rakentaminen kahdella eri työkalulla oli silmiä avaava kokemus. Vaikka sovellukset olivat varsin pieniä kokonaisuuksia, molemmista työkaluista pystyi saamaan hyvän yleiskuvan.

Yksi Reactin selkeistä vahvuuksista on sen helppous. Kehittäminen on nopeaa ja vaivatonta. Yksinkertainen syntaksi mahdollistaa olemassa olevan koodin nopean omaksumisen ja jatkokehittämisen. Myös uuden koodin ja projektin luominen käy käden käänteessä. Koska React on niin suosittu, sen ympärillä oleva yhteisö on todella iso. Ongelmien ilmaantuessa, on äärimmäisen todennäköistä, että joku muu on jo törmännyt samaan ongelmaan. Ratkaisut ongelmiin löytyvät helposti.

Heikkouksia Reactista on vaikea löytää. React kehittyy nopeaan tahtiin, ja sen päälle kehitetyt sovelluskehykset kuten Next.js kasvattavat Reactin mahdollisuuksia entisestään. Uskon, että React tulee olemaan verkkokehityksen kärjessä myös tulevaisuudessa.

Angularilla kehittäminen on ajoittain todella hankalaa. Verrattain surkea dokumentaatio hankaloitti asioiden oppimista. Parhaat opit Angulariin löytyvätkin erityisesti Youtubesta tai opetuspalveluista kuten Udemy tai PluralSight. Samoihin ongelmiin en Reactin kohdalla törmännyt. Uskon Angularin haastavuuden vaikuttavan voimakkaasti sen suosioon.

Vaikka Reactin oppiminen ei suoraan heijasta Angularin oppimiseen tai päinvastoin, on mielestäni tärkeä nähdä erilaisia tapoja sovellusten rakentamiseen. Ohjelmistoalalla työskennellessä voi olla varma, että tulee törmäämään moniin eri sovelluskehyksiin ja työkaluihin. Angular ja React ovat vain yksi pieni osa verkkosovelluskehitystä, mutta on vaikea olla huomaamatta, miksi React on niin suosittu kuin se on.

5 YHTEENVETO JA POHDINTA

Työ oli mielenkiintoinen ja opettavainen. Tavoitteena oli tutustua Reactiin sekä Angulariin, sekä toteuttaa pienet sovellukset niitä käyttäen. Pääsin tutustumaan työssäni erilaisiin verkkokehityksen osa-alueisiin. Sekä Reactilla että Angularilla on jotain, mitä ne tuovat moderniin verkkokehitykseen. Työmarkkinatilanne on suoraan verrannollinen näiden kahden sovelluskehyksen ja –kirjaston suosioon. React on selkeästi suositumpi, joten sillä on selkeästi enemmän kysyntää. Uskon Reactin suosion ja kysynnän myös lisääntyvän tulevaisuudessa.

Suosin itse Reactia sen nopean kehittämisen sekä nopean oppimisen takia. Reactin oma dokumentaatio on laaja sekä se on laadukasta. En itse välittänyt Angularista juuri ollenkaan. Sen monimutkainen syntaksi ja toiminnallisuudet yhdistettynä surkeaan dokumentointiin teki sillä sovelluksen kehittämisen todella vaikeaksi.

Jatkokehittämällä sovelluksia pääsisi tutustumaan lisää näiden kahden eri työkalun toiminnollisuuksiin. Esimerkiksi Reactin yksi tärkeistä palasista, useContext jäi kokonaan läpikäymättä tässä työssä. Angularin osalta työn laajentamisessa pääsisi tutustumaan lisää eri direktiiveihin, putkiin ja sen sisältämiin valmiisiin toiminallisuuksiin, kuten reititykseen.

Työpaikkojen etsinnässä suosittelen LinkedIniä. Se vaatii hiukan enemmän valmistelua, kuten profiilin tekemistä sekä osaamisen ja työkokemuksen lisäämistä. Näen kuitenkin, että LinkedIn on paras tapa etsiä töitä. Työpaikan löytämisen mahdollisuutta lisää kuitenkin kaikkien aiemmin mainittujen palveluiden yhteinen käyttö.

Mielestäni työni tavoitteet täyttyivät ja se motivoi minua oppimaan lisää molemmista aiheista.

LÄHTEET

1. StackOverflow Developer Survey. 2021. Verkkosivu. Luettu 1.9.2022. https://insights.stackoverflow.com/survey/2021

2. TypeScript.org. N.d. Verkkosivu. Luettu 1.9.2022 https://www.typescriptlang.org/.

3. Agingcoder.com. 27.11.2014. Verkkosivu. Luettu 2.9.2022 https://agingcoder.com/programming/2014/11/27/the-angular-20-hullaba-loo/

4. Nodejs.dev. N.d. Verkkosivu. Luettu 2.9.2022 https://nodejs.dev/en/learn/introduction-to-nodejs.

5. Scott, A, JavaScript Everywhere. 2020. Kirja. Viitattu 3.9.2022

6. GeeksForGeeks.org. 2021. Verkkosivu. Viitattu 14.9.2022. https://www.geeksforgeeks.org/why-node-js-is-a-single-threaded-language/

7. Tyson, M. What is JSON? 26.8.2022. Verkkoartikkeli. Luettu 3.9.2022 https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html

8. LinkedIn.com. Työpaikkahaku avainsanalla React. Verkkosivu. Luettu 6.9.2022. https://www.linkedin.com/jobs/search?keywords=React&location=Finland&geoId=100456013&trk=public_jobs_jobs-search-bar_search-submit&position=1&pageNum=0

9. Duunitori.fi. Työpaikkahaku avainsanalla React. Verkkosivu. Luettu 6.9.2022. https://duunitori.fi/tyopaikat?haku=React

LIITTEET

Liite 1. Lähdekoodi React-sovellukseen
https://github.com/HeneDev/BookApp-React

Liite 2. Lähdekoodi Angular-sovellukseen
https://github.com/HeneDev/BookApp-Angular

Liite 3. Lähdekoodi Node.js-sovellukseen
https://github.com/HeneDev/BookApp-Server