OULUN AMMATTIKORKEAKOULU

Ronny Friman

OPINTOMATERIAALIN LUOMINEN FULL STACK WEB-SOVELLUSKEHITYKSEEN MERN-TEKNOLOGIAPINOLLA

OPINTOMATERIAALIN LUOMINEN FULL STACK WEB-SOVELLUSKEHITYKSEEN MERN-TEKNOLOGIAPINOLLA

Ronny Friman
Opinnäytetyö
Kevät 2021
Tradenomi, tietojenkäsittely
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tradenomi (AMK), Tietojenkäsittely

Tekijä: Ronny Friman
Opinnäytetyön nimi: Opintomateriaalin luominen full stack web-sovelluskehitykseen MERN-teknologiapinolla
Työn ohjaaja: Jouni Juntunen
Työn valmistumislukukausi ja -vuosi: Kevät 2021 Sivumäärä: 32

Työn tavoitteena oli toteuttaa sen toimeksiantajalle Oulun ammattikorkeakoululle opintomateriaali moderniin full stack-web-kehitykseen käyttäen MERN-teknologiapinoa sekä koostaa teknologioiden käyttöön liittyviä lähteitä oppimisen tueksi ja resursseiksi osaksi jatkuvaa opiskelua. Käytetyt lähteet koostuivat pääsääntöisesti käytettyjen teknologioiden virallisista dokumentaatioita. Aiheisiin liittyvien teknologioiden opetus ammattikorkeakouluympäristössä koettiin työn toimeksiantajan taholta tärkeäksi ja sen tavoitteena oli lisätä erityisesti NoSQL-tietokantojen sekä Node-ympäristön opetusta. Työn kirjoitushetkellä teknologioita ei sellaisenaan tarjottu opiskeltavaksi toimeksiantajan toimesta.

MERN-teknologiat koostuvat MongoDB-tietokannasta, Express.js-palvelinsovelluskehyksestä, selaimen ohjelmakoodista vastaavasta React-kirjastosta sekä Node.js-palvelinympäristöstä. Yhdessä nämä teknologiat kattavat kaikki kehitettävän sovelluksen osa-alueet, jolloin kehityksen lopputuloksesta voidaan käyttää termiä full stack-sovellus.

Opintomateriaali koostettiin 11-osaisesta tehtäväsarjasta ja niissä kehitetystä mallisovelluksesta. Tehtäväsarjan tehtävät jakaantuivat useampiin pienempiin tehtäviin, joissa toteutettiin mallisovelluksen yksittäisiä toimintoja tai sen tietty osa-alue. Lisäksi tehtäväsarjan viimeisessä tehtävässä opiskelija implementoi vapaavalintaisen full stack-sovelluksen perustuen mallisovelluksen kehityksestä opittuihin tekniikoihin ja tapoihin.

Opintomateriaali on lähtökohtaisesti suunnattu ammattikorkeakouluopiskelijoille. Materiaalia voidaan tarjota opiskelijoille omatoimiseen itseopiskeluun tai käyttää ohjaavan opettajan toimesta lähiopetuksen tukena. Materiaalin käyttökelpoisuudesta päättää työn toimeksiantaja.

Opinnäytetyö kuvaa materiaalissa käytettyjä teknologioita, niiden käyttöä ja käyttöperusteita, opintomateriaalin luontiprosessia, siihen luotuja tehtäviä sekä perusteita mallisovelluksen ja sen toimintojen valintaan. Lisäksi se selvittää full stack-kehityksen opetuksessa ja oppimisessa havaittuja haasteita ja esittää kehitykseen liittyviä huomioita ja kehitysehdotuksia ammattikorkeakouluympäristössä.

Keywords: full-stack, web-sovelluskehitys, MERN, MongoDB, Express, React, Node

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Systems

Author: Ronny Friman
Title of thesis: Implementing course material for full stack web-development with MERN stack
Supervisor: Jouni Juntunen
Term and year when the thesis was submitted: Spring 2021
Number of pages: 32

The objective of the thesis was to implement study material for modern full stack web development with MERN technology stack and to compile sources related to the concerned technologies to support learning and to provide resources for additional learning in the future. The commissioner of the work was Oulu University of Applied Sciences. The teaching of related technologies was considered important by the commissioner and its objectives was to increase the teaching of NoSQL databases and the Node environment.

MERN is an acronym of the technologies included in the stack. The stack consists of technologies of MongoDB, Express.js, React, and Node.js which, when used together, form a full stack application.

The material includes a 11-part series of tasks and a model application developed in them. In the last task of the series, the student implements a complete full stack application with an optional subject based on the material provided. The study material is primarily allocated for students at University of Applied sciences. It can be offered to the students for self-study, or it can be used by the by teacher to support contact teaching. The usability of the material is decided by the commissioner.

The thesis describes the technologies used in the material and their use in practice. In addition, it explains the process of creating the study material and justifies the reasoning of the designated tasks and model application of the material. Furthermore, the thesis explores the challenges of learning and teaching of full stack development and presents observations and development suggestions for it.

Keywords: full stack, web deveploment, MERN, MongoDB, Express, React, Node

SISÄLLYS

1 JOHDANTO

Sovelluskehittäjien ja ohjelmoinnin opiskelijoiden on tärkeä tuntea web-kehityksen ytimessä olevat perusteknologiat, mutta myös eri tarkoituksiin käytettäviä sovelluskehyksiä ja -kirjastoja, joihin moderni web-kehitys tänä päivänä pitkälti perustuu. Verkkopalvelut ja -sovellukset koostuvat useista eri osa-alueista ja niiden kehitykseen käytettävistä teknologioista, jotka yhdessä muodostavat teknologiapinoja. Usein sovelluskehykset mukailevat toinen toisiaan, joten yhden teknologian tai teknologiapinon opiskelusta on apua myös seuraavien opiskeluun.

1.1 Opinnäytetyön tavoitteet

Opinnäytetyön toimeksiantaja on Oulun ammattikorkeakoulu. Opinnäytetyön tavoitteena on koostaa toimeksiantajalle opintomateriaali modernin web-kehityksen ja MERN-teknologioiden ympärille. Kyseiset teknologiat kokonaisena teknologiapinona eivät kuulu opinnäytetyön tekohetkellä toimeksiantajan opetusvalikoimaan. Toimeksiantaja kokee modernin full stack-ympäristön opetuksen tärkeänä ja sen tavoitteena on lisätä tässä työssä kuvattujen NoSQL-tietokantojen ja Node-ympäristön opetusta tulevaisuudessa. Opinnäytetyön laatijan tavoitteina on syventää osaamista materiaalissa käytettyjen teknologioiden ja tekniikoiden osalta sekä kehittää kykyä toimia ohjelmoinnin kouluttajana, koostaa laadukas, ajankohtainen, johdonmukainen ja innostava opintomateriaali.

Opinnäytetyössä kuvataan opetusmateriaalissa käytettäviä teknologioita, eri sovelluskehysten ja -kirjastojen käytön hyötyjä ja mahdollisuuksia sekä opintomateriaalin luomisen prosessia. Laadittava materiaali voidaan opiskella omatoimisesti tai sitä voidaan käyttää lähiopetuksen tukena. Opinnäytetyössä käytettyjä lähteitä voidaan käyttää oppimisen apuvälineenä ja laajemmin opiskeltuna syventämään osaamista.

Opetusmateriaali koostuu mallisovelluksesta sekä tehtäväsarjasta. Sarjan tehtävät sisältävät mallisovelluksen koodiesimerkkejä sekä vaadittavan teorian tehtävien toteutukseen. Materiaalin tehtäväsarja sisältää eri tasoisia tehtäviä, joista osa on ratkaistavissa seuraamalla materiaalin mallisovellusta, osa vaatii opiskelijan omaa ratkaisukykyä. Kaikki ratkaisut perustuvat kuitenkin materiaaliin tai materiaalissa tarjottuihin lähteisiin. Tehtäväsarjan tehtävät rakentuvat edellisten tehtävien

päälle. Tehtäväsarjan opiskeltua, opiskelija tuntee modernin web-kehityksen perusteet, osaa rakentaa web-käyttöliittymän ja sen vaatimaa käyttölogiikkaa, luoda HTTP-palvelimen sekä suunnitella ja toteuttaa NoSQL -tietokannan. Opintomateriaalin lopputuloksena on internettiin julkaistu full stack-sovellus.

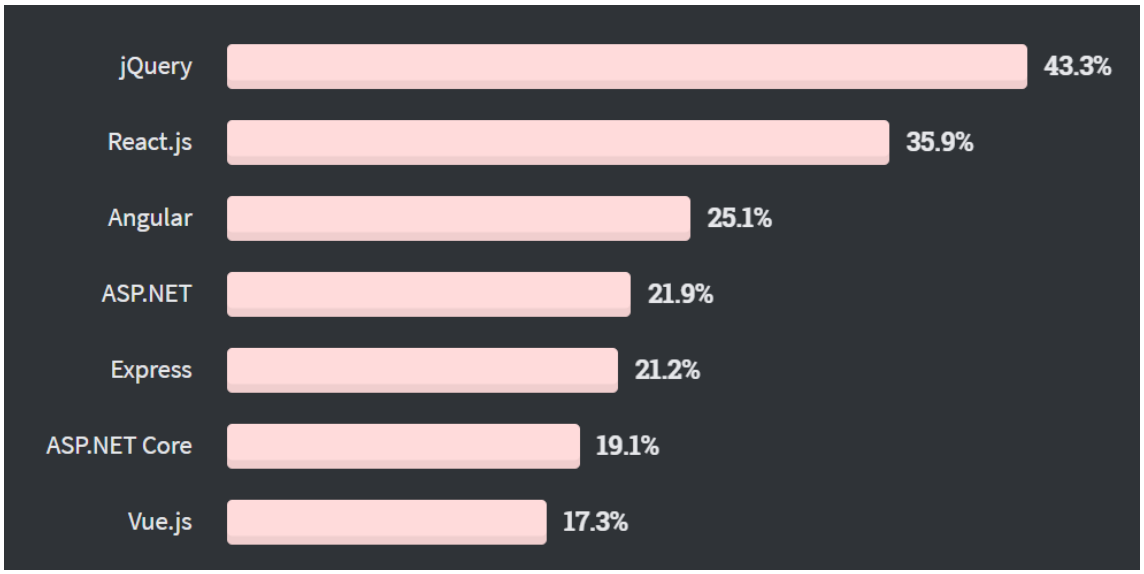
1.2 MERN

MERN tai MERN-teknologiapino (MERN-stack) on joukko teknologioita, joita voidaan käyttää moderniin full stack web-kehitykseen. MERN on akronyymi sen sisältämistä teknologioista, joita ovat MongoDB, Express JS, React JS ja Node.js. Yhdessä nämä teknologiat muodostavat kehitettävälle sovellukselle sen selaimessa näkyvän osuuden (front end), taustalla ajettavan palvelinkoodin (back end) sekä mahdollisuuden datan tallentamiselle tietokantaan. (MongoDB 2021a.)

MERN-pinossa React vastaa kehitettävän sovelluksen front endistä, Node.js ja Express muodostavat sovelluksen back endin ja MongoDB mahdollistaa datan tallentamisen tietokantaan. MongoDB-tietokanta tarjoaa tuen kaikille käytetyimmille ohjelmointikielille, mutta sopii hyvin käytettäväksi juuri Node.js- ja Express -back endin kanssa (MongoDB 2021a). Kun sovelluskehitys kattaa sekä front- että back-endin, sen lopputuloksesta voidaan käyttää termiä full stack-sovellus. MERN-teknologioita käyttäen, sovellus voidaan kirjoittaa kokonaisuudessaan vain JavaScript-ohjelmointikielellä, jonka ansiosta opiskelijan kynnys siirtyä front end-kehityksestä back end-kehitykseen tai toisinpäin, voi madaltua.

JavaScriptiin perustuvista teknologiapinoista on olemassa useita variantteja. MERN-pinon lisäksi web-kehityksessä voidaan käyttää esimerkiksi MEAN- tai MEVN-pinoja. Nämä kolme eri pinoa eroavat toisistaan sovelluksen front endissä käytettävän JavaScript-kirjaston tai -kehyksen osalta. MEAN-pinossa Reactin sijaan käytetään Angularia, MEVN-pinossa käytössä on Vue. (MongoDB 2021a.) Yhteisesti näitä ja muita JavaScriptiin perustuvia pinoja voidaan nimittää myös JavaScript-pinoiksi (JavaScript-stack).

Suositun Stack overflow -kehittäjäyhteisön kehittäjille osoittamassa web-sovelluskirjastoihin ja -kehyksiin liittyvässä kyselyssä selviää, että React on edellä mainituista JavaScriptin front end kirjastoista ja -sovelluskehyksistä suosituin. Yli 42 000 vastaajasta noin 36 % kertoi käyttävänsä Reactia ja hieman yli 25 % Angularia. Vue.js on listalla sijalla seitsemän, jota noin 17 % käyttäjistä kertoi käyttävänsä. Myös materiaalissa käytettävä Express on listalla viiden suosituimman teknologian joukossa. (Stack overflow 2020.) Alla olevassa kuviossa ote edellä mainitun kyselyn tuloksista (kuvio 1).



KUVIO 1. Suosituimmat sovelluskehykset ja -kirjastot

1.3 Kehitystehtävät

Opinnäytetyön opintomateriaalissa implementoitava mallisovellus on eri näkymiä tai ”sivuja” sisältävä elokuvalistasovellus, jonka kehitys kattaa sovelluksen selaimessa näkyvän osan, palvelinpuolen sekä tietokantatoiminnot. Sovellukseen toteutetaan kaikki CRUD (create, read, update, delete) -toiminnot, joten loppukäyttäjän on mahdollista luoda, lukea, muokata ja poistaa elokuvia. Sovellus tallentaa rekisteröityneiltä käyttäjiltä relatiivista dataa, jossa käyttäjä- ja elokuvatietoja yhdistetään toisiinsa. Valmis sovellus julkaistaan internetissä, joten käyttäjillä on pääsy henkilökohtaiseen elokuvalistaan verkkoyhteyden yli mistäpäin tahansa.

Mallisovellus kehitetään tehtäväsarjan kymmenessä ensimmäisessä tehtävässä. Tehtäväsarjan tehtävät sisältävät useamman pienemmän tehtävän, joissa opiskelija toteuttaa tietyn tehtäväkoko-

naisuuteen liittyvän toiminteen tai osan siitä. Tehtäväkokonaisuudet sekä niiden osat ovat tarkoitettu tehtäväksi kronologisessa järjestyksessä. Mallisovellus ja sen tehtävät käsittelevät web-kehityksessä materiaalin laatijan kokemuksen mukaan yleisimpiä toiminto ja toteutustapoja, joiden perusteella ne valikoitiin materiaaliin mukaan. Tehtäväsarjan viimeisessä osassa opiskelija implementoi vapaavalintaisen full stack-sovelluksen perustuen mallisovelluksen esimerkkeihin ja ohjeisiin.

Materiaali koostaa opiskelijalle myös olennaisia lähteitä sovelluksen kehityksessä käytettyihin teknologioihin, joita voidaan käyttää esimerkiksi omatoimiseen jatko-opiskeluun ja resursseina mahdollisten tulevien projektien kehityksessä. Lähteet koostuvat pääsääntöisesti mallisovelluksen kehitykseen käytettyjen teknologioiden ja sovelluskirjastojen virallisista dokumentaatioista.

1.4 Rajaukset

Opinnäytetyö ei sisällä luentomateriaalia tai -kalvoja aiheeseen. Toimeksiantajan kokemuksen mukaan luennointiin käytettävä materiaali halutaan usein koostaa luennoivan opettajan toimesta.

Opintomateriaali on lähtökohtaisesti suunnattu ammattikorkeakouluopiskelijoille. Sen esitietovaatimuksina ovat aikaisempi osaaminen HTML, CSS ja JavaScript-ohjelmointikielistä sekä perustason tuntemus verkkosivujen ja -sovellusten toiminnasta. Opiskelijan oletetaan tuntevan myös versiohallinnan perusteet.

React-ohjelmointia voidaan toteuttaa kahdella eri syntaksilla joko luokka- tai funktiokomponentteja käyttäen. Tässä materiaalissa käytetään ainoastaan funktiokomponentteja, jotka ovat Reactin versiojulkaisun 16.8 jälkeen myös kirjaston kehittäjien suosittelema käytäntö (Reactjs 2021a). Reactin opiskelijoiden on kuitenkin hyvä tuntea myös luokkakomponentit. Internetistä löytyvissä esimerkeissä tai dokumentaatioissa voi olla käytetty luokkakomponentteja. Näissä tapauksissa helpottaa, kun opiskelija osaa vähintään konvertoida luokkakomponentit omiin projekteihin sopiviksi funktiokomponenteiksi. Lisäksi työelämässä ohjelmistokehittäjänä on mahdollista, että työstettävää projektia toteutetaan luokkakomponenteilla.

Materiaalissa kuvataan toteutuksessa käytettyjä työkaluja käsittelemättä niitä kuitenkaan syvällisemmin. Materiaalin mallisovelluksen kehityksessä käytettävät sovellukset ovat Visual Studio

Code, Postman, Node.js, Git ja Heroku CLI. Materiaali on mahdollista suorittaa myös vaihtoehtoisia työkaluja käyttäen. Materiaalin ohjeiden mukainen kehittäminen edellyttää opiskelijalta rekisteröitymisen Github, MongoDB Atlas ja Heroku -palveluihin. Opintomateriaalin tekohetkellä kaikkien materiaalissa käytettyjen teknologioiden, sovellusten, lisäosien ja palveluiden käyttö on maksutonta, eivätkä ne vaadi käyttäjiltä esimerkiksi luottokorttitietojen syöttämistä.

2 MONGODB

MongoDB on avoimeen lähdekoodiin perustuva dokumenttipohjainen, ei-relatiivinen NoSQL-tietokanta ja tietokantajärjestelmä. MongoDB:ssä data tallennetaan olion kaltaisiin dokumentteihin ja niiden muodostamiin kokoelmiin. Dokumentti koostuu nimi-arvo-pareista, joiden arvo -kenttä voi sisältää esimerkiksi merkkijonon, numeron, taulukon (array) tai toisen dokumentin. Tarkemmin kuvattuna dokumentti on BSON-muotoinen (Binary JSON), joka vastaa binäärimuotoista JSON-oliota (JavaScript Object Notation). Useista tallennetuista dokumenteista muodostuu yksi tai useampi kokoelma ja kokoelmista kokonainen tietokanta. Dokumenttitietokannoissa kokoelma vastaa SQL-tietokantojen taulua. (MongoDB 2021b.)

2.1 NoSQL

MongoDB on yksi useista tietokannoista, jotka poikkeavat perinteisimmistä SQL-tietokannoista. Yhteisesti ei-relatiivisia tietokantoja voidaan kutsua ”NoSQL” (”Not only SQL”) -tietokannoiksi. Vaikka relatiivinen MySQL on edelleen suosituin tietokantapalvelu, NoSQL -tietokantojen käyttö on kasvanut 2000-luvusta lähtien datan tallennuksen kustannusten laskujen myötä. (MongoDB 2021c.)

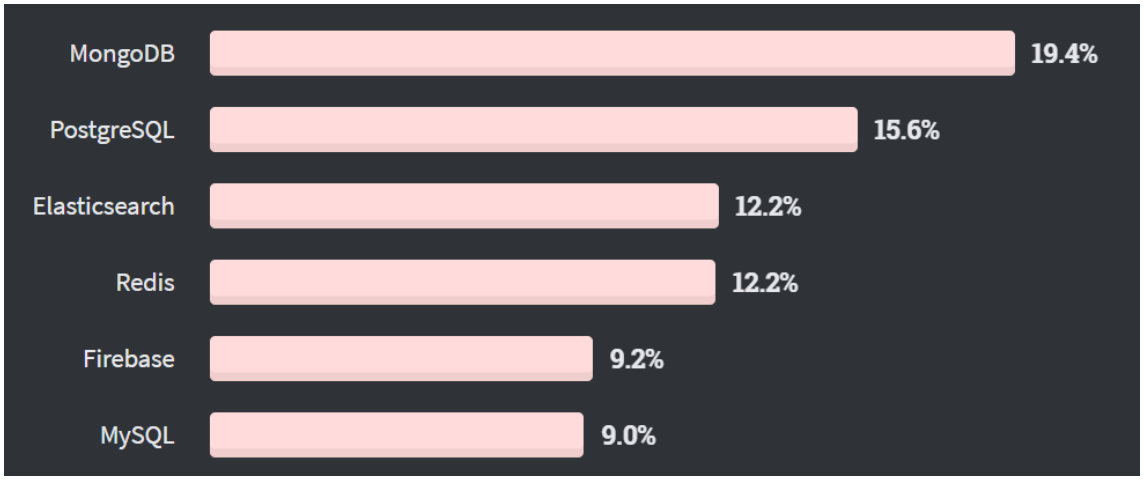
SQL ja NoSQL -tietokantatekniikoiden yksi keskeisimmistä eroista tallennusmuodon ohella on datan mallinnus (data modeling) ja mallien mahdollinen muodon muuttaminen kehityksen aikana. NoSQL -kannoissa datamallit ovat tyypillisesti joustavia. Niiden ansiosta kehittäjien on mahdollista muuttaa tallennettavan datan, tässä tapauksessa dokumentin, muotoa, mikäli se sovelluksen kehittämisen aikana on tarpeen. Vaikka datamallien suunnittelu onkin tärkeä työvaihe myös NoSQL-kantojen kanssa työskennellessä, sen voidaan kokea olevan helpompaa juuri joustavien mallien ansiosta. Mikäli datamallia muutetaan sovelluksen elinkaaren aikana, datan tallentamista voidaan siitä huolimatta jatkaa samaan kokoelmaan, jolloin kokoelma sisältää eri muotoista dataa. (MongoDB 2021c.)

Muita NoSQL:n etuja ovat nopeat tietokantakyselyt eli tehokkuus ja skaalautuvuus. Verratessa esimerkiksi MySQL:n ja MongoDB:n tehokkuuksia, voidaan todeta, että MySQL suorittaa nopeammin

suuren määrän datan valintaa, kun taas MongoDB on huomattavasti nopeampi lisäämään ja päivittämään suuren määrän tietueita. NoSQL-tietokantoja voidaan käyttää suurienkin datamäärien tallennukseen, jonka ansiosta se skaalautuu hyvin myös suurempien sovellusten käyttöön. (MongoDB 2021d.)

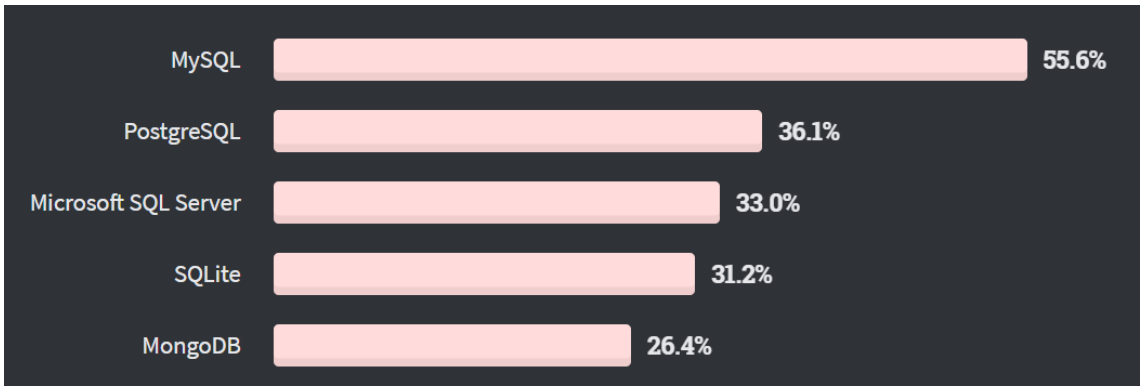
Vaikka MongoDB:n yhteydessä puhutaan ei-relatiivisesta tietokannasta, voidaan silläkin tallentaa relatiivista dataa. MongoDB:ssä relatiivisuus voidaan toteuttaa määrittelemällä datamalliin kokoelmien ja dokumenttien välisiä ristikkäisiä id-tietoja ja viitteitä, jotka myöhemmin korvataan niihin liittyvillä kokonaisilla dokumenteilla tai osilla niistä. Toimintoa kutsutaan datan populoinniksi. (Mongoose 2021.)

NoSQL-kantojen käyttö voidaan kokea kehittäjiltä kerätyn tiedon mukaan helpommaksi ja kiinnostavammaksi SQL-kantojen käyttöön verrattuna. Stack Overflown kyselyssä viisi kehittäjien mielestä kiinnostavinta tietokantateknologiaa olivat NoSQL-teknologialla toteutettuja. (Stack Overflow 2020.) MongoDB oli kyselyn tuloksissa kiinnostavin tietokantateknologia (kuvio 2).



KUVIO 2. Stack Overflow: kehittäjät, jotka eivät vielä käytä tietokantateknologiaa, mutta ovat kiinnostuneita siitä

Kun kehittäjiltä kysyttiin, mitä tietokantaa he käyttävät, MySQL osoittautui edelleen suosituimmaksi (Stack Overflow 2020). MongoDB on suosituimpien teknologioiden listalla viides. (Kuvio 3.)



KUVIO 3. Stack Overflow: suosituimmat tietokantapalvelut, vastauksia 49 537

2.2 MongoDB tietokantakyselyt ja Mongoose

Esimerkiksi MySQL, kuten monet muut relatiiviset tietokannat, käyttävät kyselykielenään Structured Query Language (SQL) -kyselykieltä. MongoDB:n kyselykieli on MongoDB Query Language (MQL), joka on suunniteltu kehittäjien käyttöön mahdollisimman helpoksi. (MongoDB 2021e.) Seuraavassa esimerkki edellä mainittujen kielien erosta kyselyssä, jossa MQL:llä ja SQL:llä etsitään tietokannan tietueet, joissa tila (status) on ”A” ja lukumäärä (qty) on pienempi kuin 30 (kuvio 4).

```
const cursor = db.collection('inventory').find({
  status: 'A',
  qty: { $lt: 30 }
});
```

```
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

KUVIO 4. Esimerkki MQL ja SQL -kyselyistä (ylempi MongoDB:n kysely, alempi SQL)

Vaikka MongoDB:n tietokantakyselyt, datan manipulointi ja datan mallinnus on kokonaan MQL-kyselykielellä toteutettavissa, sen päällä on mahdollista käyttää Mongoose-kirjastoa, jonka tarkoitus on edelleen helpottaa MongoDB:n käyttöä. Mongoose on Node.js:ään perustuva olio-datamallinnus (Object Data Modeling, ODM) -kirjasto. Mongoosen keskiössä on sillä määriteltävä ”Schema”, joka toimii pohjapiirustuksena dokumenttien luontiin. Schema kertoo tietokannalle, missä muodossa dokumentit tallennetaan eli määrittää dokumenttien kentät ja niiden tietotyypit. Kun Schema on määritetty, siitä luodaan Schemaan perustuva Mongoosen datamalli. (MongoDB 2021e.) Alla olevassa esimerkissä viittaus Mongoosen Schemaan ja datamallin luontiin (kuvio 5).

```
copy code

var blog = new Schema({
    title: String,
    slug: String,
    published: Boolean,
    content: String,
    tags: [String],
    comments: [{
        user: String,
        content: String,
        votes: Number
    }]
})

var Blog = mongoose.model('Blog', blog)
```

KUVIO 5. Mongoose Schema ja datamallin luonti

Kun datamalli on luotu, kaikki tietokantakyselyt ja -muutokset voidaan tehdä mallia vasten. Kyselyt tehdään Mongoosen malliolion eri metodeja käyttäen. Esimerkissä määritellään tallennettavan dokumentin kenttien sisältö, dokumentti tallennetaan ja etsitään tietokannasta (kuvio 6).

```
copy code

// Create a new blog post
var article = new Blog({
    title:'Awesome Post!',
    slug:'awesome-post',
    published: true,
    content: 'This is the best post ever',
    tags: ['featured', 'announcement'],
})

// Insert the article in our MongoDB database
article.save();

// Find a single blog post
Blog.findOne({}, (err,post)=>{
    console.log(post);
})
```

KUVIO 6. Tietokantaoperaatioita käyttäen Mongoosen datamallia

3 EXPRESS JS

Express JS on suosittu Noden päälle rakentuva sovelluskehys. Sen tarkoitus on abstraktoida palvelinkoodia, jonka ansiosta kehittäjän työtä voidaan helpottaa. Express tarjoaa suuren määrän erilaisia ominaisuuksia Nodessa jo olemassa olevien ominaisuuksien lisäksi. (Express 2021a.)

Express ei itse sovelluskehyksenä ota kantaa siihen, kuinka sillä toteutettuja sovelluksia pitäisi tehdä. Expressissä voidaan käyttää mitä tahansa siihen sisäänrakennettuja tai ulkoisia lisäosia vastaamaan tietystä sovelluksen osa-alueesta tai toiminnosta. Expressillä toteutettu sovellus voidaan toteuttaa yhteen tai useampaan tiedostoon eikä esimerkiksi projektien kansiorakenteeseen oteta sovelluskehyksen puolesta kantaa. (MDN Web Docs 2021a.)

3.1 Node.JS

Node.js on avoimen lähdekoodin ekosysteemi ja alustariippumaton ajoympäristö (runtime environment) JavaScript -koodin suorittamiseen selaimen ulkopuolella, esimerkiksi palvelimella tai paikallisella työasemalla (Nodejs 2021a). Node.js perustuu Googlen V8 JavaScript-moottoriin, samaan moottoriin, joka pyörii myös esimerkiksi Google Chromen taustalla. V8-moottorin ansiosta Node-sovellukset voivat olla hyvin suorituskykyisiä ja alustariippumattomia – se mahdollistaa myös esimerkiksi työpöytäsovellusten kehityksen Nodea käyttäen. (Nodejs 2021b.)

Node-sovellusten etuna JavaScriptiä jo tunteville ohjelmoijille tai ohjelmoinnin opiskelijoille on, että samaa ohjelmointikieltä voidaan käyttää sekä sovelluksen front endissä että back endissä. Kuitenkin, vaikka ohjelmointikieli ja koodin syntaksi ovat samat (Nodejs 2021a), Nodella ja selaimessa toimivan sovelluksen toiminta poikkeaa toisistaan täysin. (Nodejs 2021c.)

Node.js sisältää useita sisäänrakennettuja moduuleja, API-rajapintoja, jotka ovat kehittäjien käytettävissä ilman erillisiä asennuksia. Moduulit tarjoavat kehitettävien toimintojen taustalle valmista ohjelmakoodia, joiden tarkoitus on helpottaa ja nopeuttaa kehittäjien työtä. (Nodejs 2021a.) Näitä moduuleja ovat esimerkiksi fs (filesystem) -moduuli, joka tarjoaa Node-sovellukselle pääsyn paikalliseen tiedostojärjestelmään, jossa tiedostoja voidaan lukea, kirjoittaa ja poistaa (Nodejs 2021d.) sekä HTTP-moduuli, jolla voidaan luoda HTTP-palvelin. (Nodejs 2021e.)

Sisäänrakennettujen moduulien tai pakettien lisäksi Node.js tarjoaa Node Package Manager (”npm”) -paketinhallintatyökalun, jonka avulla valmiita koodipaketteja on ladattavissa yli miljoonan julkaistun paketin rekisteristä (Snyk 2019). NPM on alun perin Node-ekosysteemiin kehitetty työkalu, mutta sittemmin laajentunut vallitsevaksi tavaksi JavaScript-projektien paketinhallintaan. Paketinhallintatyökalu koostuu verkkopalvelusta, josta paketteja voidaan etsiä, komentorivityökalusta, jonka kautta käyttäjät suorittavat pakettien lataukset ja muun hallinnan sekä pakettirekisteristä. (Npm Docs 2021.)

3.2 Reititys ja middlewaret

Express-sovelluksessa HTTP-pyyntöjen reititystä voidaan hallita lukemalla URL-polkuja (route), joihin pyynnöt tulevat (Express 2021b). Polku määritetään osoitteen verkkopäätteen jälkeisessä osuudessa kauttaviivalla erotettuna, kuten seuraavasta esimerkistä on huomattavissa (kuvio 7).

```
const myURL = new URL('https://example.org/abc/xyz?123');
console.log(myURL.pathname);
// Prints /abc/xyz
```

KUVIO 7. Osoitteen polun osoittaminen ja tulostaminen

Sovelluskehys tarjoaa mahdollisuuden luoda käsittelijöitä palvelimen vastaanottamien HTTP-pyyntöjen käsittelemiseksi eri routeissa. Lisäksi käsittelijöissä määritellään, mitä HTTP-metodia ne kuuntelevat. Kun selaimesta lähetetty HTTP-pyynnön route täsmää Expressissä määritettyyn routeen, sen käsittelijät suoritetaan. Käsittelijät vastaavat siitä, kuinka selaimen pyyntöön vastataan. (MDN Web Docs 2021a.)

Express-sovellusten yksi oleellinen osa on middlewaret. Middleware on funktio, jolla on pääsy sovellukseen sisään tuleviin HTTP-pyyntöihin ja sen mukana tuleviin olioihin. Ne voivat myös suorittaa mitä tahansa JavaScript-koodia ja tehdä muutoksia pyyntöihin. Expressistä löytyy useita sisäänrakennettuja Middlewareja ja niitä voidaan asentaa myös Expressin ulkopuolelta. (Express 2021c.)

4 REACT JS

React JS on Facebookin perustama ja ylläpitämä avoimen lähdekoodin komponenttipohjainen JavaScript -kirjasto sovelluksen selaimessa näkyvään osuuteen eli front endiin. React-ohjelmakoodi perustuu JavaScript funktioihin, jotka palauttavat paloja JSX-koodia, jotka lopulta kääntyvät selaimessa näytettäväksi HTML-koodiksi. Reactissa näitä funktioita kutsutaan komponenteiksi (Reactjs 2021b).

4.1 Single page-sovellukset (SPA)

Perinteisessä web-kehityksessä verkkosivut koostuvat useista HTML-tiedostoista, joita selain lataa yksitellen, kun käyttäjä on vuorovaikutuksessa sovelluksen kanssa. Käyttäjän kirjoittaessa selaimen osoiteriville osoitteen, selain lähettää HTTP-pyynnön (HTTP-request) palvelimelle, jossa se pyytää palvelinta palauttamaan pyydetyn verkkosivun kopion takaisin pyytäjälle. Mikäli palvelin hyväksyy selaimen pyynnön, se vastaa siihen tilakoodilla ”200 OK” ja aloittaa verkkosivuston tiedostojen lähettämisen selaimelle. Selain aloittaa tiedostojen ja niiden ohjelmakoodin renderöinnin näytölle. Kun käyttäjä navigoi sivuston sisällä esimerkiksi toiseen osioon, selainikkuna päivittyy ja edellä mainittu prosessi suoritetaan uudestaan. (MDN Web Docs 2021b.)

Perinteisen web-kehityksen rinnalla on kuitenkin ollut jo pitkään myös malli, jossa sovellus koostuu vain yhdestä HTML-tiedostosta ja jossa käyttöliittymää päivitetään JavaScriptin toimesta käyttäjän vuorovaikutuksesta vain tietyltä osin, esimerkiksi navigoidessa sovellusta tai hiiren painalluksesta, eikä kokonasta sivua ja selainta ladata uudestaan. Edellä kuvattua mallia ja tekniikkaa käyttäviä verkkosivuja ja -sovelluksia kutsutaan ”single-page-sovelluksiksi”. (Sagar 2020.)

4.2 Komponentit ja JSX

React-sovellukset perustuvat komponentteihin, jotka ovat pieniä paloja eristettyä koodia ja sovelluksen näkymää. Materiaali kuvaa ainoastaan funktiokomponentit, jotka ovat tavallisia JavaScript-funktioita, jotka palauttavat aina palan JSX:ää. Reactia voidaan kirjoittaa myös käyttäen JavaScriptin luokkasyntaksia, jolloin puhutaan luokkakomponenteista (class components). (Reactjs 2021c.) Alla olevassa kuvassa esimerkki Reactin funktiokomponentista (kuvio 8).

```
1  const Hello = (props) => {
2    return <h1>Hello, {props.name}</h1>
3  }
4
5  const element = <Hello name="John"  />
```

KUVIO 8. React-funktiokomponentti

Komponenteille voidaan antaa JSX-attribuutteja HTML-koodista tutulla syntaksilla. Komponentti ottaa vastaan Reactiin sisäänrakennetun olion props, johon on kerätty kaikki komponentille annetut JSX-attribuutit ja niiden arvot. Props-olion kautta komponentti pääsee siis lukemaan sille annettuja attribuutteja eli dataa. (Reactjs 2021c.) Koska komponentit ovat tavallisia funktioita, ne voivat sisältää JSX:n lisäksi muutakin logiikkaa, kuten apufunktioita, HTTP-kutsuja tai muita komponentteja.

JSX vaikuttaa HTML-koodilta, mutta itse asiassa se on jatke JavaScriptin syntaksille. Kehityksen aikana JSX kääntyy ensin tavallisiksi JavaScriptin funktiokutsuiksi ja lopulta selaimessa näkyväksi HTML-koodiksi. React-koodissa JSX siis tuottaa selaimessa näytettävät elementit. (Reactjs 2021b.) Kuvassa esimerkki JSX-elementistä (kuvio 9).

```
1  const element = <h1>Hello World!</h1>
```

KUVIO 9. JSX-esimerkki

JSX-lausekkeiden sisään on mahdollista upottaa JavaScipt-logiikkaa, kuten ehtoja, laskutoimituksia, silmukoita tai funktiokutsuja, joita voidaan käyttää esimerkiksi selaimessa näkyvän sisällön muuttamiseksi käyttäjän toimintojen perusteella. Reactia voidaan kirjoittaa myös ilman JSX:ää, mutta sen käytön voidaan kokea helpottavan käyttöliittymän logiikan luomista. (Reactjs 2021b.)

4.3 Virtual DOM

DOM (Document Object Model) viittaa HTML-puuhun ja sen elementteihin eli käytännössä selaimessa näkyvään käyttöliittymän. Kun DOM-elementeissä tapahtuu muutoksia, muutokset renderöidään selaimeen näkyviin. Muutokset DOM:ssa ovat nopeita, mutta muutoksien renderöinti eli piirtäminen selaimeen näkyviin on raskas operaatio. Lisäksi esimerkiksi kehitystä tehtäessä perinteisemmällä JavaScriptin itsenäisellä versiolla (Vanilla JavaScript) uudelleen renderöintiä tapahtuu tarpeettoman paljon, joka hidastaa käyttöliittymän toimintaa. (Hamedani 2018.)

Reactissa on sisäänrakennettu konsepti Virtual DOM, joka on nimensä mukaisesti virtuaalinen versio tai kopio oikeasta DOM:sta. Virtual DOM on keskeisessä roolissa React-sovellusten tehokkuuden kannalta. Kun React-sovelluksen käyttöliittymässä tapahtuu muutoksia, React luo uuden Virtual DOM:n, johon muutokset ovat päivitetty. Kun uusi Virtual DOM on luotu, React vertaa sen sisältöä oikeaan DOM:iin ja päivittää sitä vain muuttuneiden elementtien osalta laskemallaan mahdollisimman optimaalisella tavalla. Prosessin johdosta renderöinnin suoritus on kevyempää ja käyttöliittymä vastaa muutoksiin nopeammin. (Hamedani 2018.)

4.4 Komponenttien tila ja ”hookit”

Web-kehityksessä sovelluksien tilaa (state) voidaan säädellä yhdellä tai useammalla muuttujan arvolla, yksinkertaisimmillaan boolean-arvolla. Tilojen tehtävä on määrittää sovelluksen toimintaa jollain tapaa, niiden perusteella käyttöliittymässä voidaan suorittaa esimerkiksi jokin tietty toiminne käyttäjän vuorovaikutuksesta. (Quisenberry 2018.)

Reactin luokkakomponentteja käyttäessä tila ja sen hallinta ratkaistaan sisäänrakennetulla state-oliolla. Ennen Reactin versiosta 16.8, funktiokomponentteja ei käytännössä ollut mahdollista käyttää, koska tilan omaavat komponentit oli mahdollista määritellä vain luokkasyntaksia käyttäen. (Reactjs 2021a.)

Versiossa 16.8 Reactiin tuotiin uutena ominaisuutena hookit (hooks), joiden ansiosta luokkakomponenteista oli mahdollista siirtyä funktiokomponentteihin. Hookeja ei ole mahdollista käyttää luokkakomponenteissa ja kirjaston dokumentaatiossa suositellaankin siirtymistä hookien käyttöön.

Hookit ovat Reactin sisäänrakennettuja funktioita, joiden avulla päästään esimerkiksi käsiksi komponenttien tilaan tai niiden eri elinkaaren kohtiin. (Reactjs 2021d.)

Reactissa tärkeimmät hookit ovat useState ja useEffect. Sovelluksen tilaa voidaan kontrolloida useState-hookilla. Erilaisia sivuvaikutuksia komponenttien eri elinkaarien vaiheissa on mahdolista toteuttaa useEffect-hookilla. Hookien käytössä on myös sääntöjä: hookeja voidaan kutsua vain funktiokomponenteissa, komponenttien ylätasolla, eikä esimerkiksi silmukoiden tai ehtolausekkeiden sisällä. Reactissa on yhteensä kymmenen sisäänrakennettua hookia, joiden lisäksi on mahdollista rakentaa myös omia kustomoituja hookeja (custom hooks). (Reactjs 2021d.)

Hookit ovat funktioita ja käyttäessä esimerkiksi useState-hookia, sitä kutsutaan ensin alustavalla arvolla. Alustava arvo voi olla mikä tahansa JavaScriptin primitiivi tai vaikka toinen funktiokutsu. Funktio palauttaa taulukon, joka sisältää kaksi alkiota: tämänhetkisen komponentin tilan arvon sekä funktion, jolla tilaa voidaan muuttaa. Kun komponentti renderöidään ensimmäistä kertaa, sen tila on alustava arvo, jolla useStatea on kutsuttu. Tilan muuttaminen aiheuttaa aina sen komponentin uudelleen renderöinnin, jonka ansiosta päivitetty tila voidaan nähdä selaimessa. (Reactjs 2021e.) Kuvassa useState-esimerkkilauseke (kuvio 10).

```
1  const [count, setCount] = useState(0);
```

KUVIO 10. useState-esimerkki

Useat komponentit voivat käyttää samaa tilaa, jolloin tilan määrittäminen kannattaa nostaa komponenttien vanhemmalle, josta tila voidaan jakaa sitä tarvitseville komponenteille. Tilat, kuten muu data sovelluksen sisällä, voidaan jakaa komponenttien välillä propsien avulla. (Reactjs 2021f.)

4.5 Näkymät

Single-page-sovellukset tarvitsevat jonkun tavan, miten eri näkymien tai ”sivujen” välillä voidaan navigoida lataamatta selainta uudestaan. Reactissa tähän ei ole sisäänrakennettua tapaa, joten sen toteuttamiseksi tarvitaan ulkoista kirjastoa. Vaihtoehtoisia kirjastoja on olemassa, mutta hyvin usein web-sovelluksissa käytetään React-router-kirjaston React-router-dom-pakettia. (Mittal 2020.)

Navigointi näkymien välillä perustuu reitteihin (route), joissa React-komponentit kääritään kirjaston Route-komponentteihin, joille on annettu propseina polut, joissa tietyt komponentit renderöidään. Router on koko ajan tietoinen selaimen tämänhetkisestä polusta ja kun polku täsmää Route-komponenttiin annettuun, määritetyt komponentit renderöidään ja selaimen näkymä muuttuu. Navigointi polkujen tai näkymien välillä tapahtuu kirjaston Link-komponentin avulla, johon on sisäänrakennettu toiminnallisuus, joka estää selaimen uudelleenlatauksen. (Mittal 2020.)

Kirjasto tarjoaa myös hookeja, kuten useHistory ja useParams. Hookeista useHistoryn avulla päästään käsiksi history-olioon, joka sisältää esimerkiksi tiedot edellisistä poluista, tämänhetkisen polun ja funktioita, joiden avulla näiden välillä voidaan liikkua ohjelmallisesti. Toiseksi mainitulla useParams-hookilla voidaan lukea osoiterivillä annettuja parametrejä. (React Training 2021.)

4.6 Tyylit

React-sovellusten tyylittelyyn on useita eri tapoja. Perinteisessä web-kehityksessä usein yksi CSS-tiedosto vastaa koko sovelluksen tyylimäärittelyistä. Tämä tapa on myös Reactissa käytetty tapa. (Saring 2018.)

React perustuu komponenttiajatteluun. Sitä vahvistaakseen, voidaan myös tyylit ottaa komponenttikonseptiin mukaan luomalla jokaiselle komponentille myös oma tyylitiedosto. Tyylitiedoston määrittelyt näkyvät tällöin vain oman komponentin näkyvyysalueella, jolloin elementtien nimeäminen on helpompaa eikä ristiin menevistä nimistä tarvitse huolehtia. (Saring 2018.)

Myös inline-tyylit ovat hyväksytty ja käytetty tapa Reactissa, vaikka tyylittely esimerkiksi CSS-luokkia käyttämällä onkin suorituksen kannalta tehokkaampi tapa (Reactjs 2021g). Inline-tyyleillä tarkoitetaan CSS-määrityksiä React-koodin sisällä. React-komponenteilla on sisäänrakennettu attribuutti style, jonka sisään voidaan tehdä CSS-sääntöjä käyttäen oliosyntaksia. Olio voidaan sijoittaa suoraan attribuutin sisään tai se voi olla komponentin tai elementin ulkopuolella nimetty olio, joka annetaan tyyliattribuutin koodilohkon sisään. Molemmilla tavoilla tyyleihin voidaan vaikuttaa helposti JavaScriptin avulla esimerkiksi ehtolauseita tai erilaisia laskutoimituksia käyttäen. (Saring 2018.)

Reactissa voidaan käyttää perinteisen web-kehityksen tavoin myös ulkoisia tyyli- tai komponenttikirjastoja. Nämä kirjastot yleensä tarjoavat CSS-luokkia, jotka sisältävät valmiiksi määritettyjä tyylisääntöjä tai kokonaisia etukäteen tyyliteltyjä komponentteja. Lisäksi ne voivat sisältää erilaisia työkaluja ja tapoja toteuttaa sovelluksen komponenttien asettelua. (Vetter-Neo 2020.)

Komponenttikirjastot, kuten ulkoiset kirjastot yleensä, voivat nopeuttaa kehitystä. Lisäksi ne saattavat helpottaa tyylien yhteensopivuutta eri tuottajien selainten välillä tai sovelluksen esteettömyyden toteuttamisessa. Näiden kirjastojen käytöllä voi olla kuitenkin myös haittoja sovelluksen kehityksen kannalta. Kirjastojen tarjoamat valmiit tyylit saattavat vahvasti määritellä käyttöliittymän tiettyyn ulkoasuun ja tuntumaan. Tyylejä yleensä on mahdollista kustomoida, mutta valmiiksi tehtyjen CSS-sääntöjen ylikirjoittaminen voi olla työlästä. Vaikka tyylimäärittely olisikin jokseenkin saman tapaista eri kirjastojen välillä, uuden kirjaston käyttöönottoon aiheuttaa aina myös uuden opettelua. Tyylikirjastot voivat olla myös hyvin kevyitä ja tarjoavat vain esimerkiksi tietyn tyylillisen toiminnallisuuden tai antavat vain perusrungon toteutukselle ja mahdollistavat elementtien kustomoinnin helposti. (Vetter-Neo 2020.)

5 OPINTOMATERIAALI

Opinnäytetyön tuloksena on toteutettu mallisovellus ja 11-osainen tehtäväsarja, jonka tehtävät rakentuvat toistensa päälle. Tehtävät tarjoavat opiskelijalle tarvittavan osaamisen ja työkalut tehtäväsarjan viimeiseen tehtävään, jossa opiskelija implementoi vapaavalintaisen materiaaliin perustuvan full stack-sovelluksen.

Verkkopalvelut ja -sovellukset tyypillisesti sisältävät useita erilaisia näkymiä, lomakkeita, renderöitäviä listoja, painikkeita ja niihin kiinnitettyjä dynaamisia toimintoja sekä HTTP-pyyntöjä. Palvelinpuolella keskiössä ovat HTTP-pyyntöjen käsittely ja niihin liittyvät polut, tietokantaoperaatiot ja -mallit sekä virheenkäsittely. Opintomateriaalin mallisovellus ja tehtäväsarja pyrittiin perustamaan näiden toimintojen ja käyttötapausten ympärille. Myös relatiivisen datan tallentaminen NoSQL -kantaa käyttäen haluttiin nostaa yhdeksi osaksi mallisovellusta ja tehtäväsarjaa sovelluksen internettiin julkaisemisen ohella. Käyttöliittymän ulkoasun suunnittelua ja toteutusta materiaali ei kuitenkaan käytännön tasolla käsittele.

Opintomateriaali ei sisällä luentomateriaaleja, joten tehtävien yhteydessä käydään läpi myös tehtäviin toteutettujen toimintojen ja käytettyihin tekniikoihin ja teknologioihin liittyvää teoriaa. Tehtävät sisältävät linkkejä resursseihin, joita opiskelemalla opiskelijan on myös mahdollista syventää osaamistaan aiheisiin liittyen.

5.1 Mallisovellus

Opintomateriaalin tehtäväsarjassa implementoidaan elokuvalistasovellus, joka toimii mallisovelluksena sarjan viimeisessä tehtävässä opiskelijan omatoimisesti ja opitun perusteella toteutettavalle vapaavalintaiselle full stack-sovellukselle. Teknisesti sovellus sisältää muun muassa käyttöliittymän elementtien ja osien näyttämistä ja muokkaamista dynaamisesti, lomakkeiden käsittelyä, datan ja listojen manipulointia, näkymien ja reitityksen hallintaa, HTTP-pyyntöjen lähettämistä ja vastaanottamista ja niistä mahdollisesti syntyvien virhetilanteiden käsittelyä sekä tietokantaoperaatioita ja datan populointia. Valitut toiminnot ja käytänteet tarjoavat opiskelijalle modernin web-kehityksen perusteet ja tarvittavan osaamisen nykyaikaisen full stack-sovelluksen luontiin.

Sovelluksessa käyttäjien on mahdollista listata sovellukseen elokuvia ja muokata niitä merkkaamalla elokuvat joko katsotuksi tai katsomattomiksi. Sovellus toimii siis käyttäjien henkilökohtaisena listauksena elokuvista, joita käyttäjät ovat nähneet ja toisaalta esimerkiksi muistilistana elokuvista, joita käyttäjä haluaa nähdä.

5.2 Opintomateriaalin tehtävät

Tehtäväsarjassa tuotettavan mallisovelluksen implementointi aloitetaan luomalla uusi React-projekti käyttäen kirjastoon sisäänrakennettua create-react-app toimintoa, joka konfiguroi sovelluksen asetuksia, luo sovellusrungon ja tarvittavia tiedostoja eli niin sanotun ”boilerplaten”. Sarjan ensimmäisessä tehtävässä luodaan ensimmäiset komponentit, tutustutaan Reactin props-käsitteeseen ja niiden toimintaan teoriassa sekä käytännössä, toteutetaan ehdollista renderöintiä käyttäjän vuorovaikutuksesta sekä harjoitellaan listojen renderöinnin perusteita.

Tehtävässä kaksi kehityksessä edetään useState-hookin käyttöön ja sovelluksen tilanhallintaan sekä web-kehityksessä hyvin keskeisiin toimintoihin eli taulukoiden ja datan manipulointiin. Nämä toiminteet näyttäytyvät käyttöliittymässä mahdollisuutena uusien elokuvien lisäykseen, muokkaukseen ja poistoon. Lisäksi tehtävässä tutustutaan lomakkeiden käsittelyyn Reactissa. Toiminteet vaativat tapahtumakäsittelijöiden lisäämistä sovellukseen, jotta käyttäjän vuorovaikutusta sovelluksen kanssa voidaan kuunnella.

Kehitettävä sovellus sisältää edellisen tehtävän suoritettua useamman komponentin, tilan, lomakkeen sekä sovelluslogiikkaa ja apufunktioita. Kuitenkin implementoitu koodi elää tässä vaiheessa kehitystä kokonaisuudessaan yhdessä tiedostossa, vaikka React perustuu komponenttiajatteluun, jossa komponentit usein kirjoitetaan myös omiin tiedostoihinsa. Materiaalin seuraavassa tehtävässä tehdään olemassa olevaan koodin uudelleenjärjestelyä, jossa suoritetaan edellä mainittua koodin eristämistä, jonka johdosta myös projektin kansiorakenteeseen tehdään muutoksia. Lisäksi tehtävä käsittelee Reactissa tehtävää tyylimäärittelyä.

Tehtävässä neljä sovelluksen perustoiminnot ovat jo käytettävissä sovelluksen front endistä, joten seuraavaksi materiaalissa siirrytään palvelinpuolen kehitykseen. Back endin kehitys aloitetaan luomalla sille uusi projekti ja konfiguroimalla Node-ympäristö. Ympäristöön asennetaan projektin ensimmäinen riippuvuus, Express, jolla toteutetaan yksinkertainen HTTP-palvelin sekä ensimmäiset

polut ja niiden käsittelijät, joiden toimivuutta testataan esimerkeissä Postman-sovelluksella. Lisäksi tehtävässä otetaan käyttöön sisäänrakennettuja ja ulkoisesti asennettavia middlewareja sekä toteutetaan myös oma logger-middleware.

Tehtävässä viisi sovelluksen front- ja back end yhdistetään ja kehityksessä siirrytään takaisin React-sovellukseen, jossa toteutetaan palvelimelle tehtyä ohjelmakoodia vastaavat lisäykset myös sovelluksen front endiin. Tehtävässä tutustutaan uuteen Reactin hookiin useEffect, HTTP-pyyntöjen lähettämiseen Axios-kirjastoa käyttämällä ja asynkroniseen JavaScriptiin.

Kun sovelluksen osien keskinäinen kommunikointi on toteutettu, tehtävässä kuusi teknologiapinoon voidaan lisätä MongoDB-tietokanta. Tehtävä sisältää ohjeet tietokannan asentamiseen MongoDB Atlas -palvelussa. Asennuksen jälkeen palvelin yhdistetään tietokantaan, luodaan sovellukseen tallennettavia elokuvia vastaava tietokantamalli ja toteutetaan CRUD- eli tietokantaoperaatiot. Operaatioita tehtäessä tutustutaan Mongoosen eri metodeihin, joilla tietokantaan voidaan tehdä muutoksia. Toteutuksessa esitellään uudet HTTP:n PUT- ja DELETE-metodit ja niiden käsittelijät sekä harjoitellaan HTTP-pyntöjen virheenkäsittelyä. Lisäksi myös back endin olemassa olevaa koodia uudelleen järjestetään noudattamaan MVC (model-view-controllers) -sovellusarkkitehtuuria. Tehtävän jälkeen opiskelija on implementoinut toimivan full stack-sovelluksen.

Käyttäjänhallinta aloitetaan tehtävässä seitsemän sovelluksen back endistä, johon luodaan käyttäjäkyselyihin liittyen uusi tietokantamalli ja HTTP-pyyntöjen polku sekä käsittelijät käyttäjien rekisteröintiin ja kirjautumiseen. Käyttäjän tietokantamalli sisältää käyttäjätunnuksen, sähköpostiosoitteen ja salasanan. Turvallisuussyistä salasanaa ei tallenneta sellaisenaan tietokantaan. Salasana salataan käyttäen ulkoista Bcrypt-kirjastoa, eikä edes salattua salasanaa palauteta missään tilanteessa takaisin selaimelle. Nämä ovat turvallisuuden kannalta vähimmäisseikat, jotka on hyvä tuntea ja opiskella heti ensimmäisestä projektista lähtien.

Tehtävässä kahdeksan sovelluksen rekisteröinti- ja kirjautumismahdollisuudet toteutetaan front endiin. Tehtävässä luodaan sekä rekisteröitymiselle että kirjautumiselle omat näkymänsä, jolloin sovelluksessa on mahdollista navigoida edellä mainittujen lomakkeiden ja sovelluksen etusivun välillä. Navigointi tapahtuu sovelluksen yläpalkkiin lisättävien linkkien avulla. Rekisteröinti- ja kirjautumissivuilla käyttäjää ohjataan käyttöliittymässä sivujen välillä ohjelmallisesti React-routerin useHistory -hookia käyttämällä. Lisäksi elokuvien lisääminen estetään kirjautumattomilta käyttäjiltä.

Tehtävän yhdeksän aloittaessa opiskelija on toteuttanut sovellukseen toimivan käyttäjähallinnan, mutta tietokantamalleja laajennetaan vielä ristikkäisillä tiedoilla käyttäjistä ja käyttäjien lisäämistä elokuvista. Muutoksen jälkeen elokuvat sisältävät tunnuksen sen lisäämästä käyttäjästä ja vastaavasti käyttäjätietoihin tallennetaan taulukko, joka sisältää käyttäjän lisäämien elokuvien tunnukset. Lopuksi ristikkäin tallennetut tunnukset korvataan niitä vastaavilla kokonaisilla tai osittaisilla dokumenteilla käyttäen Mongoosen populate-metodia, jolloin tietokanta sisältää relatiivista dataa. Datan populointi mahdollistaa, että sovellukseen lisätyt elokuvat luetaan ja näytetään käyttäjäkohtaisesti käyttäjän kirjautuessa sovellukseen sisään.

Viimeisessä mallisovelluksen implementointiin liittyvässä tehtävässä sovellus julkaistaan internettiin käyttämällä Heroku-alustapalvelun web-käyttöliittymää ja tarvittaessa komentorivityökalua. Tehtävässä React-sovelluksesta tehdään optimoitu tuotantoversio, jonka koodi siirretään palvelinprojektin juureen. Näin ollen sovelluksen eri osa toimivat samassa palvelimen osoitteessa. Koodimuutosten lisäksi tehtävä sisältää kokonaisuudessaan ohjeet Heroku-palvelun käyttöön, julkaisuun ja mahdollisten virhetilanteiden käsittelyyn.

Materiaalin viimeisessä tehtävässä opiskelija implementoi materiaalissa tarjottujen ohjeiden, esimerkkien, lähteiden ja aikaisemmin opitun perusteella vapaavalintaisen full-stack web-sovelluksen. Sovelluksen vaatimukset vastaavat mallisovellukseen tehtyjä toiminallisuuksia siinä käytetyillä teknologioilla ja tekniikoilla.

6 POHDINTA

Opinnäytetyön tavoitteena oli tuottaa sen toimeksiantajalle Oulun Ammattikorkeakoululle opintomateriaali full stack web-sovelluskehitykseen MERN-teknologioita käyttäen. Modernin full stack-kehityksen opetus koetaan tärkeäksi toimeksiantajan toimesta, mutta opinnäytetyössä kuvattua, tai sitä vastaavaa, teknologiakokonaisuutta ei kuitenkaan työn kirjoitushetkellä tarjota opiskeltavaksi.

Opinnäytetyön tuloksena koostettiin sen laatijan tavoitteiden mukainen opintomateriaali. Materiaalia voidaan tarjota opiskeltavaksi ammattikorkeakouluopiskelijoille, mutta sen käytöstä ja käyttökelpoisuudesta päättää työn toimeksiantaja.

Full stack-kehityksen perusteiden opettamisen ja opiskelun haasteena on sen vaatiman materiaalin laajuus. Kun kehityksessä käytettäviä teknologioita on useita, myös opittavan materiaalin määrä väistämättä kasvaa. On tärkeää, että materiaalin toteuttamisessa pyritään johdonmukaisuuteen ja jatkuvaan harkintaan, mitkä asiat ovat kehityksen ja käyttötapausten kannalta priorisoituja. Opiskelumateriaalin progressiivisesti kehittyvillä haasteilla voidaan pyrkiä koko opiskelun ajan tarjoamaan opiskelijalle onnistumisen tunteita ja pitämään opiskelijan kiinnostusta yllä. Myös opiskelumotivaatio on tärkeä osa oppimista. Kun opiskelija tiedostaa, että opiskeltava asia on ajankohtainen, ratkaistavat ongelmat todellisiin käyttötapauksiin perustuvia ja niistä saatava osaaminen esimerkiksi työnantajien puolesta kysyttyä, motivaatio oppia on todennäköisemmin korkeampi. Ajankohtaisella materiaalilla ja esimerkkisovelluksilla on varmasti vaikutus myös innostavuuteen, joka puolestaan taas ruokkii opiskelijan oppimismotivaatiota.

JavaScript-teknologiapinot ovat otollisia teknologioita full stack-kehityksen opiskelun aloittamiseen, koska kehittäminen vaatii vain yhden ohjelmointikielen. Samalla ohjelmointisyntaksilla opiskelijan on mahdollista tutustua sekä sovelluksen selaimessa näkyvään että palvelinpuolen koodiin ja niiden vaatimaan toimintalogiikkaan. Tietokantaohjelmoinnin opiskelun voidaan kokea olevan helpommin lähestyttävissä, kun se aloitetaan NoSQL-kannoista joustavilla tietokantamalleilla verrattaessa jokseenkin enemmän pohjatyötä ja teoriatietoa vaativiin relatiivisiin kantoihin.

Opinnäytetyössä koostetussa materiaalissa kaikki käytetyt teknologiat, työkalut ja palvelut ovat ilmaiseksi käytettävissä, eivätkä ne edellytä niiden käyttäjiltä esimerkiksi luottokorttitietojen luovut-

tamista. Teknologiat ovat sekä yritysten että yksittäisten kehittäjien suosiossa globaalisti ja oppimiseen tarvittavaa materiaalia on tarjolla runsaasti. Teknologioiden opiskeluun laajemmin ammattikorkeakouluympäristössä ei näiden asioiden valossa tulisi olla esteitä.

LÄHTEET

Express 2021a. Fast, unopinionated, minimalist web framework for Node.js. Hakupäivä 20.5.2021. https://expressjs.com/.

Express 2021b. Basic routing. Hakupäivä 20.5.2021. https://expressjs.com/en/starter/basic-routing.html.

Express 2021c. Using middleware. Hakupäivä 20.5.2021. https://expressjs.com/en/guide/using-middleware.html#middleware.third-party.

Hamedani, Mosh 2018. React Virtual DOM Explained in Simple English. Hakupäivä 24.5.2021. https://programmingwithmosh.com/react/react-virtual-dom-explained/

MDN Web Docs 2021a. Express/Node introduction. Hakupäivä 20.5.2021. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction.

MDN Web Docs 2021b. How the Web works. Hakupäivä 6.5.2021. https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works.

Mittal, Aman 2020. A guide to using React Router v6 in React apps. Hakupäivä 20.5.2021. https://blog.logrocket.com/react-router-v6/.

MongoDB 2021. Kuvakaappaus. Query Documents. Hakupäivä 24.5.2021. https://developer.mongodb.com/article/mongoose-versus-nodejs-driver/

MongoDB 2021. Kuvakaappaus. Do You Need Mongoose When Developing Node.js and MongoDB Applications? Hakupäivä 24.5.2021. https://developer.mongodb.com/article/mongoose-versus-nodejs-driver/

MongoDB 2021. Kuvakaappaus. Introduction to MongoDB. Hakupäivä 5.5.2021.

MongoDB 2021a. MERN Stack. Hakupäivä 4.5.2021. https://www.mongodb.com/mern-stack.

MongoDB 2021b. Introduction to MongoDB. Hakupäivä 5.5.2021. https://docs.mongodb.com/manual/introduction/.

MongoDB 2021c. MongoDB vs MySQL Differences. Hakupäivä 20.5.2021. https://www.mongodb.com/compare/mongodb-mysql.

MongoDB 2021d. NoSQL vs SQL Databases. Hakupäivä 5.5.2021. https://www.mongodb.com/nosql-explained/nosql-vs-sql.

MongoDB 2021e. Do You Need Mongoose When Developing Node.js and MongoDB Applications? Hakupäivä 24.5.2021. https://developer.mongodb.com/article/mongoose-versus-nodejs-driver/

Mongoose 2021. Populate. Hakupäivä 5.5.2021. https://mongoosejs.com/docs/populate.html.

Nodejs 2021a. Introduction to Node.js. Hakupäivä 6.5.2021. https://nodejs.dev/learn/introduction-to-nodejs.

Nodejs 2021b. The V8 JavaScript Engine. Hakupäivä 6.5.2021. https://nodejs.dev/learn/the-v8-javascript-engine

Nodejs 2021c. Differences between Node.js and the Browser. Hakupäivä 6.5.2021. https://nodejs.dev/learn/differences-between-nodejs-and-the-browser.

Nodejs 2021d. File system. Hakupäivä 6.5.2021. https://nodejs.org/api/fs.html.

Nodejs 2021e. Http. Hakupäivä 6.5.2021. https://nodejs.org/api/http.html.

Npm Docs 2021. About NPM. Hakupäivä 6.5.2021. https://docs.npmjs.com/about-npm.

Reactjs 2021a. Hooks FAQ. Hakupäivä 4.5.2021. https://reactjs.org/docs/hooks-faq.html.

Reactjs 2021b. Introducing JSX. Hakupäivä 7.5.2021. https://reactjs.org/docs/introducing-jsx.html.

Reactjs 2021c. Components and Props. Hakupäivä 7.5.2021. https://reactjs.org/docs/components-and-props.html.

Reactjs 2021d. Hooks at Glance. Hakupäivä 7.5.2021. https://reactjs.org/docs/hooks-overview.html.

Reactjs 2021e. Hooks API Reference. Hakupäivä 7.5.2021. https://reactjs.org/docs/hooks-reference.html.

Reactjs 2021f. Lifting State up. Hakupäivä 7.5.2021. https://reactjs.org/docs/lifting-state-up.html.

React 2021g. Styling And CSS. Hakupäivä 20.5.2021. https://reactjs.org/docs/faq-styling.html.

React Training 2021. History. Hakupäivä 20.5.2021. https://reactrouter.com/web/api/history.

React Training 2021. UseParams. Hakupäivä 20.5.2021. https://reactrouter.com/web/api/Hooks/useparams.

Sagar, Paresh 2020. What Is a Single Page Application? Meaning, Pitfalls & Benefits. Hakupäivä 7.5.2021. https://www.excellentwebworld.com/what-is-a-single-page-application/.

Saring, Jonathan 2018. 5 Ways to Style React Components in 2020. Hakupäivä: 8.5.2021. https://blog.bitsrc.io/5-ways-to-style-react-components-in-2019-30f1ccc2b5b.

Snyk 2019. Npm passes the 1 millionth package milestone! What can we learn? Hakupäivä: 8.5.2021. https://snyk.io/blog/npm-passes-the-1-millionth-package-milestone-what-can-we-learn/.

Stack Overflow 2020. Kuvakaappaus. Most Loved, Dreaded, and Wanted Databases. Hakupäivä 5.5.2020.

Stack Overflow 2020. Kuvakaappaus. Databases. Hakupäivä 5.5.2020.

Stack Overflow 2020. 2020 Developer Survey. Hakupäivä 4.5.2021. https://insights.stackoverflow.com/survey/2020#technology-web-frameworks-all-respondents2.

Vetter-Neo, Natalia 2020. Sunscrapers. When to use a UI component library in a React project? Hakupäivä 25.5.2021. https://sunscrapers.com/blog/when-to-use-a-ui-component-library-in-a-react-project/#:~:text=UI%20component%20libraries%20are%20no,and%20match%20the%20existing%20components.

Quisenberry, David 2018. Daylight Studio. What is State and Why Does it Matter? Hakupäivä 25.5.2021. https://thedaylightstudio.com/blog/2018/03/14/what-is-state-in-web-application-development.