Tampereen ammattikorkeakoulu

Verkkosivuprojektin toteuttaminen MERN-teknologiapaketilla

Aapo Attila

OPINNÄYTETYÖ
Huhtikuu 2021

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

ATTILA, AAPO:
Verkkosivuprojektin toteuttaminen MERN-teknologiapaketilla

Opinnäytetyö 48 sivua
Huhtikuu 2021

Opinnäytetyössä toteutettiin full-stack-järjestelmä, joka suunniteltiin ja toteutettiin projektin asiakkaalta saadun vaatimusmäärittelyn mukaisesti. Järjestelmä toteutettiin MERN-teknologiapaketilla, jonka todettiin toimivan hyvin järjestelmälle asetettujen vaatimusten toteuttamiseen. Toteutettu järjestelmä koostuu Reactilla toteutetusta web-käyttöliittymästä, sekä Node.js:llä, Expressillä ja MongoDB:llä toteutetusta palvelin- ja tietokantaosasta.

Työssä toteutettua järjestelmää käsitellään kolmessa osassa, jotka ovat järjestelmän arkkitehtuuri, käyttöliittymän toteutus ja palvelin- ja tietokantaosuuden toteutus. Osuuksia käsitellään koodiesimerkkien, käyttöliittymäkuvien sekä järjestelmän ja sen komponenttien toimintaa havainnollistavien kuvaajien kautta. Toteutettua järjestelmää koskevien esimerkkien lisäksi työssä käsitellään yksinkertaisten React- ja Express-sovellusten luominen, joiden pohjalta voidaan toteuttaa opinnäytetyössä toteutetun projektin kaltainen kokonaisuus.

Järjestelmälle määritetyt sisällölliset ja toiminnalliset vaatimukset täytettiin, mutta aikataulullisista rajoitteista johtuen järjestelmää ei saatu vietyä tuotantoympäristöön. Järjestelmä tullaan jatkokehityksen yhteydessä saattamaan tuotantoon, ja opinnäytetyössä käsitellään myös tähän vaadittavia toimenpiteitä.

Tehdyt teknologiavalinnat koettiin tämän projektin tarkoituksiin onnistuneiksi, mutta tapauksissa joissa asiakkaan on voitava ylläpitää suurta osaa sivustosta itsenäisesti, suositellaan muita ratkaisuja kuten WordPressiä.

Asiasanat: full-stack, react, node.js, vaatimusmäärittelyt

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT Engineering
Software Engineering

ATTILA, AAPO:
Building a Full-stack Application with the MERN-stack

Bachelor's thesis 48 pages
April 2021

The objective of this thesis project was to design and develop a web-based system to customer specification. The system was built using the MERN-stack, which was concluded to fulfill the requirements set for the project. The developed system consists of a front-end built with React and a back-end built with Node.js, Express and MongoDB.

The thesis discusses the system in three parts, which are the overall system architecture, a detailed explanation of the user interface, and a closer look at the back-end developed for the system. Code examples, architectural diagrams and images of the user interface are used to elaborate on the structure and operation of the system. In addition, the thesis also provides instructions on setting up simple React and Express applications. These examples form the basis for the system described in this thesis.

The requirements for the system were met, but due to optimistic planning regarding the timeline of the project, the system was not put into production. The system will be put into production in future development, and the steps necessary to do so are discussed in this thesis.

The technology choices made in the project were successful, but in cases where the customer needs to do more extensive maintenance on their own, a solution like WordPress is recommended.

Key words: full-stack, react, node.js, requirement specification

SISÄLLYS

1 JOHDANTO 7
2 TAUSTATIEDOT JA SUUNNITTELUN LÄHTÖKOHDAT 8
2.1. Asiakkaan tarve 8
2.2. Toteuttajan näkökulma 9
2.3. Käyttöliittymän suunnittelu 9
3 TEKNOLOGIAT JA KOMPONENTIT 11
3.1. JavaScript 11
3.2. React 11
3.3. Node.js 12
3.4. Express 12
3.5. MongoDB 12
3.6. Docker 13
4 YLEINEN ARKKITEHTUURI 14
4.1. MERN-teknologiapaketti 14
4.1.1 React 15
4.1.2 Node.js ja Express 16
4.1.3 MongoDB 17
4.2. Toteutunut arkkitehtuuri 18
4.3. Tavoiteltu arkkitehtuuri ja järjestelmän tuotantoon saattaminen 19
4.3.1 Nginx 19
4.3.2 Certbot 19
4.4. Tuotantovalmiin järjestelmän arkkitehtuuri 20
5 KÄYTTÖLIITTYMÄ 21
5.1. Koodauskäytännöt 21
5.1.1 Git 21
5.1.2 ESlint 22
5.2. Create React App 22
5.3. Projektityön käyttöliittymä 23
5.3.1 Material-UI 25
5.3.2 React Hooks 26
5.3.3 React-i18next 27
5.4. Käyttöliittymän näkymät 28
5.4.1 Etusivu 28
5.4.2 Projektit 29
5.4.3 Kalenteri 30
5.4.4 Yhteydenotto 31

5.5. Ylläpitäjälle tarkoitetut näkymät 33
5.5.1 Kirjautumisnäkymä 33
5.5.2 Lisää tapahtuma -näkymä 34
6 PALVELINOSUUS JA TIETOKANTA 36
6.1. Yksinkertaisen Express-sovelluksen luominen 36
6.2. Koodauskäytännöt 37
6.2.1 Dotenv 38
6.3. Express 38
6.3.1 Reititys 39
6.3.2 Tunnistautuminen 40
6.4. MongoDB 41
7 POHDINTA 44
7.1. Vaatimusten toteutuminen 44
7.2. Teknologiavalinnat 44
7.3. Aikataulu 45
7.4. Kehittäjän kokemus 45
7.5. Jatkokehitys 46
LÄHTEET 47

LYHENTEET JA TERMIT

API	Application Programming Interface Ohjelmointirajapinta
CSS	Cascading Style Sheets WWW-dokumenttien tyyliohjeiden laji
Full-stack	Sekä käyttöliittymän, palvelimen että tietokannan kattava ohjelmointiparadigma
JSON	JavaScript Object Notation Avoimen standardin tiedostomuoto tiedonvälitykseen
JWT	JSON Web Token Kahden osapuolen väliseen turvalliseen tiedonsiirtoon käytetty työkalu
MERN	MongoDB, Express, React, Node.js Joukko usein yhdessä käytettyjä teknologioita
NoSQL	Relaatiomallia noudattamattomat tietokannat
SPA	Single Page Application Yksisivuinen verkkosivu
SSL	Secure Sockets Layer Tietoliikenteen salausprotokolla
SQL	Structured Query Language Relaatiotietokantojen haku-, muutos- ja lisäyskieli
URL	Uniform Resource Locator Tiedon hakemiseen tarvittavan tiedon ilmaiseva merkkijono

1 JOHDANTO

Opinnäytetyössä toteutettiin full-stack-verkkosivuprojekti henkilöasiakkaan freelance-muotoisen työn markkinointiin. Sivuston sisällön suunnittelu toteutettiin yhteistyössä asiakkaan kanssa, mutta teknisen toteutuksen sekä sivuston ulkoasun suunnitteluun ja toteutuksiin saatiin vapaat kädet.

Asiakkaan toivoma järjestelmä toteutettiin MERN-teknologiapaketilla (MongoDB, Express, React, Node.js). Teknologiapakettia käyttäen järjestelmälle toteutettiin käyttöliittymä, palvelintoteutus ja tietokanta. Järjestelmää oli tarkoitus laajentaa tuotantoon vietäessä muillakin teknologioilla, mutta aikataulullisista rajoitteista johtuen järjestelmää ei saatu tuotantovalmiiksi opinnäytetyön puitteissa.

Opinnäytetyössä esitellään kehitetyn järjestelmän toteutukseen ja sen ymmärtämiseen vaaditut teknologiat, niiden roolit projektissa sekä koodiesimerkkejä niiden käytöstä. Suurin osa projektin parissa vietetystä ajasta kului järjestelmän palvelin- ja tietokantakomponenttien kehitystyöhön, sillä tämä oli projektiin ryhdyttäessä vähiten tuttua.

Koska työssä esitellään sivujen tekniseen toteutukseen ja tietoturvaseikkoihin liittyviä yksityiskohtia, on joitakin projektiin liittyviä tietoja anonymisoitu. Esittelemättä jätetään sivujen lopullinen URL-osoite, asiakkaan nimi ja ammatti, sekä muut mahdollisesti yksilöivät tiedot. Myös sivujen viimeistelty ulkoasu jätetään esittelemättä tekijänoikeuksien sekä asiakkaan identiteetin suojelemiseksi. Dokumentissa paneudutaan siis lähinnä sivujen tekniseen toteutukseen, ja kaikki ulkoasusta esiteltävä materiaali on esimerkinomaista eikä edusta viimeistä, tuotantoon päätyvää versiota.

2 TAUSTATIEDOT JA SUUNNITTELUN LÄHTÖKOHDAT

Tässä kappaleessa esitellään projektin aloittamisen alkusyyt, asiakkaan projektille esittämät tarpeet sekä näiden tarpeiden pohjalta johdetut suunnittelulähtökohdat. Projektin suunnittelutyötä käsitellään sekä asiakkaan että kehittäjän näkökulmista.

2.1. Asiakkaan tarve

Projekti sai alkusysäyksensä asiakkaan pyynnöstä toteuttaa hänelle edustuskelpoiset kotisivut. Hän tarvitsi freelance-luontoisten projektiensa markkinointiin verkkosivut, joilla hän voisi esitellä asiakkaille ja yhteistyökumppaneille esimerkiksi tulevia esiintymisiään ja aiemmin toteutettuja projekteja. Asiakkaan toive sivujen sisällöstä oli seuraavanlainen:

- etusivu
- perustietoja-osio, jossa esitelty asiakkaan ansioluettelo sekä yleistä tietoa asiakkaasta
- projektit-osio, jossa tietoa asiakkaan aiemmin toteuttamista projekteista
- kalenteriosio, jossa asiakkaan tulevia ja menneitä julkisia esiintymisiä
- yhteydenotto-osio, jonka kautta kiinnostunut sivuilla kävijä voisi ottaa yhteyttä asiakkaaseen

Asiakas ei itse ole teknisesti etevä henkilö, joten edellä mainitut ominaisuudet täyttävien sivujen tekniseen suunnitteluun saatiin vapaat kädet. Sivujen ulkoasun suunnittelu sekä joitakin tekniseen toteutukseen liittyviä yksityiskohtia kuitenkin hyväksytettiin asiakkaalla ennen kehitystyön aloitusta. Teknisen suunnittelun vaatimukset olivat siis lähtökohtaisesti hyvin löysät, mutta niiden täyttämiseksi pyrittiin silti valitsemaan perustellut ratkaisut.

2.2. Toteuttajan näkökulma

Asiakkaan esittämien sisällöllisten vaatimusten lisäksi järjestelmälle asetettiin asiakkaan kanssa käydyn keskustelun perusteella seuraavat toiminnalliset vaatimukset:

- Sivun tulee olla saatavilla sekä suomeksi että englanniksi.
- Osaa sivuista tulee voida päivittää asiakkaan toimesta.

Asiakas oli projektia suunnitellessa tietoinen aiemmin saman tyyppiseen tarkoitukseen toteutetusta projektista, jota hän käytti omien tarpeidensa ja vaatimustensa muotoilemiseen. Aiemmin toteutetussa projektissa oli käytetty MERN-teknologiapakettia (MongoDB, Express.js, React, Node.js), joten asiakkaan toivomusten mukainen projekti päätettiin toteuttaa hyödyntäen samoja teknologioita.

Ulkoasun suunnittelussa otettiin aiemmin toteutettuun projektiin verrattuna erilainen lähestymistapa, jonka uskottiin palvelevan asiakkaan tarpeita paremmin. Suunnittelutyössä otettiin huomioon asiakkaan edustama toimiala, jota pyrittiin tuomaan esiin ulkoasun suunnitteluratkaisujen kautta. Sivujen antama vaikutelma pyrittiin myös pitämään jokseenkin virallisempana kuin aiemmin toteutetun projektin, sillä opinnäytetyöprojektin asiakaskunnan ajateltiin edustavan eri ikäluokkaa ja arvopohjaa.

Sivujen ulkoasun suunnittelutyökaluna käytettiin Adobe XD:n ilmaisversiota, joka on maksettuun versioon verrattuna jokseenkin rajoittunut, mutta täysin riittävä tämän kaltaisen projektin suunnitteluun.

2.3. Käyttöliittymän suunnittelu

Sivusto päätettiin jakaa neljään osioon, joiden sisältö määräytyi kappaleessa 2.1 esitellyn listan mukaisesti sillä poikkeuksella, että perustietoja-osio päätettiin yhdistää etusivuun. Näin ollen vierailija näkisi heti asiakkaan profiilin, ja saisi suurpiirteisen käsityksen asiakkaasta ja tämän tarjoamista palveluista.

Projektit-osioon sijoitettavien projektien esittely päätettiin toteuttaa kortti-tyylisesti. Korttiin tulisi projektiin liittyvä kuva, lista projektin yhteistyökumppaneista, toteutusvuosi sekä tarvittaessa linkki projektin lopputulokseen.

Kalenteri-osio suunniteltiin sisältämään kronologinen lista tulevista ja menneistä tapahtumista. Jokaisesta tapahtumasta esitettäisiin tapahtuman nimi tai otsikko, päivämäärä ja kellonaika, tapahtuman mahdolliset yhteistyökumppanit sekä kuvaus tapahtuman sisällöstä.

Yhteydenotto-osio päätetiin toteuttaa lomakkeena, jossa vierailija antaisi tietoinaan nimen, sähköpostiosoitteen, edustamansa organisaation sekä vapaaehtoisen viestin, jonka haluaisi välittää asiakkaalle. Lomakkeen lähetysnappulaa painaessa ruudulle tulisi modaali, jossa vierailija pyydetään tarkistamaan antamiensa tietojen oikeellisuus ennen lähettämistä.

Kaikkien sivujen yläreunassa sijaitsisi kaikille osioille yhteinen navigaatiopalkki, johon on painikkeiden muodossa toteutettu navigaatiolinkit kuhunkin osioon, sekä painike sivuston kielen vaihtamista varten.

3 TEKNOLOGIAT JA KOMPONENTIT

3.1. JavaScript

JavaScript on syntaksiltaan Javaa sekä C++:aa muistuttava kevyt, dynaaminen ohjelmointikieli, jonka käyttökohteet ovat pääasiassa selainsovelluksissa (Mozilla, 2021). Moniparadigmaisena ohjelmointikielenä se tarjoaa kehittäjille joustavuutta monien eri ohjelmointitapojen muodossa.

Selainasovelluksia toteutettaessa JavaScriptiä käytetään esimerkiksi käyttäjän selaimessa suoritettavien animaatioiden ja käyttöliittymäelementtien toteuttamiseen. Sen avulla voidaan siis luoda interaktiivisempia käyttäjäkokemuksia kuin pelkällä HTML:llä ja CSS:llä.

3.2. React

React on avoimen lähdekoodin JavaScript-kirjasto käyttöliittymien ja käyttöliittymäkomponenttien toteuttamiseen (Facebook, 2021c). Reactin kantava ajatus on sovelluksen pilkkominen pienempiin palasiin, komponentteihin. Komponenteilla on yleensä oma render-metodinsa, jonka ansiosta komponentteja voidaan päivittää yksitellen.

Reactin etuihin kuuluu sen helppo ja sujuva tilanhallinta, jonka ansiosta esimerkiksi käyttäjän kirjautumisstatuksen muuttuessa selaimessa päivitetään vain vaadittavat komponentit. Oikein käytettynä tämä ominaisuus nopeuttaa sivun toimintaa huomattavasti perinteisiin menetelmiin verrattuna, joissa koko sivun sisältö on päivitettävä yhden komponentin tai datapisteen muuttuessa.

3.3. Node.js

Node.js on avoimen lähdekoodin JavaScript-ajoympäristö, joka perustuu Googlen V8 JavaScript-moottoriin (OpenJS Foundation, 2021b). Se on tapahtumapohjaisen arkkitehtuurinsa sekä asynkronisuutensa ansiosta saavuttanut suuren suosion muun muassa reaaliaikaisten kommunikaatiosovellusten ajoympäristönä.

Modernien JavaScript-käyttöliittymäkirjastojen käytön yleistyessä on Node.js:n suosio kasvanut myös sen kehitysympäristöön tuomiensa synergiaetujen kautta. Kehittäjien on mahdollista kirjoittaa sekä käyttöliittymä- että palvelinpään koodia samalla ohjelmointikielellä, JavaScriptillä. Tämä yksin lienee vaikuttanut huomattavasti full-stack-roolien yleistymiseen ohjelmistoteollisuudessa.

3.4. Express

Express tai Express.js on kevyt, web-sovellusten palvelintoteutuksiin erikoistunut ohjelmistokehys, joka on rakennettu Node.js:n päälle. Itsessään melko minimalistinen Express on vakiinnuttanut paikkansa kehittäjien ensisijaisena valintana web-palvelimeksi Node.js-ajoympäristöissä valtavan lisäosamääränsä ansiosta. Se tarjoaa kehittäjille mahdollisuuden poimia käyttöönsä vain tarvittavat lisäosat pitäen kokonaisuuden kevyenä.

Express on olennainen komponentti muun muassa MERN- ja MEAN-teknologiapaketeissa, joiden suuri käyttöaste on kasvattanut myös Expressin suosiota viime vuosina.

3.5. MongoDB

MongoDB on dokumenttipohjainen tietokantaratkaisu, joka luokitellaan NoSQL-tietokannaksi (MongoDB, 2021a). Se käyttää tietokantataulujen sijaan tiedon tallentamiseen JSON-objektien tyyliin formatoituja dokumentteja. MongoDB on suo-

sittu ratkaisu käyttökohteissa, joissa tallennettavien tietojen välille ei tarvitse muodostaa monimutkaisia suhteita. Toisin sanoen MongoDB on erinomainen tietokanta tapauksiin, joissa ei ole tarvetta relaatiotietokannalle.

3.6. Docker

Docker on 2013 julkaistu ohjelmistotuote, joka tarjoaa kehittäjille mahdollisuuden kehitys- ja ajoympäristöjen standardisointiin (Docker, 2021). Ajoympäristöjen standardisointi helpottaa kehittäjien työtä poistamalla ajoympäristöjen eroista johtuvat virheet. Standardisoituja ohjelmistokomponentteja kehitetään ja julkaistaan paketeissa, joita kutsutaan konteiksi (container).

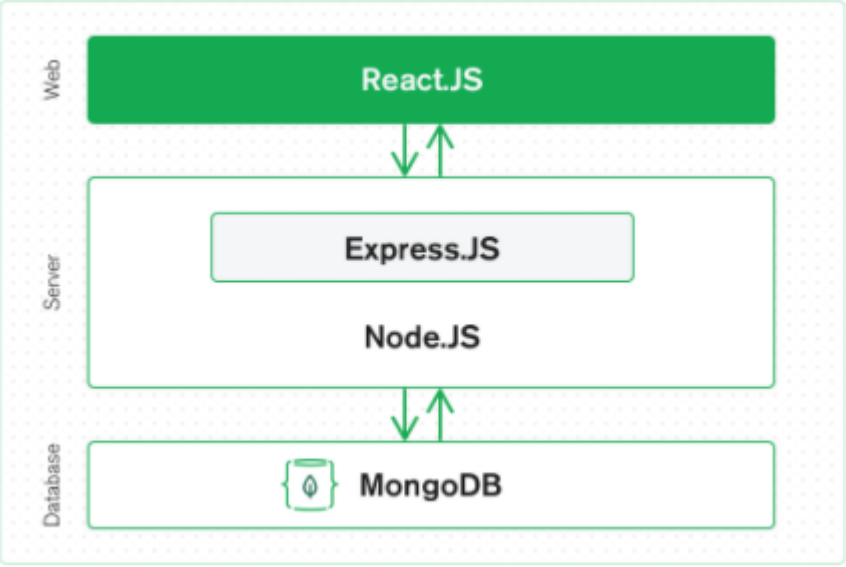
Kontteja on saatavilla kokonaisista käyttöjärjestelmistä tarkemmin rajattuihin tuotteisiin, jotka on tarkoitettu suorittamaan tiettyjä toiminnallisuuksia. Tämän projektin kontekstissa kontteina suoritetaan sovelluksen käyttöliittymää, palvelinta sekä tietokantaa. Kontit voivat kommunikoida keskenään määritettyjen kanavien kautta, joka mahdollistaa kokonaisten ohjelmistokokonaisuuksien ajamisen joukossa keskenään kommunikoivia kontteja, kuten tässäkin projektissa tehdään.

4 YLEINEN ARKKITEHTUURI

Tässä kappaleessa kuvataan projektin teknisen toteutuksen yleistä arkkitehtuuria, sen muodostavia komponentteja sekä valittuun arkkitehtuuriin johtaneita suunnitteluratkaisuja. Sekä käyttöliittymän että palvelinosuuden tarkempia toteutuksia käsitellään yksityiskohtaisesti kappaleissa 5 ja 6.

4.1. MERN-teknologiapaketti

MERN-teknologiapaketti on web-kehityksessä suosittu joukko moderneja, pääosin JavaScriptin ja JSON:in käyttöön pohjautuvia teknologioita (MongoDB, 2021b). Sitä käytettäessä päädytään järjestelmän arkkitehtuurin suunnittelussa kolmikerroksiseen ratkaisuun, jossa web-kerroksen muodostaa React-käyttöliittymä, palvelinkerroksen Node.js ja Express ja tietokantakerroksen MongoDB (kuvio 1).



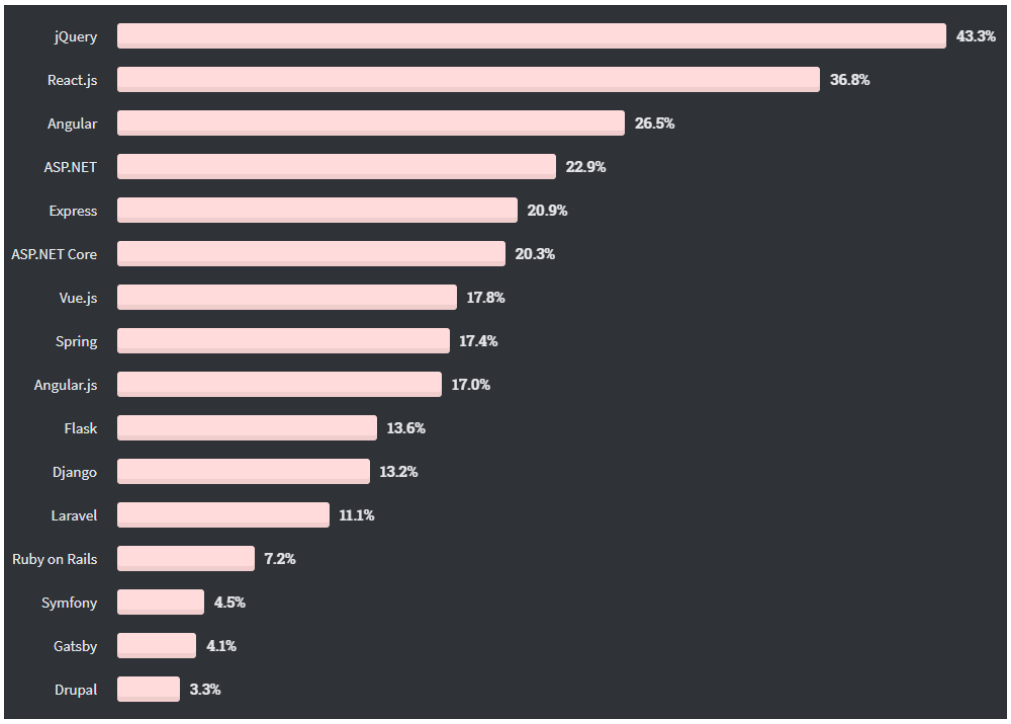
Kuvio 1. MERN-järjestelmän rakenne (MongoDB, 2021b)

MERN-teknologiapaketin suuri suosio nojaa sen kehittäjille asettamiin vaatimuksiin: jos kehittäjä hallitsee JavaScriptin ja ymmärtää JSON:in toimintaperiaatteet, osaa hän käytännössä luoda sovelluksia MERN-teknologiapaketilla. Koska osaa-

misvaatimukset ovat verrattain matalat, ja teknologioilla pystyy toteuttamaan laajoja, moderneja ohjelmistokokonaisuuksia, ei kyseisen teknologiapaketin suuri suosio ole ihme.

4.1.1 React

Tuotantokelpoisten järjestelmien toteuttamiseksi MERN-teknologiapaketilla on Reactin hallitseminen kehittäjän suurin osaamisvaatimus ja haaste. React on kasvattanut markkinaosuuttaan tasaisesti viime vuosina, ja on ajamassa web-ohjelmistokehyksien kilpailussa jQueryn ohi (kuvio 2).

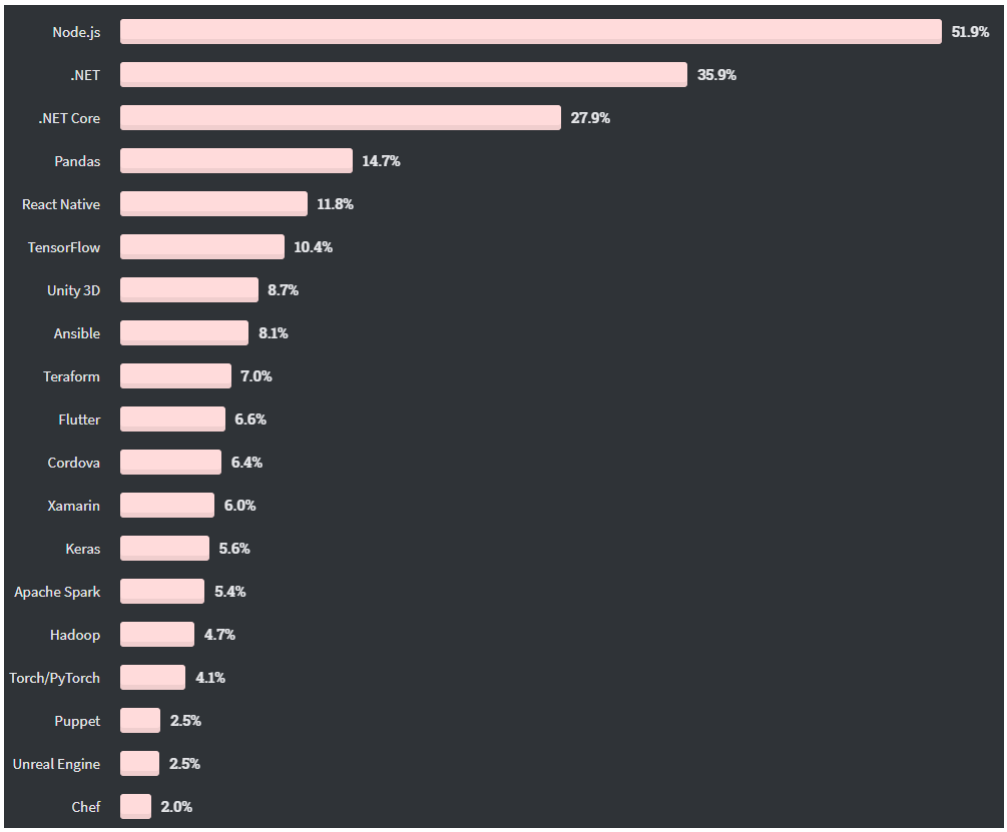


Kuvio 2. Web-ohjelmistokehysten markkinaosuudet (Stack Overflow, 2020a)

Tulokseen johtaneessa tutkimuksessa jää epäselväksi, lasketaanko tulokseen vain vuoden aikana aloitetut projektit vai kaikki käynnissä olevat projektit, mutta Reactin valtaama markkinaosuus on kummassakin tapauksessa vaikuttava. Laajan suosionsa ansiosta Reactin hallitseminen on tärkeä kriteeri myös useimmissa käyttöliittymäkehittäjiä etsivissä työpaikkailmoituksissa, mikä kannustaa myös aloittelevia kehittäjiä sen omaksumiseen.

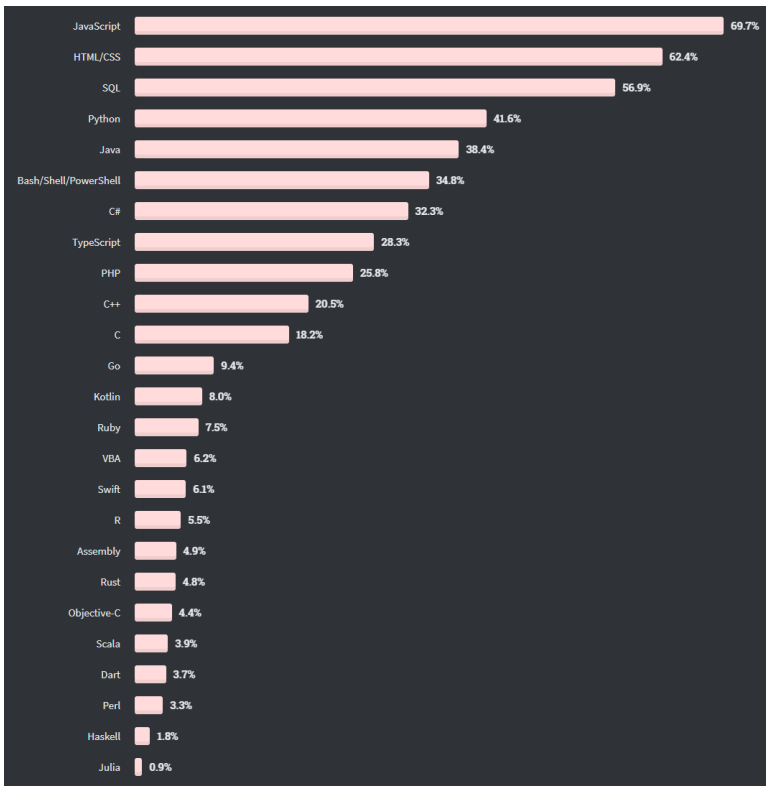
4.1.2 Node.js ja Express

Node.js ja Express ovat kehittäjien keskuudessa erittäin suosittuja teknologioita, joiden käyttöaste on pysynyt korkeana vuodesta vuoteen. Express sijoittuu web-ohjelmistokehysten vertailussa viidenneksi (kuvio 2) ja Node.js on ammattimaisten ohjelmistokehittäjien eniten käyttämä monikäyttöinen ohjelmistokehys 51,9 prosentin käyttöasteella (kuvio 3).



Kuvio 3. Monikäyttöisten ohjelmistokehysten markkinaosuudet (Stack Overflow, 2020c)

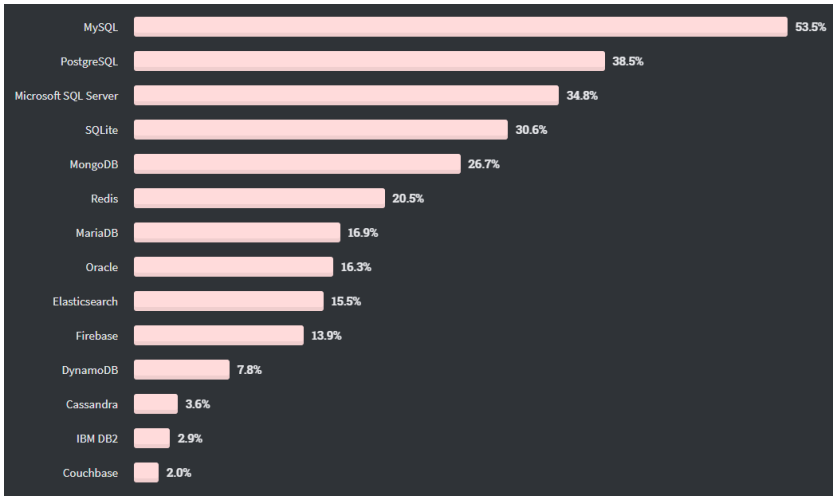
Node.js:n suuri suosio pohjaa sen monikäyttöisyyteen, sekä JavaScriptin suureen suosioon kehittäjien keskuudessa (kuvio 4). Se voidaankin lukea jo lähes välttämättömäksi osaksi modernin web-kehittäjän työkalupakkia, eikä sen suosion laskusta näy merkkejä. Express puolestaan on yleinen valinta Node.js:n web-palvelinkehykseksi, joten teknologioiden suosio kulkee käsi kädessä.



Kuvio 4. Ohjelmointi- ja skriptauskielten markkinaosuudet (Stack Overflow, 2020d)

4.1.3 MongoDB

MongoDB jää muista MERN-teknologiapaketin teknologioista poiketen markkinaosuusvertailussa kilpailijoidensa varjoon. Se ei ole onnistunut vakiinnuttamaan asemaansa SQL-pohjaisten tietokantaratkaisujen korvaajana, ja jää tietokantojen markkinaosuusvertailussa näiden alapuolelle (kuvio 5).

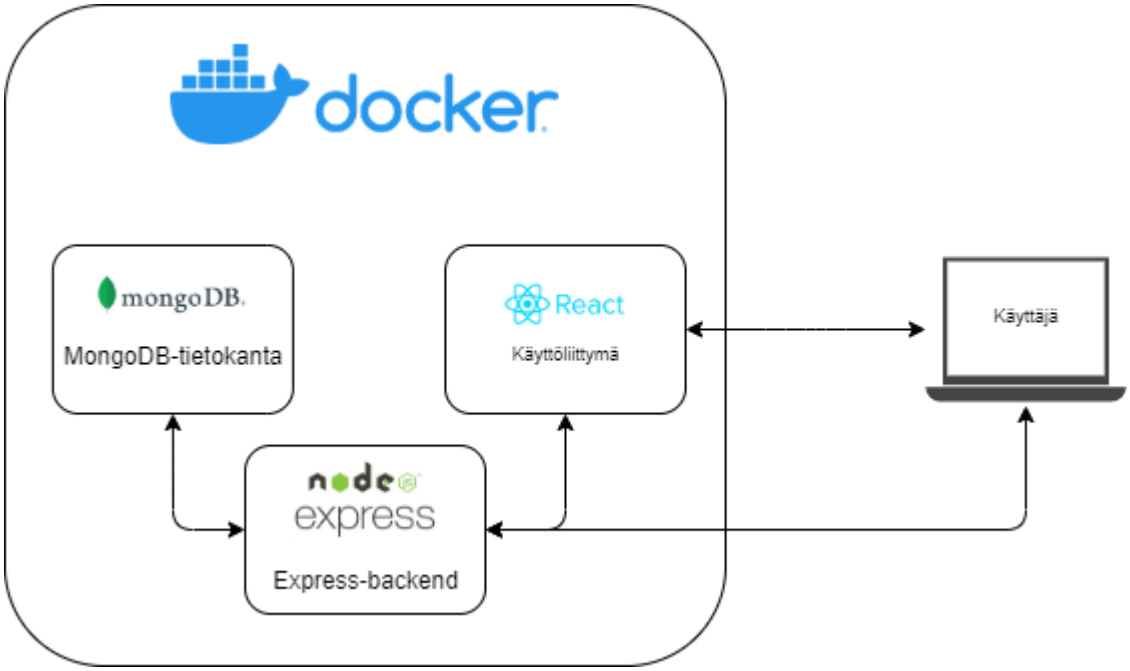


Kuvio 5. MongoDB:n markkinaosuus (Stack Overflow, 2020e)

Tämä johtuu osittain sen teknologisista rajoitteista, lähinnä relaatiotietokantojen tuen puutteesta, joka tekee siitä SQL-pohjaisiin ratkaisuihin verrattuna huonomman valinnan useissa laajoissa järjestelmäkokonaisuuksissa, kuten pankkijärjestelmissä ja henkilötietojärjestelmissä. Pienemmissä järjestelmissä, kuten useimmissa verkkosivustoissa ja mobiiliapplikaatioissa, MongoDB taas tarjoaa erinomaisen vaihtoehdon helppokäyttöisen tallennus- ja hakumallinsa ansiosta.

4.2. Toteutunut arkkitehtuuri

Järjestelmän toteutuksessa päädyttiin tämän opinnäytetyöprojektin puitteissa arkkitehtuuriin, jossa käyttöliittymä, palvelinosuus ja tietokanta tarjoillaan kukin omassa Docker-kontissaan (kuvio 6). Käyttöliittymän osalta Docker-kontista haetaan vain staattisia resursseja, ja useimmat käyttöliittymän elementit suoritetaan käyttäjän selaimessa.



Kuvio 6. Järjestelmän toteutunut arkkitehtuuri

Arkkitehtuuri toimii hyvin kehitysympäristön tarpeisiin, ja antaa valmiudet järjestelmän helpolle tuotantoon saattamiselle. Tätä ei opinnäytetyöprojektille asetetun aikataulun sisällä saatu kuitenkaan tehtyä, joten tuotantovalmiuden käsittely jää tällä erää jokseenkin hypoteettiseksi.

4.3. Tavoiteltu arkkitehtuuri ja järjestelmän tuotantoon saattaminen

Vaikka järjestelmää ei tämän opinnäytetyön puitteissa saatukaan vietyä tuotantoon, käsitellään tuotantovalmiuden toteuttamiseksi vaadittavia toimenpiteitä esimerkinomaisesti, aiempiin kokemuksiin nojaten. Järjestelmän tuotantoversioon oli suunniteltu integroitavaksi kolme toteutetusta järjestelmästä puuttuvaa komponenttia: Nginx, Certbot ja pilvipalveluntarjoajalta vuokrattu palvelin.

4.3.1 Nginx

Nginx on ilmainen, avoimen lähdekoodin HTTP-palvelin ja käänteinen proxy, joka tunnetaan suorituskyvystään ja vähäisestä resurssien kulutuksestaan (F5, 2021). Se on laajasti käytetty ratkaisu verkkopalvelimien liikenteenhallintaan, joka tarjoaa käyttäjilleen monipuolisia käyttömahdollisuuksia yksinkertaisella konfiguraatiolla.

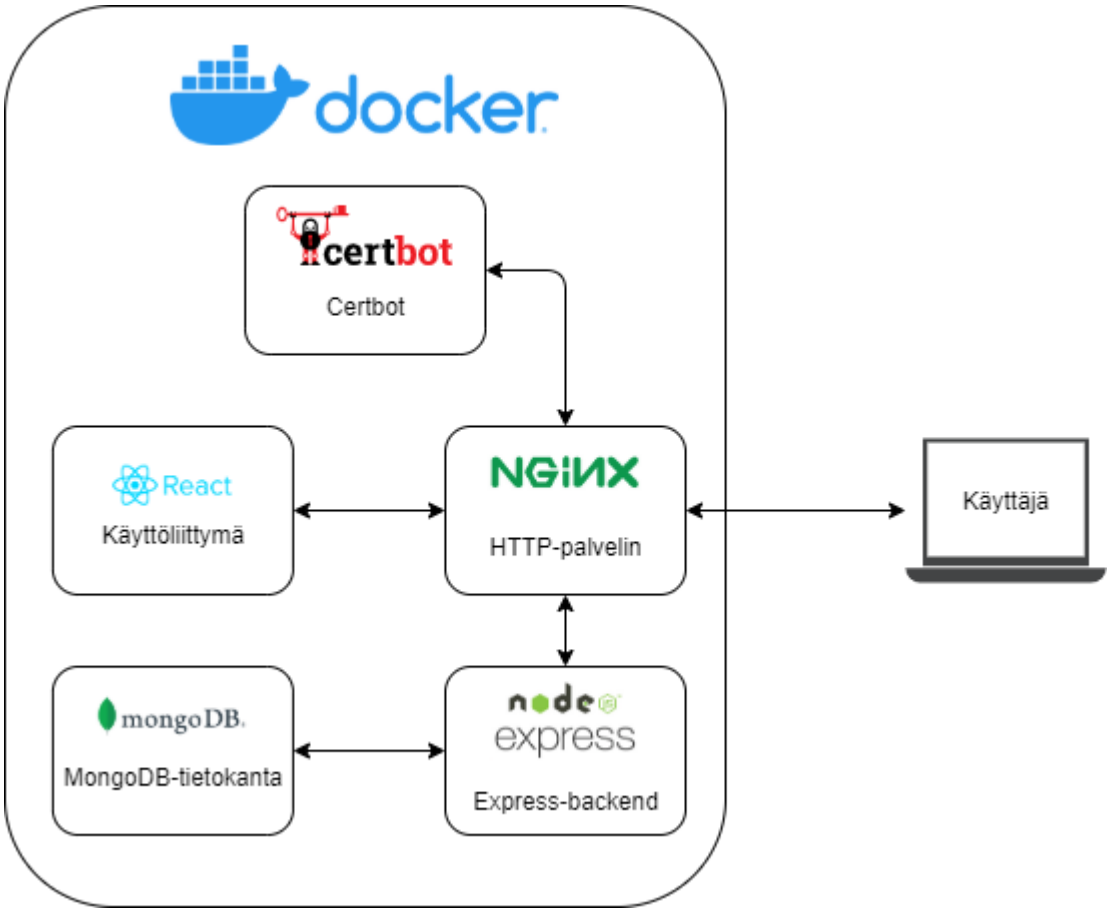
Opinnäytetyöprojektissa Nginx olisi toiminut nimenomaan tässä roolissa, ja olisi yhdessä Certbotin tarjoamien SSL-sertifikaattien kanssa mahdollistanut salatun HTTPS-liikenteen sivustolle, joka on kriittinen ominaisuus järjestelmälle joka tarjoaa salasanamuotoisen vaihtoehdon tunnistautumiseen. Salaamattomaan HTTP-liikenteeseen ei voida luottaa siirrettäessä käyttäjien käyttäjätunnuksia ja salasanoja internetin yli. Nginx on saatavilla valmiina Docker-konttina Docker Hubista.

4.3.2 Certbot

Certbot on avoimen lähdekoodin ohjelmistotuote, joka automatisoi Let’s Encryptin tarjoamien SSL-varmenteiden käyttöönoton ja uusimisen (Electronic Frontier Foundation, 2021). Sen kehittäjä Electronic Frontier Foundation ajaa digitaalisen yksityisyyden, sananvapauden ja innovaation arvoja, ja haluaa mahdollistaa turvallisen tietoliikenteen helpon käyttöönoton verkkosivujen ylläpitäjille. Nginxin tapaan se on saatavilla valmiina konttina Docker Hubista.

4.4. Tuotantovalmiin järjestelmän arkkitehtuuri

Tuotantoympäristössä järjestelmän suunniteltiin noudattavan kuvion 7 arkkitehtuuria. Järjestelmän rakenne eroaa toteutetusta järjestelmästä (kuvio 6) siten, että kaikki Dockerin sisään kulkeva liikenne kulkisi tuotantojärjestelmässä Nginxin kautta. Myös Certbot on tuotantojärjestelmässä uutta, mutta sen lisäys ei vaikuta käyttöliittymän, tietokannan ja palvelinosan väliseen kommunikaatioon.



Kuvio 7. Tuotantovalmiin järjestelmän suunniteltu arkkitehtuuri

Käyttöliittymän ja palvelinosuuden Docker-konttien, MongoDB:n, Nginxin ja Certbotin tuotantovalmiiden konfiguraatioiden ja asetusten kirjoittamiseen löytyy ohjeita kunkin palvelun tarjoamasta dokumentaatiosta.

5 KÄYTTÖLIITTYMÄ

Tässä kappaleessa käsitellään opinnäytetyöhön toteutetun käyttöliittymän toteutusta koodiesimerkein. Läpi käydään projektissa sovelletut koodauskäytännöt, React-sovelluksen luominen, käyttöliittymän komponenttirakenne sekä toteutus näkymittäin.

5.1. Koodauskäytännöt

Projektin käyttöliittymäosuuden rakenteen ja koodauskäytäntöjen muodostamiseen käytettiin käytäntöjä, jotka selkeyttivät projektin kehitystyötä ja hallintaa. Monet mainituista työkaluista ja käytännöistä ovat ohjelmistokehittäjien globaalisti käyttämiä ja kuuluvat nykypäivän ohjelmistokehittäjältä odotettuihin perustaitoihin.

5.1.1 Git

Git on ilmainen avoimen lähdekoodin versionhallintajärjestelmä, joka on nykyisin käytössä lähes kaikissa ohjelmistoalalla toimivissa organisaatioissa (Git, 2021). Gitin laaja suosio perustuu projektien helppoon haarauttamiseen, joka tekee uusien ominaisuuksien kehittämisestä ja julkaisusta sekä kehitys- että tuotantoympäristöihin yksinkertaista ja vaivatonta. Eräänä suosittuna Gitin käyttömallina mainittakoon Gitflow Workflow (Bitbucket, 2021).

Tässä projektissa Git:iä käytetään lähinnä keskitettynä hallintapaikkana, josta koodin viimeisin versio on saatavilla. Kehitystyötä tehtiin useilla eri laitteilla selainyhteensopivuuden varmistamiseksi, joten Gitin kaltainen keskitetty koodinhallintaratkaisu oli tarpeen. Gitin avulla koodi voidaan välittää useisiin eri kolmannen osapuolen hallintajärjestelmiin, esimerkiksi GitHubiin, GitLabiin tai Bitbuckettiin. Tässä projektissa käytössä oli GitHub.

5.1.2 ESlint

ESlint on työkalu, joka havaitsee malleja ja syntaksivirheitä kehittäjän kirjoittamasta JavaScript-koodista (ESlint, 2021). Sen tarjoama mahdollisuus havaita ja korjata sekä syntaksi- että formatointivirheitä on varsinkin suuremmissa projekteissa ensiarvoisen tärkeää. Koodin lukemisesta ja hallinnasta tulee huomattavasti helpompaa, kun koodi on tasalaatuista ja samalla tavalla formatoitua eri tiedostojen ja kirjoittajien välillä.

ESlint:in on saatavilla valmiita konfiguraatioita, jotka asettavat kirjoitettavan koodin syntaksille, sisällölle ja formatoinnille tiettyjä sääntöjä ja rajoitteita. Tässä projektissa käytettiin hieman mukautettua versiota eslint-config-airbnb -konfiguraatiosta, joka on erittäin suosittu. Konfiguraatio määritetään .eslintrc.js -tiedostossa, josta tässä projektissa käytetyn koodieditori Visual Studio Coden ESlint-lisäosa lukee käytössä olevan konfiguraation. Tämän perusteella koodieditori antaa reaaliaikaista palautetta koodissa esiintyvistä mahdollisista virheistä.

5.2. Create React App

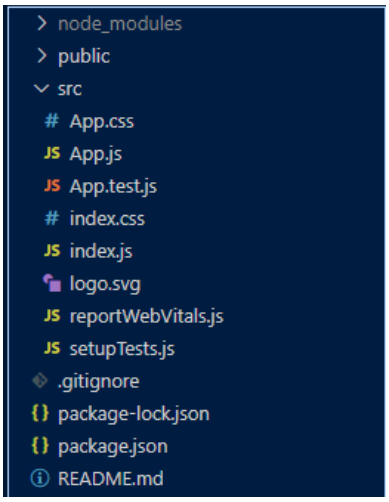
Työn käyttöliittymä luotiin Create React App-moduulilla. Create React App luo kehittäjälle valmiin projektipohjan React-sovelluksen kehittämiseen, ja on yksi suosituimmista tavoista Reactilla toteutettavien SPA:en (single-page application) luomiseen (Faceboook, 2021a). Kirjoitushetkellä Create React App-moduulin käyttö vaatii Node.js -version 10.16 ja Node.js:n mukana tulevan npm-paketinhallintatyökalun version 5.6.

Create React App ajetaan kuvassa 1 esiteltävillä komennoilla. Komennot luovat uuden my-app -nimisen kansion, asentavat React-sovellukseen tarvittavat tiedostot sekä käynnistävät lokaalin kehityspalvelimen, joka tuo sovelluksen kehittäjän näkyviin osoitteeseen http://localhost:3000.

```
npx create-react-app my-app
cd my-app
npm start
```

Kuva 1. Create React Appin ajamisen komennot (Facebook, 2021a)

Komento luo my-app-kansioon kuvan 2 mukaisen kansiorakenteen. Suurin osa kehittäjän työstä keskittyy src-kansiossa sijaitseviin, ja sinne luotaviin tiedostoihin.

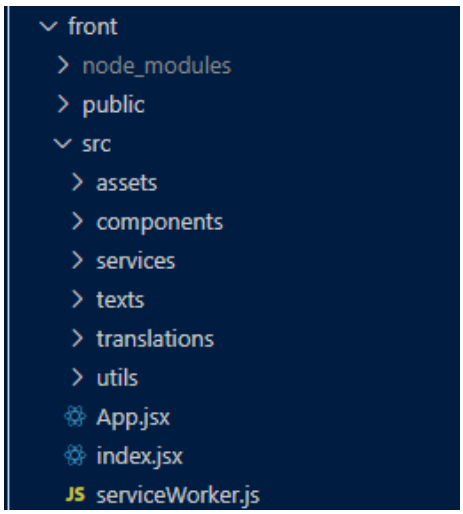


Kuva 2. Create React Appin luoma kansiorakenne

Kehittäjän näkökulmasta huomionarvoisin komennon luomista tiedostoista on App.js. Tähän tiedostoon kehittäjä voi alkaa toteuttaa omia komponenttejaan ja kehittää omia toiminnallisuuksiaan.

5.3. Projektityön käyttöliittymä

Projektia kehitettäessä Create React Appin tarjoamaa kansiorakennetta mukautettiin valittujen toimintatapojen sekä projektin asettamien vaatimusten mukaisiksi. Komponenteille, kuville, palveluille kuten API-komponenteille, teksteille, käännöksille sekä apumuuttujille ja -funktioille tehtiin omat kansionsa (kuva 3). Valittuun kansiorakenteeseen päädyttiin aiemmin toteutetuissa projekteissa hyviksi havaittujen toimintatapojen perusteella.



Kuva 3. Käyttöliittymän kansiorakenne

Projektin komponenttirakenne selviää parhaiten App.jsx-tiedostoa tarkastelemalla (kuva 4). Sovelluksen yläpalkki renderöidään näkymästä huolimatta, ja itse näkymät renderöidään kukin oman reittinsä alla. Reititys on toteutettu React Routerilla. Se on React Trainingin kehittämä työkalu muun muassa SPA-sovellusten reitittämiseksi, johon sitä tässä projektissa hyödynnetään (React Training, 2021). SPA-sovellukset eivät luonnostaan tue toiminnallisuutta, jossa käyttäjä voi URL-osoitetta muokkaamalla navigoida tiettyyn näkymään sovelluksen sisällä. Tätä toiminnallisuutta kuitenkin oletetaan käyttäjien puolelta, ja sen toteuttaminen mahdollistaa esimerkiksi tietyn näkymän tallentamisen selaimen kirjanmerkkeihin.

```
<div>
  <Header />
  <Route path="/albums" component={Albums} />
  <Route path="/calendar" component={Calendar} />
  <Route path="/contact" component={Contact} />
  <Route path="/login">
    <Login
      authenticated={authenticated}
      setAuthenticated={setAuthenticated}
    />
  </Route>
  <Route path="/addEvent">
    <AddEvent
      authenticated={authenticated}
      setAuthenticated={setAuthenticated}
    />
  </Route>
  <Route exact path="/" component={LandingPage} />
</div>
```

Kuva 4. Käyttöliittymän komponenttirakenne

Näkymien yksinkertaisuuden ja staattisen luonteen vuoksi komponentteja toteuttaessa ei ilmennyt tarvetta toteuttaa useiden komponenttien käyttämiä alemman tason komponentteja. Dynaamisempaa sisältöä vaativissa tapauksissa on yleistä toteuttaa esimerkiksi jokin sivupalkki omana komponenttinaan, jota useat muut komponentit käyttävät. Tämä on yksi Reactin ja muiden modernien JavaScript-käyttöliittymäkirjastojen tarjoamista ydineduista perinteisiin web-kehitysmenetelmiin verrattuna.

5.3.1 Material-UI

Yhdessä modernien JavaScript-kirjastojen kanssa käytetään usein myös moderneja UI-kirjastoja. Näistä ehkä käytetyimmät ovat Bootstrap ja Material-UI. Tämän projektin UI-kirjastoksi valittiin Material-UI, lähinnä kokeilunhalun vuoksi. Material-UI on lähes kaikissa selaimissa toimiva käyttöliittymien tyylittelyyn tarkoitettu kirjasto, joka tarjoaa valmiita React-komponentteja, kuten taulukoita ja alasvetovalikoita kehittäjien käyttöön (Material-UI, 2021).

Tyylien kustomointiin käytettiin Material-UI:n tarjoamaa makeStyles-hookia (kuva 5). Hookin käyttö todettiin kehitystyön edetessä helpoksi, mutta koska tyylit joudutaan määrittämään samassa tiedostossa niitä käyttävän koodin kanssa, tulee tiedostojen lukemisesta tarpeettoman hankalaa. Tulevissa projekteissa tyylit toteutetaan menetelmällä, joka sallii tyylien määrittelyn eriyttämisen omiin tiedostoihinsa.

```
const styles = makeStyles({
  container: {
    marginTop: '60px',
    marginBottom: '60px',
    width: '100%',
    display: 'flex',
    flexDirection: 'column',
  },
  item: {
    marginTop: '30px',
    width: '100%',
  },
  field: {
    width: '100%',
  },
  label: {
    marginBottom: '10px',
  },
  login: {
    marginTop: '30px',
  },
});
```

Kuva 5. Tyylien määrittely makeStyles-hookilla

5.3.2 React Hooks

Hookeja käytettiin myös itse Reactin puolella. Hookit ovat korvaava menetelmä luokkapohjaisten React-komponenttien elinkaarimetodeille ja tilanhallinnalle, jotka aiemmin vaativat kehyksekseen luokkapohjaisen komponentin (Facebook, 2021b). Hookeja käytettäessä komponentit voidaan kirjottaa modernimmalla funktiosyntaksilla, joka tuo Reactin kirjoittamisen paremmin linjaan modernin JavaScriptin kanssa.

Projektissa eniten käytetyt hookit ovat useState ja useEffect, joiden käyttöä esitellään kuvan 6 koodissa. Rivin 5 syntaksilla alustetaan muuttuja users tyhjäksi listaksi, sekä määritetään metodi setUsers, jolla users-muuttujan arvoa voidaan muuttaa rivillä 10 esitellyllä syntaksilla. Tässä esimerkissä useEffect-hookia puolestaan käytetään datan hakuun, joka on hyvin yleinen käyttöskenaario. Hookin riippuvuudet on rivillä 15 jätetty tyhjäksi, joten useEffectin sisältö suoritetaan vain kerran komponentin kiinnittyessä. Se korvaa siis Reactin elinkaarimetodeista tässä tapauksessa ComponentWillMountin. Jos useEffect palauttaisi funktion, suoritettaisiin funktion sisältö komponentin irrottuessa. Tällöin yhdellä hookilla olisi korvattu sekä ComponentWillMount että ComponentWillUnmount.

```
1  import React, { useState, useEffect } from 'react';
2  import axios from 'axios';
3
4  const App = () => {
5    const [users, setUsers] = useState([]);
6
7    useEffect(() => {
8      axios.get('https://randomuser.me/api/')
9      .then(({ data: { results } }) => {
10       setUsers(results);
11     })
12     .catch(err => {
13       console.error(err);
14     })
15   }, [])
16
17   return (
18     <div>
19       {users.map(({ name: { title, first, last }}) => (
20         <div>
21           <span>{title} {first} {last}</span>
22         </div>
23       ))}
24     </div>
25   );
26 }
27
28 export default App;
```

Kuva 6. Esimerkki hookien käytöstä

5.3.3 React-i18next

Sivuille asetetun kaksikielisyysvaatimuksen täyttämiseen käytettiin react-i18next -kirjastoa. Se on Reactille räätälöity versio i18next-lokalisaatiotyökalusta, joka tekee verkkosivujen ja mobiiliapplikaatioiden tarjoamisesta usealla kielellä helppoa (React-i18next, 2021). Se on optimoitu palvelimella renderöityihin sovelluksiin, mutta toimii tämän projektin tarkoituksiin erinomaisesti.

Projektin kansiorakenteessa käännökset sijoitettiin translations-kansioon. Kansio jaettiin alakansioihin ”en” ja ”fi” joihin kumpaankin sijoitettiin translations.json-nimiset tiedostot. Näiden lisäksi kansioon tehtiin common.json-tiedosto, joka säilöö molemmille kielille yhteisiä tekstejä, kuten erisnimiä ja vuosilukuja. Käännöstiedostoissa tekstit tallennettiin JSON-objektiin avain-arvo-pareina. Avaimet ovat molemmissa tiedostoissa samat arvon muuttuessa kielestä toiseen.

Koodissa käännöksiä käytettiin react-i18nextin tarjoaman Trans-komponentin avulla. Komponentille annettiin ”i18nKey”-niminen parametri, jonka arvoksi annettiin halutun käännöstekstin avain (kuva 7).

```
<Typography component="h6">
  <Trans i18nKey="home" />
</Typography>
```

Kuva 7. Trans-komponentin käyttö

Kielen vaihtaminen toteutettiin Header-komponenttiin, jossa kielen vaihdon suorittava nappi sijaitsee. Kielen vaihtamisesta huolehtii funktio joka tunnistaa käyttöliittymän kielen i18nextin ”changeLanguage”-funktion avulla, ja vaihtaa käyttöliittymän kielen (kuva 8).

```
const changeLanguage = () => {
  const { languages: { en, fi } } = Constants;

  i18n.changeLanguage(isFinnish(i18n.language) ? en : fi);
};
```

Kuva 8. Käyttöliittymän kielen vaihtava funktio

5.4. Käyttöliittymän näkymät

Käyttöliittymä koostuu neljästä käyttäjälle tarkoitetusta näkymästä ja kahdesta vain ylläpitäjille tarkoitetusta näkymästä. Näkymien sisältöä sekä nostoja niiden koodista esitellään näkymäkohtaisesti. Kuten on aiemmin mainittu, havainnekuvat näkymien ulkoasusta eivät edusta tuotantoon päätyvää versiota, vaan toimivat esimerkkeinä.

5.4.1 Etusivu

Etusivulle toteutettiin koko selainikkunan kokoinen landing page-tyyppinen, visuaalisesti näyttävä kokonaisuus, jossa suurin rooli annettiin asiakkaan nimelle ja ammatille (kuva 9). Sivuston ensivaikutelman haluttiin olevan jylhä ja arvokas, jonka vuoksi etusivun värimaailma pidettiin minimalistisena.

Kuva 9. Käyttöliittymän etusivu

Sivua alaspäin selatessa käyttäjälle esitetään tietoa asiakkaan koulutuksesta, työhistoriasta sekä kiinnostuksenkohteista niin ammatillisesti kuin vapaa-ajallakin. Etusivun perusteella on tarkoitus saada kattava kuva asiakkaasta ja hänen työstään.

5.4.2 Projektit

Projektit-näkymässä asiakkaan projekteja on esitelty korttien muodossa. Kortit on jaettu horisontaalisesti kahteen osaan, vasemmalla projektia kuvaavaan kuvaan ja oikealla projektin perustietoihin (kuva 10).

Kuva 10. Esimerkki projektikortista

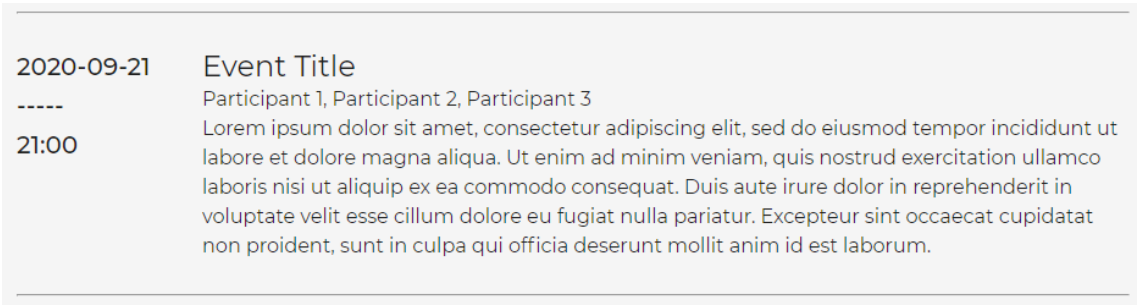
Kortit toteutettiin hyödyntäen Material-UI:n Card-komponenttia, joka koostuu CardMedia- ja CardContent-alakomponenteista (kuva 11). Korttiin haluttu kuva sijoitettiin CardMedia-osioon ja teksti puolestaan CardContent-osioon. Material-UI:n avulla korttimuotoisten esitystapojen toteuttaminen oli vaivatonta.

```
<Card className={classes.card}>
  <CardMedia
    component="img"
    className={classes.cover}
    image={img}
    title="img_title"
  />
  <div className={classes.details}>
    <CardContent className={classes.content}>
      <Typography variant="h5" className={classes.text}>
        <Trans i18nKey="cardTitle" />
      </Typography>
      <Typography variant="body1" className={classes.text}>
        <Trans i18nKey="cardParticipants" />
      </Typography>
      <Typography variant="body1" className={classes.text}>
        <Trans i18nKey="cardDate" />
      </Typography>
    </CardContent>
  </div>
</Card>
```

Kuva 11. Esimerkki korttikomponentin koodista

5.4.3 Kalenteri

Kalenterinäkymässä esitellään asiakkaan menneitä ja tulevia tapahtumia päivämäärineen. Näkymän on tarkoitus toimia ensisijaisena tiedotuskanavana sivujen vierailijoille. Kalenteritapahtumasta esitellään otsikko, mahdolliset muut osallistujat, päivämäärä, kellonaika sekä yksityiskohtainen kuvaus tapahtuman sisällöstä (kuva 12).



Kuva 12. Esimerkki tapahtumasta

Tapahtumat haetaan sivulle navigoitaessa tietokannasta. Datan haku toteutetaan aiemmin kappaleessa 5.3.2 annetun esimerkin tapaan useEffect-hookissa, jonka sisältö suoritetaan kerran komponentin kiinnittyessä (kuva 13). Tapahtumat tallennetaan events-taulukkoon, jossa kukin tapahtuma on oma objektinsa.

```
const [events, setEvents] = useState([]);

useEffect(() => {
  API.getEvents()
    .then(({ data }) => {
      setEvents(data);
    })
    .catch(err => {
      console.error(err);
    });
}, []);
```

Kuva 13. Datan haku kalenterikomponentissa

Kalenteritapahtumat populoidaan käyttöliittymään dynaamisesti (kuva 14). Dynaamisessa populoinnissa hyödynnetään JavaScriptin Array.map -funktiota, josta palautetaan iteratiivisesti kunkin tapahtuman tiedoilla erillinen Material-UI:n

Grid-komponenteista muodostettu elementti. Elementit populoidaan Array.map -funktiota ympäröivään sarakkeeseen, johon myös käytettiin Grid-komponenttia.

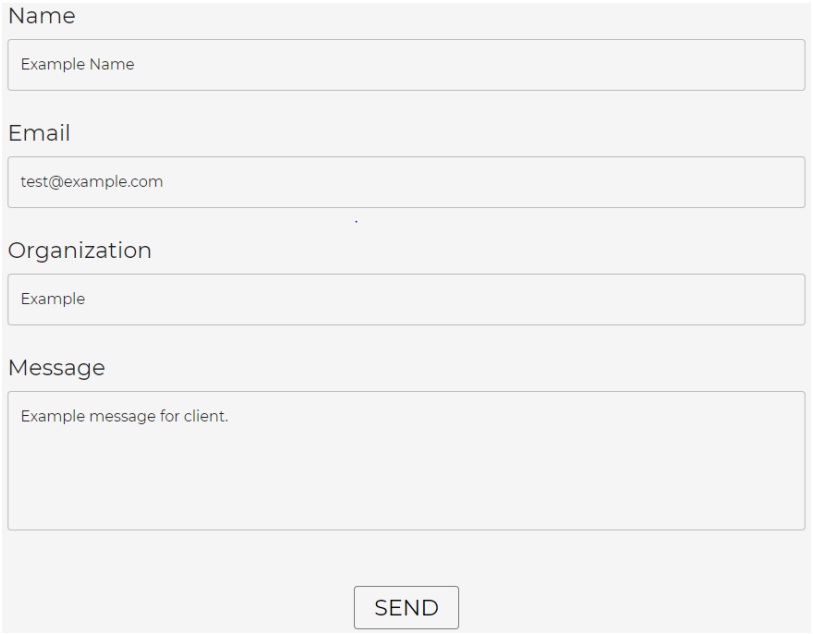
```
<Grid container direction="column" className={classes.grid}>
  {events.map(({ _id, date, time, title, participants, description }) => (
    <Grid item className={classes.row} key={_id}>
      <hr />
      <Grid container direction="row">
        <Grid item xs={2} className={classes.item}>
          <Typography variant="h6">
            {date}
          </Typography>
          <Typography variant="h6">
            -----
          </Typography>
          <Typography variant="h6">
            {time}
          </Typography>
        </Grid>
        <Grid item xs={10} className={classes.item}>
          <Typography variant="h5">
            {title}
          </Typography>
          <Typography variant="body1">
            {participants}
          </Typography>
          <Typography variant="body1">
            {description}
          </Typography>
        </Grid>
      </Grid>
    </Grid>
  ))}
</Grid>
```

Kuva 14. Kalenteritapahtumien populointi dynaamisesti

Kalenteritapahtumien dynaaminen populointi on järkevää, sillä tapahtumien määrä ja sisältö muuttuvat ajan myötä. Dynaaminen populointi myös vähentää kirjoitettavan koodin määrää huomattavasti verrattuna yksitellen kirjoitettuihin tapahtumiin, joiden koodi olisi muuttujia lukuun ottamatta täsmälleen sama.

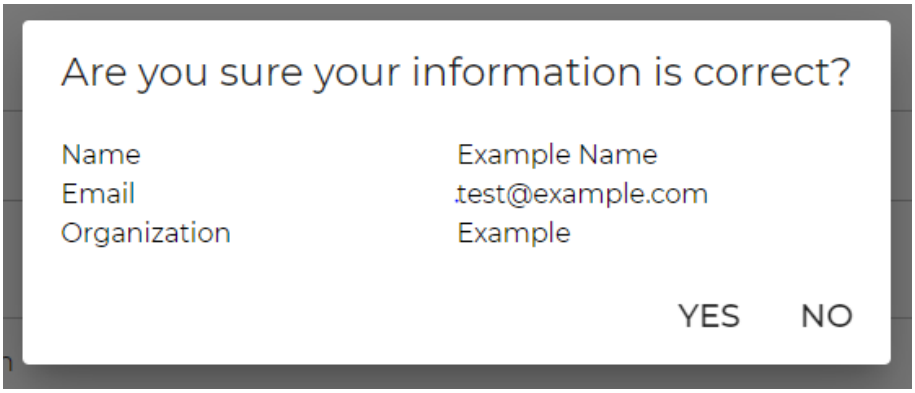
5.4.4 Yhteydenotto

Yhteydenottonäkymä tarjoaa kävijöille lomakemuotoisen yhteydenottomahdollisuuden asiakkaaseen (kuva 15). Lomakkeessa vaadittavat tiedot ovat yhteydenottajan nimi, sähköpostiosoite ja edustama organisaatio. Näiden lisäksi on mahdollista lähettää viesti asiakkaalle.



Kuva 15. Yhteydenottolomake

Lomaketta lähetettäessä käyttöliittymä tarjoaa dialogin, jossa yhteydenottajan on mahdollista tarkistaa tietojensa oikeellisuus (kuva 16). Jos dialogista valitaan ei-vaihtoehto, palautetaan kävijä korjaamaan tietojaan lomakkeeseen. Jos taas valitaan kyllä-vaihtoehto, lähetetään asiakkaalle sähköposti lomakkeen tiedoilla.



Kuva 16. Dialogi lähetettävien tietojen varmistamiseksi

Lomakkeen toteutukseen hyödynnettiin Material-UI:n tarjoama TextField-komponenttia (kuva 17). Lomakkeen kentät ovat saavutettavia, ja niihin voidaan navigoida ilman hiiren käyttöä.

```
<span className={classes.item}>
  <Typography variant="h5" className={classes.label}>
    <Trans i18nKey="formName" />
  </Typography>
  <TextField
    required
    id="name"
    variant="outlined"
    color="secondary"
    className={classes.field}
    value={name}
    onChange={handleNameChanged}
  />
</span>
```

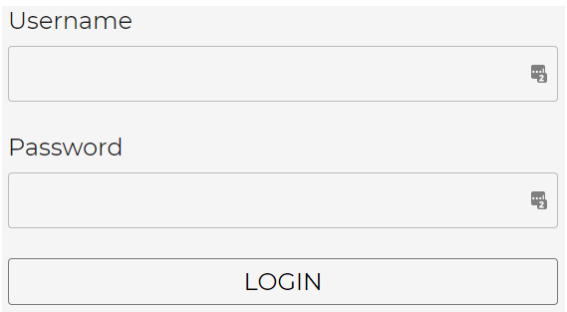
Kuva 17. Lomakkeen nimikentän toteutus

5.5. Ylläpitäjälle tarkoitetut näkymät

Käyttöliittymä sisältää kaksi näkymää, jotka on tarkoitettu vain ylläpitäjien käytettäväksi. Näkymiin ei tarjota käyttöliittymässä näkyvää navigaatiomahdollisuutta, vaan niihin tulee navigoida URL-hakukenttää käyttäen. Näkymiin navigointi päätettiin toteuttaa näin, koska reittien olemassaoloa ei haluta näyttää sivuilla vieraileville käyttäjille.

5.5.1 Kirjautumisnäkymä

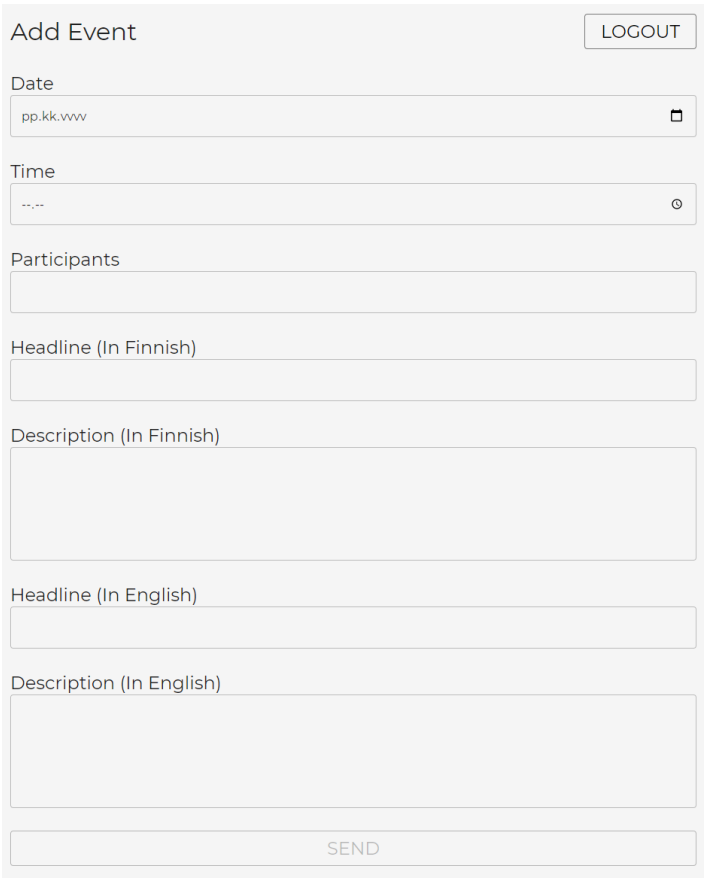
Kirjautumisnäkymässä käyttäjälle tarjotaan kentät käyttäjätunnukselle sekä salasanalle (kuva 18). Käyttäjän syöttäessä oikeat kirjautumistiedot, ohjataan hänet suoraan kappaleessa 5.5.2 esiteltävään Lisää tapahtuma -näkymään. Sovelluksen kirjautumismekanismia käsitellään tarkemmin kappaleessa 6.3.2.



Kuva 18. Kirjautumisnäkymä

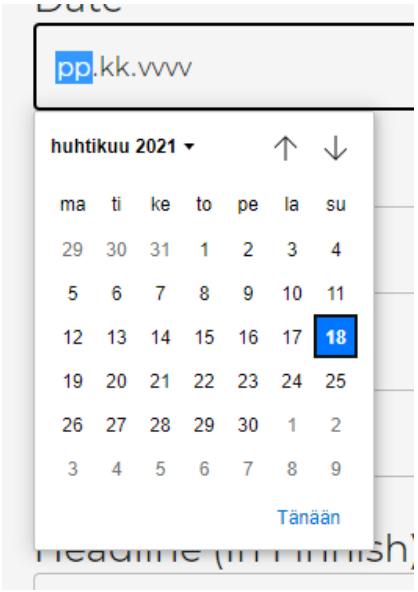
5.5.2 Lisää tapahtuma -näkymä

Lisää tapahtuma -näkymä on tarkoitettu vain sivuston ylläpitäjille. Näkymän kautta tietokantaan voidaan tallentaa uusia tapahtumia, jotka tulevat näkyviin kappaleessa 5.4.3 esiteltyyn kalenterinäkymään. Tapahtuman lisäämiseen vaaditaan kaikkien näkymän kenttien täyttäminen (kuva 19). Lisäksi tapahtuman nimi ja kuvaus on täytettävä sekä suomeksi että englanniksi, jotta kalenterinäkymän kaksikielisyysvaatimus toteutuu.



Kuva 19. Lisää tapahtuma -näkymä

Lomakkeen toteutukseen käytettiin pääosin samoja metodeja kuin kappaleessa 5.4.4 esitellyn yhteydenottolomakkeen toteutukseen. Päivämäärä- ja kellonaikakenttien toteuttamiseen päätettiin kuitenkin käyttää Material-UI:n tarjoamia valitsimia pelkkien tekstikenttien sijaan (kuva 20).



Kuva 20. Päivämääräkenttä

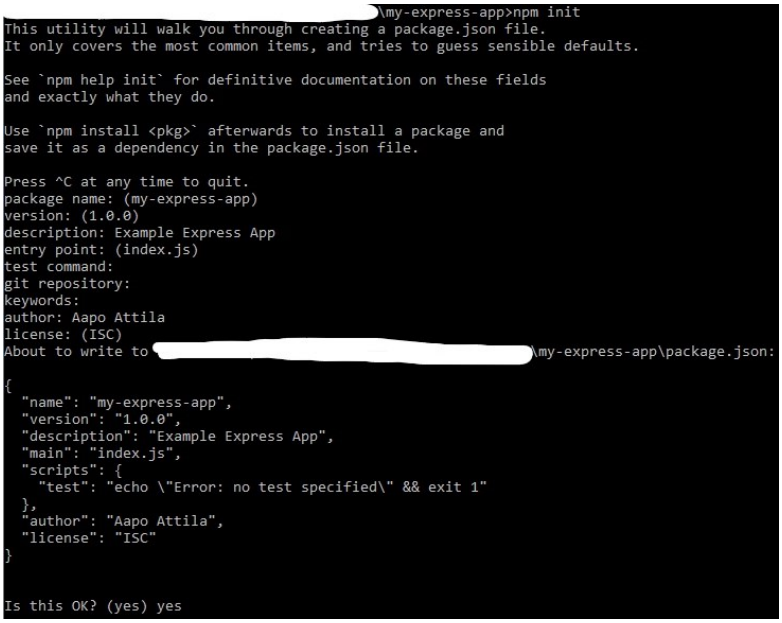
6 PALVELINOSUUS JA TIETOKANTA

Opinnäytetyön palvelinosuus tarjoaa käyttöliittymälle palveluita, kuten yhteydenottosähköpostien välittämisen sekä datan haun tietokannasta. Palvelinosuuden toteutus pyrittiin pitämään mahdollisimman kevyenä ja yksinkertaisena tulevien toiminnallisten lisäysten tai muutosten helpottamiseksi.

Palvelinosuus koostuu Node.js-ajoympäristöstä, Expressillä toteutetusta web-palvelimesta sekä MongoDB-tietokannasta. Tässä kappaleessa käsitellään näiden komponenttien roolia ja toimintaa sekä yksitellen että yhdessä.

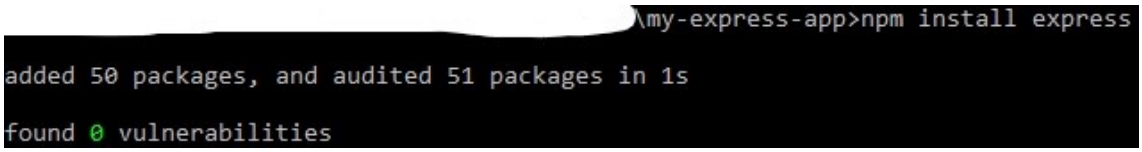
6.1. Yksinkertaisen Express-sovelluksen luominen

Express-sovelluksen luominen edellyttää Node.js:n asentamista. Useimmille käyttäjille suositellaan pitkäaikaisen tuen LTS-versiota, joka kirjoitushetkellä on 14.16.1 (OpenJS Foundation, 2021b). Esimerkkisovelluksen ohjeet on muotoiltu Expressin Hello World Examplen mukaan (OpenJS Foundation, 2021c). Kun Node.js on asennettu, luodaan sovellukselle uusi kansio ”my-express-app”. Kansiossa ajetaan komento ”npm init”, joka luo projektille ”package.json”-tiedoston (kuva 21).



Kuva 21. "npm init"-komennon ajaminen

Projektin metadataa ja riippuvuuksia hallitsevan ”package.json”-tiedoston luomisen jälkeen voidaan projektiin asentaa Express. Asennus tapahtuu npm-paketinhallintatyökalun avulla komennolla ”npm install express”, joka ajetaan ”npm init”-komennon tapaan projektikansiossa (kuva 22).



Kuva 22. "npm install express" komennon ajaminen

Tämän jälkeen kansioon voidaan luoda ”app.js”-niminen tiedosto, johon sijoitetaan kuvassa 23 esitelty koodi. Opinnäytetyön ”app.js”-tiedostossa Express-sovellus luodaan samaa metodia käyttäen.

```
1  const express = require('express')
2  const app = express()
3  const port = 3000
4
5  app.get('/', (req, res) => {
6    res.send('Hello World!')
7  })
8
9  app.listen(port, () => {
10   console.log(`Example app listening at http://localhost:${port}`)
11 })
```

Kuva 23. Yksinkertainen Express-sovellus (OpenJS Foundation, 2021c)

Koodi ajetaan projektikansiossa komennolla ”node app.js”. Komento käynnistää koodissa määritetyn web-palvelimen, joka kuuntelee uusia yhteyksiä porttiin 3000. Navigoitaessa selaimella osoitteeseen http://localhost:3000, palauttaa palvelin selaimeen viestin ”Hello World!”.

6.2. Koodauskäytännöt

Projektin palvelinosuus noudattaa kappaleissa 5.1.1 ja 5.1.2 esiteltyjen käytäntöjen lisäksi muutamia huomionarvoisia toimintatapoja, joiden käyttö parantaa sovelluksen turvallisuutta sekä kehittäjän työnkulkua. Merkittävimpinä näistä mainitaan ”.env”-tiedostojen käyttö.

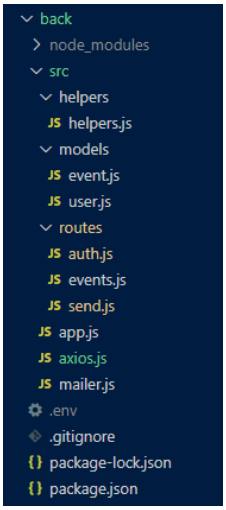
6.2.1 Dotenv

Dotenv on ulkoisista riippuvuuksista vapaa moduuli, joka mahdollistaa ympäristömuuttujien helpon käytön npm-paketinhallintatyökalua käyttävissä projekteissa (Dotenv, 2021). Se mahdollistaa konfiguraatioiden tallentamisen erilleen ohjelmakoodista ajoympäristöön tallennettavassa ”.env”-tiedostossa. Dotenv-moduulin avulla tallennettuja muuttujia voidaan käsitellä ikään kuin globaaleina muuttujina käyttäen ”process.env.MUUTTUJAN_NIMI”-syntaksia.

Dotenv-moduulin käyttö on järkevää esimerkiksi tapauksissa, joissa ohjelmakoodi ottaa yhteyttä tunnistautumista vaativiin ulkopuolisiin järjestelmiin. Esimerkiksi tietokantojen käyttäjätunnukset ja salasanat ovat yleisiä ”.env”-tiedostoon tallennettavia muuttujia. Muuttujien tallentaminen keskitetysti myös helpottaa niiden muuttamista, koska muutos tarvitsee tehdä vain yhteen paikkaan useamman sijaan.

6.3. Express

Projektin palvelinosuus noudattaa kuvassa 24 esiteltyä kansiorakennetta. Kehitystyön kannalta merkittävät tiedostot sijaitsevat ”src”-kansiossa, joka on jaettu alakansioihin tarpeen mukaan. Sovelluksen toiminnan kannalta olennaisimmat tiedostot löytyvät ”routes”-kansiosta, joihin on määritetty Express-sovelluksen käyttämät reitit sekä reiteille saatavissa olevat metodit.



Kuva 24. Palvelinosuuden kansiorakenne

6.3.1 Reititys

Reittejä voidaan määrittää muutamalla tavalla, joista yleisimmät ovat kutsumalla Express applikaatio-objektin HTTP-pyyntötyyppejä vastaavia metodeja, ja käyttämällä Expressin Router-luokkaa (OpenJS Foundation, 2021d). Tässä projektissa käytetään jälkimmäisenä mainittua metodia sen mukanaan tuomien työnkulullisten etujen takia. Router-luokan käyttö mahdollistaa reittien jakamisen tiedostotasolla omiin loogisiin kokonaisuuksiinsa, joka helpottaa reittien hallintaa ja apufunktioiden käyttöä reittien yhteydessä. Esimerkkinä apufunktiosta voidaan mainita funktio, joka tarkastaa että käyttäjä on tunnistautunut ennen reitin suorittamista.

Esimerkkinä Router-luokkaa käyttävästä reitistä tarkastellaan ”send.js”-tiedostoa, jossa on määritetty kappaleessa 5.4.4 esitellyn lomakkeen tietojen lähettäminen asiakkaalle (kuva 25). Kun käyttöliittymä kutsuu rajapinnan ”/send”-reittiä HTTP-metodilla POST, poimitaan kutsun ”body”-objektista asiakkaan täyttämät kentät.

```
1  const express = require('express');
2  const { transporter, getMessage } = require('../mailer');
3
4  const router = express.Router();
5
6  router.post('/', ({ body: { name, organization, email, message } }, res) => {
7    res.sendStatus(200);
8    transporter.sendMail(getMessage(name, organization, email, message))
9    .then(() => res.sendStatus(200))
10   .catch(err => {
11     console.log(err);
12     res.sendStatus(500).json({
13       msg: "Internal Server Error"
14     })
15   });
16 });
17
18 module.exports = router;
```

Kuva 25. "send.js"-tiedosto

Router-instanssi otetaan ”app.js”-tiedostossa käyttöön ”app.use”-metodilla, joka avaa tiedostoissa määritetyt reitit Express-sovelluksen käyttöön (kuva 26). Tätä menetelmää käyttämällä reittien hallinnointi, uusien reittien lisääminen sekä sovelluksen kokonaisuuden hahmottaminen helpottuu huomattavasti.

```
54  const sendRouter = require('./routes/send');
55  const authRouter = require('./routes/auth');
56  const eventsRouter = require('./routes/events');
57
58  app.use('/send', sendRouter);
59  app.use('/auth', authRouter);
60  app.use('/events', eventsRouter);
```

Kuva 26. Express Router-instanssien käyttöönotto "app.js"-tiedostossa

6.3.2 Tunnistautuminen

Sovelluksen tunnistautumista hallinnoidaan Express-sovelluksen kautta. Kirjautumiseen käytetään perinteistä käyttäjätunnus-salasana -yhdistelmää, ja se vaaditaan operaatioihin, joita vain sivuston ylläpitäjän on tarkoitus tehdä. Sivujen käyttö vierailijana ei vaadi tunnistautumista.

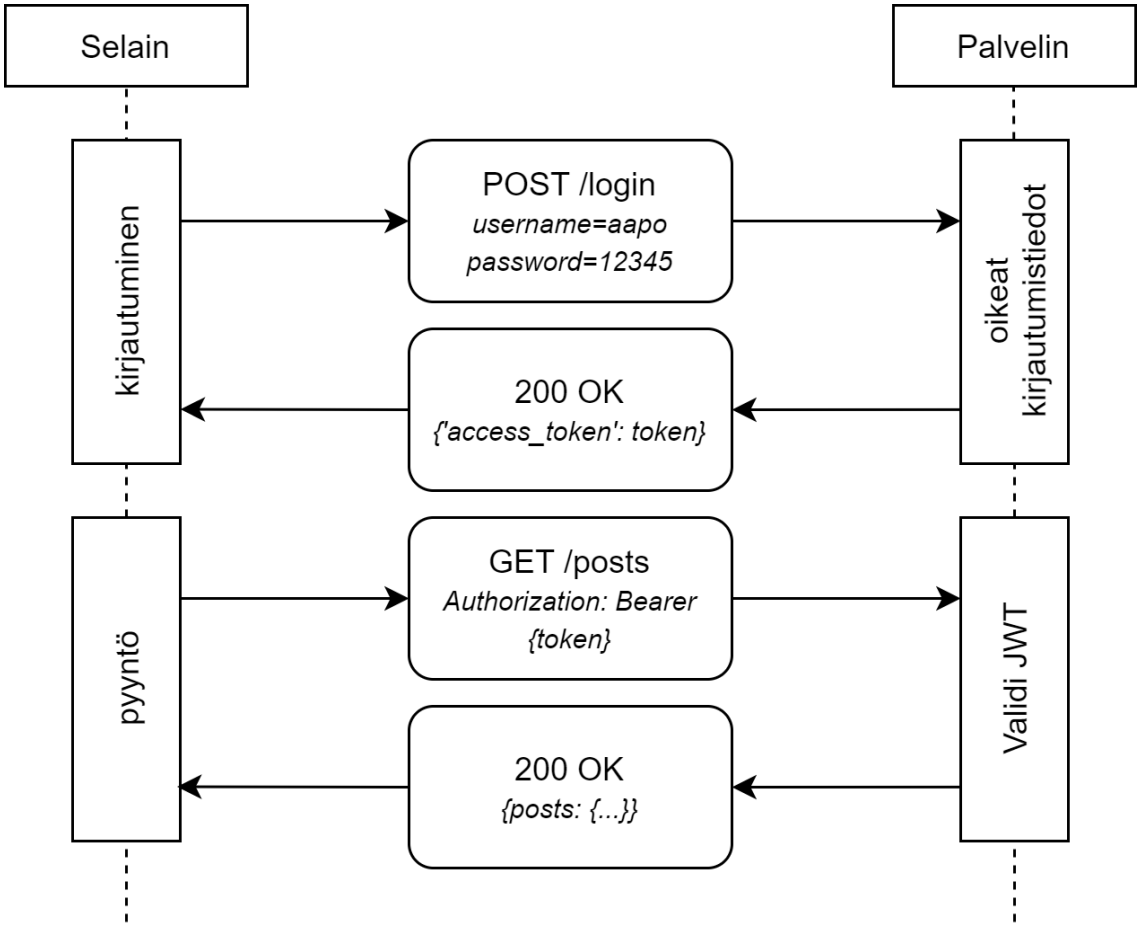
Kirjautumisstatuksen hallinnointiin ja käyttäjän tietojen turvalliseen välittämiseen käytetään JSON Web Tokeneita (JWT) (kuva 27). JSON Web Token on avoimen standardin teknologia, jonka avulla tietoa voidaan siirtää turvallisesti JSON-objekteina (Auth0, 2021). JWT:nä siirrettävään tietoon ja sen alkuperään voidaan luottaa digitaalisen vahvistuksen ansiosta. Tokenit voidaan vahvistaa joko salausavaimen tai julkisesta avaimesta ja salaisesta avaimesta koostuvan avainparin avulla.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Kuva 27. Esimerkki JSON Web Tokenista (Auth0, 2021)

JSON Web Tokeneita käytetään web-sovellusten yhteydessä useimmiten käyttäjien tunnistautumiseen. Käyttäjän kirjautuessa sisään oikealla käyttäjätunnuksella ja salasanalla, palauttaa palvelin käyttäjälle JSON Web Tokenin, joka sisältää käyttäjän tunnistamiseen käytettävää informaatiota. Tämä informaatio on täysin kehittäjän päätettävissä, mutta sen tulee riittää käyttäjän yksilölliseen tunnis-

tamiseen. Token lähetetään tulevien kutsujen mukana palvelimelle, jossa tokeniin säilötyn tiedon oikeellisuus ja itse tokenin voimassaoloaika tarkastetaan. Jos tokeni hyväksytään, palautetaan käyttäjän pyytämät resurssit selaimelle (kuvio 8).



Kuvio 8. JSON Web Tokenin toimintamalli

6.4. MongoDB

Projektissa tietokantaan tallennetaan käyttäjiä ja tapahtumia. Myös käyttöliittymän Projektit-näkymässä esitellyt projektit voitaisiin tallentaa tietokantaan, mutta se jätettiin tässä opinnäytetyöprojektissa tekemättä aikataulullisista syistä. Esiteltävien projektien ei myöskään oleteta muuttuvan usein, joten niiden päivittäminen koodimuutoksilla ei lisää kehittäjän ylläpidollista työmäärää merkittävästi. Uusia tapahtumia sen sijaan ilmenee jatkuvasti, joten tapahtumat päätettiin tallentaa

tietokantaan, ja antaa asiakkaalle mahdollisuus tallentaa uusia tapahtumia itse käyttöliittymän kautta.

Tietokantaa hallinnoivana väliohjelmistona käytetään Mongoose-moduulia, joka on saatavilla npm-paketinhallintatyökalun kautta. Mongoose hallinnoi tietokantaan tallennettavaa ja sieltä haettavaa tietoa mallipohjaisesti (Mongoose, 2021). Tällä tarkoitetaan sitä, että tallennettavan tiedon dokumenttirakenne on mahdollista määrittää kooditasolla käyttäen Mongoosen tarjoamaa Schema-rakentajaa (kuva 28). Schema tallennetaan Mongoose-mallina, jonka ansiosta kaikki samaan malliin pohjautuvat objektit tallennetaan tietokantaan samaa tietorakennetta noudattaen.

Kuvan 28 esimerkissä käyttäjänimi määritetään yksilölliseksi kentäksi, mutta salasanaa ei. Kahdella käyttäjällä voi siis olla keskenään sama salasana. Käyttäjänimien yksilöllisyydellä on merkitystä, koska käyttäjänimeä käytetään yhtenä JWT-tunnisteeseen tallennettavana yksilöivänä tietona.

```
1  const mongoose = require('mongoose');
2  const mongooseUniqueValidator = require('mongoose-unique-validator');
3
4  mongoose.set("useCreateIndex", true);
5
6  const schema = new mongoose.Schema({
7    username: { type: String, required: true, unique: true },
8    password: { type: String, required: true },
9  });
10
11 schema.plugin(mongooseUniqueValidator);
12
13 const User = mongoose.model("User", schema);
14
15 module.exports = {
16   User,
17 };
```

Kuva 28. Käyttäjän tietomallin määrittäminen Mongoosen avulla

Esimerkissä määritettyä User-mallia käytetään myös hakiessa tietoa tietokannasta. Express-palvelimen kirjautumisreitistä ”/auth/login” saadaan käyttäjän käyttäjänimi sekä salasana, ja tietokannasta haetaan käyttäjänimen perusteella

käyttäjää (kuva 29). Haku tapahtuu käyttäen ”Model.findOne”-syntaksia, joka hakee tietokannasta yhden kappaleen halutun mallin objekteja. Koska kantaan tallennettujen käyttäjien käyttäjänimet ovat uniikkeja, ei vain yhden käyttäjän hakeminen tuota tässä tapauksessa ongelmia. Jos käyttäjä löytyy, verrataan käyttäjän antamaa salasanaa kannasta löytyvään salasanaan. Jos salasanat täsmäävät, lähetetään käyttäjälle JSON Web Token, jonka avulla hän voi suorittaa tunnistautumisella suojattuja toiminnallisuuksia.

```
router.post('/login', ({body: {username, password } }, res) => {

  User.findOne({ username: username }, (err, user) => {
    if (err) {
      console.error(err);
      return res.status(500).json({
        msg: "Internal server error"
      });
    }
    if (!user) {
      return res.status(401).json({
        msg: "Invalid login credentials"
      });
    }
    if (getEncryptedString(password) !== user.password) {
      return res.status(401).json({
        msg: "Invalid login credentials"
      });
    }
    setToken(user._id, res);
    res.status(200).json({
      msg: "Login succesful"
    });
  })
})
```

Kuva 29. Kirjautumisesta vastaava funktio Express-sovelluksessa

Kehittäjän näkökulmasta Mongoosen käyttö projektissa oli vaivatonta. MongoDB:n suhteen suurimmat ongelmat syntyvätkin tietomallin muodostamisessa sekä tietokannan tietoturvasta huolehtiessa, kun järjestelmä viedään tuotantoon. Tähän vaiheeseen ei opinnäytetyöprojektin puitteissa päästy, mutta tietokanta tulee tuotannossa suojata salasanalla ja sitä käyttäville henkilöille ja palveluille tulee muodostaa käyttäjäroolit, joilla on vain välttämättömät toimivaltuudet. Esimerkiksi tapauksissa, joissa palvelimen ei tarvitse kirjoittaa kantaan uutta tietoa, on järkevää varustaa käyttäjä vain lukuoikeudella.

7 POHDINTA

Opinnäytetyössä toteutettiin henkilöasiakkaalle verkkosivujärjestelmä, joka sisältää käyttöliittymän sekä palvelimen tietokantoineen. Projekti suunniteltiin sisällöllisesti asiakkaan toiveiden mukaiseksi, ja ulkoasun sekä teknisen toteutuksen suunnittelusta vastasi kehittäjä. Järjestelmä päätettiin toteuttaa MERN-teknologiapaketilla.

Tässä kappaleessa analysoidaan opinnäytetyöprojektin onnistuneisuutta eri mittareilla. Mittareina käytetään projektille asetettujen tavoitteiden saavuttamisen lisäksi teknologiavalintojen tarkoituksenmukaisuutta sekä kokemuksia projektin toteuttamisesta ja etenemisestä.

7.1. Vaatimusten toteutuminen

Asiakkaan projektille asettamat sisällölliset sekä toiminnalliset vaatimukset täytettiin riittävällä tavalla. Toteutettu sivusto tarjoaa käyttäjilleen asiakkaan toivomat tiedot, eli asiakkaan profiilin ja työhistorian, tietoa asiakkaan toteuttamista projekteista sekä informaatiota asiakkaan tulevista tapahtumista. Myös yhteydenottomahdollisuus toteutettiin tavalla, joka ei edellytä asiakkaan yhteystietojen tekemistä julkisiksi.

Sivuille toteutettiin asiakkaan toiveiden mukaisesti saatavuus kahdella kielellä, sekä mahdollisuus lisätä sivuille uusia tapahtumia käyttöliittymän kautta. Nämä toiminnot olivat asiakkaan näkökulmasta välttämättömät, mutta jatkokehityksessä sivujen ylläpidettävyyttä tullaan varmasti parantamaan.

7.2. Teknologiavalinnat

Projektin toteutukseen valittu teknologiapaketti, MERN, palveli projektin tarkoituksia hyvin. Valittujen teknologioiden käyttö oli jo ennen projektin aloitusta tuttua, joten merkittäviä ongelmakohtia ei kehitystyössä tullut vastaan. Yksi projektille

asetetuista tavoitteista oli asiakkaan mahdollisuus tapahtumien itsenäiseen lisäämiseen. Tähän tarkoitukseen valittu lähestymistapa ja teknologiapaketti toimi riittävän hyvin. Jos projektin asiakkaalla olisi enemmän tarpeita sivujen sisällön muokkaamiseen, suositeltaisiin käytettäväksi WordPressin kaltaisia valmiita ratkaisuja.

Tämän opinnäytetyöprojektin asiakas ei kuitenkaan ole teknisesti kyvykäs, joten ylläpitotehtävät olisivat todennäköisesti WordPressiä käytettäessäkin jääneet kehittäjän vastuulle. Tästä näkökulmasta tarkasteltuna MERN-teknologiapaketilla toteutetun järjestelmän jatkokehittäminen ja ylläpitäminen tarjoavat kehittäjälle mahdollisuuden osaamisen kehittämiseen ja ammatilliseen kasvuun, vaikuttamatta asiakkaan työmäärään tai projektin aikataulutukseen merkittävästi.

7.3. Aikataulu

Opinnäytetyöprojektille asetetussa aikataulussa ei pysytty, eikä järjestelmää saatu vietyä projektin puitteissa tuotantoon. Aikataulun pettämisen syistä merkittävimmät lienevät optimistisuus kehitystyön etenemisvauhdin sekä projektille käytettävissä olevan ajan suhteen. Järjestelmän tuotantoon saattaminen asetettujen aikamääreiden sisällä ei kuitenkaan ollut asiakkaalle välttämättömän tärkeää, joten viivästys ei aiheuttanut vahinkoa asiakkaalle.

7.4. Kehittäjän kokemus

Kehittäjän näkökulmasta projektin onnistunein osuus oli sen suunnitteluvaihe. Asiakkaan tarpeet ja toiveet kirjattiin yksityiskohtaisesti, ja lopullisiin suunnittelulähtökohtiin päädyttiin keskustelun kautta. Yksimielisyys siitä, mitkä projektin tavoitteet olivat ja miltä valmis tuote tulisi näyttämään, saavutettiin yllättävän sujuvasti. Vaikka suunnittelu ei onnistunut aikataulutuksen osalta, koettiin suunnittelutyö kokonaisuudessaan onnistuneeksi.

Itse kehitystyö oli myös sujuvaa. Järjestelmä toteutettiin tutuilla ja toimiviksi todetuilla teknologioilla, joiden laaja suosio takasi ajantasaisen ja tarkan dokumentaation helpon saatavuuden.

7.5. Jatkokehitys

Projektista opinnäytetyön puitteissa käsiteltävä osuus päättyy tilanteeseen, jossa järjestelmä vaatii vielä kehitystyötä sekä tuotantoon siirtämisen. Tuotantoympäristöön siirtämisen lisäksi merkittävimmät kehityskohteet löytyvät käyttöliittymän hiomisesta ja päivittämisestä, sekä asiakkaan toimesta suoritettavien ylläpitotehtävien lisäämisestä.

Kehitystyö pyritään saamaan päätökseen kesän 2021 aikana, jonka jälkeen projekti siirretään ylläpitovaiheeseen, jossa aktiivista kehitystyötä ei enää tehdä, vaan projektin parissa toiminta keskittyy ylläpidollisiin tehtäviin. Projektin pariin palataan varmasti tulevaisuudessa, kun käyttöliittymän ulkoasun päivittäminen tulee ajankohtaiseksi tai asiakkaan tarpeet muuttuvat alkuperäisiin suunnittelulähtökohtiin verrattuna.

LÄHTEET

Auth0. 2021. Introduction to JSON Web Tokens. Luettu 18.4.2021. https://jwt.io/introduction

Bitbucket. 2021. Gitflow Workflow. Luettu 17.4.2021. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Docker. 2021. Docker makes development efficient and predictable. Luettu 18.3.2021. https://www.docker.com/

Dotenv. 2021. Luettu 18.4.2021. https://www.npmjs.com/package/dotenv

Electronic Frontier Foundation. 2021. About Certbot. Luettu 18.4.2021. https://certbot.eff.org/about/

ESlint. 2021. Getting Started With ESlint. Luettu 17.4.2021. https://eslint.org/docs/user-guide/getting-started

Facebook. 2021a. Create a New React App. Luettu 17.4.2021. https://reactjs.org/docs/create-a-new-react-app.html

Facebook. 2021b. Introducing Hooks. Luettu 17.4.2021. https://reactjs.org/docs/hooks-intro.html

Facebook. 2021c. React. Luettu 18.4.2021. https://reactjs.org/

F5. 2021. Welcome to the NGINX and NGINX Plus Documentation. Luettu 18.4.2021. https://docs.nginx.com/nginx/

Git. 2021. Luettu 17.4.2021. https://git-scm.com/

Material-UI. 2021. Material-UI. Luettu 17.4.2021. https://material-ui.com/

MongoDB. 2021a. The database for modern applications. Luettu 18.4.2021. https://www.mongodb.com/

MongoDB. 2021b. MERN Stack. Luettu 18.4.2021. https://www.mongodb.com/mern-stack

Mongoose. 2021. Luettu 18.4.2021. https://mongoosejs.com/

Mozilla. 2021. JavaScript. Luettu 18.4.2021. https://developer.mozilla.org/fi/docs/Web/JavaScript

OpenJS Foundation. 2021a. About Node.js. Luettu 18.4.2021. https://nodejs.org/en/about/

OpenJS Foundation. 2021b. Node.js. Luettu 18.4.2021. https://nodejs.org/en/

OpenJS Foundation. 2021c. Hello World Example. Luettu 18.4.2021. https://expressjs.com/en/starter/hello-world.html

OpenJS Foundation. 2021d. Routing. Luettu 18.4.2021. https://expressjs.com/en/guide/routing.html

React-i18next. 2021. Introduction. Luettu 18.4.2021. https://react.i18next.com/

React Training. 2021. React Router. Luettu 17.4.2021. https://reactrouter.com/

Stack Overflow. 2020a. Stack Overflow Developer Survey 2020. Luettu 18.4.2021. https://insights.stackoverflow.com/survey/2020#technology-web-frameworks-professional-developers2

Stack Overflow. 2020b. Stack Overflow Developer Survey 2020. Luettu 18.4.2021. https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-web-frameworks-wanted2

Stack Overflow. 2020c. Stack Overflow Developer Survey 2020. Luettu 18.4.2021. https://insights.stackoverflow.com/survey/2020#technology-other-frameworks-libraries-and-tools-professional-developers3

Stack Overflow. 2020d. Stack Overflow Developer Survey 2020. Luettu 18.4.2021. https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-professional-developers

Stack Overflow. 2020e. Stack Overflow Developer Survey 2020. Luettu 18.3.2021. https://insights.stackoverflow.com/survey/2020#technology-databases-professional-developers4