Sukupuu-websovelluksen kehitys

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tieto- ja viestintätekniikka, insinööri (AMK)

kevät 2023

Joni Tanhuansuu

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka
Tekijä Joni Tanhuansuu
Työn nimi Sukupuu-websovelluksen kehitys
Ohjaaja Petri Kuittinen

Tiivistelmä
Vuosi 2023

Opinnäytetyön tarkoituksena oli kehittää sukupuu-websovellus, johon sukututkimusta tekevä henkilö pystyy keräämään sukututkimustietoa, tarkastelemaan sitä ja hyödyntämään sitä uuden tiedon löytämisessä. Opinnäytetyön idea syntyi todellisesta tarpeesta, kun sukututkimusta tekevällä henkilöllä oli tämän kaltaiselle sovellukselle tarvetta.

Opinnäytetyössä tutustutaan teoriapuolella siihen, miten websovellus on toteutettu, mikä tekniikoita siinä on käytetty ja mitä muita teknologioita siinä olisi voinut käyttää. Lisäksi toteutettua sovellusta käydään läpi hieman tarkemmin ohjelmoinnin kautta sekä esitellään myös sovelluksen käyttöliittymää.

Lopputuloksena syntyi käyttötarpeita vastaava ja käyttövalmis sovellus, jota ylläpidetään pilvipalvelussa. Sovellusta on helppo ylläpitää, sillä siihen liittyvä ylläpito on mahdollisimman automatisoitu ja käytännössä uuden version sovelluksesta saa käyttäjälle puskemalla uudet muutokset GitHubiin. Kehitystyö jatkuu sitä mukaan, kun sovelluksessa ilmenee muutostarpeita. Kokonaisuudessaan opinnäytetyön tuloksena syntyi toimiva full stack -sovellus.

Avainsanat Sukututkimus, websovellus, MERN-pino
Sivut 55 sivua ja liitteitä 0 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology
Author Joni Tanhuansuu
Subject Family tree web application development
Supervisor Petri Kuittinen

Abstract
Year 2023

The purpose of the thesis was to develop a web application for family tree, which allows a person studying their genealogy to collect family research information, view it, and use it to discover new information. The idea for the thesis arose from a real need when a person researching their genealogy had a need for such an application.

The theoretical part of the thesis explores the implementation of the web application and the technologies used for this, as well as presents other technologies that could have been used in the thesis project. In addition, the implemented application is examined more closely through programming, and the application's user interface is also presented.

As a result, a ready-to-use application was created that meets the needs of the user, and it is maintained in a cloud service. The application is easy to maintain as the associated maintenance is as automated as possible, and practically a new version of the application can be pushed to the user by pushing the new changes to GitHub. The development work continues as changes are needed in the application. Overall, a functional full stack application was created due to the thesis.

Keywords Genealogy, web application, MERN stack
Pages 55 pages and appendices 0 pages

Sisällysluettelo

Kuvat, taulukot ja kaavat

1 Johdanto

Tavoitteena on rakentaa sovellus, jota pystyy hyödyntämään sukututkimuksessa. Sovelluksen tarkoituksena on helpottaa esimerkiksi kerätyn tiedon keskittämistä ja tallentamista yhteen paikkaan. Ajatuksena on, että sukututkimusta tehdessä esimerkiksi jo löydetty tieto olisi helpommin nähtävissä kokonaisuutena ja sitä pystyisi hyödyntämään helpommin uuden tiedon löytämiseen. Sukututkimuksessa tietoa etsitään pääsääntöisesti esimerkiksi kirkonkirjoista ja arkistoista, niin tämän sovelluksen tarkoitus on olla se paikka, johon tiedot kerätään ja yhdistetään esimerkiksi Excelin tai paperivihkon sijaan.

Sovelluksessa on aivan välttämätöntä olla mahdollisuus tallentaa ja hakea tietoa esimerkiksi henkilön nimen perusteella sekä mahdollisuus yhdistää tietoja keskenään ja luoda henkilöille esimerkiksi perhetauluja. Perhetauluja on mahdollista halutessaan tulostaa paperille. Kerätyt tiedot on tärkeä saada suojaan ulkopuolisilta tahoilta, joten tästä syystä sovellus tulee toiminaan kirjautumisen takana. Kirjautuminen suojaa esimerkiksi myös siltä, että ulkopuolinen taho ei pääse käsiksi tietokantaan. Käyttäjän ei ole mahdollista rekisteröityä sovellukseen itse lainkaan, vaan uuden käyttäjän luominen tulee olemaan ylläpitäjän hallussa täysin. Ylläpitäjä pystyy esimerkiksi luomaan uuden käyttäjän tekemällä pyynnön sovellukselle Postmanin kautta.

Henkilötietojen osalta sovellukseen pystyy tallentamaan esimerkiksi etunimet, sukunimen, syntymäpaikan ja -ajan, kummit, kastepäivän, kuolinpäivän ja -ajan sekä mahdollisen syyn kuolemalle, hautapaikan ja -ajan, pienoiselämäkerran ja mahdolliset lähteet olemassa olevalle tiedolle. Edellä mainittuja tietoja hyödyntäen sovellukseen pystyy luomaan henkilölle perhetaulun. Perhetaulusta löytyy esimerkiksi tarkasteltavan henkilön lisäksi tämän vanhempien tietoja (nimet, syntymä- ja kuolinpaikat), vihkipäivä ja -aika, puoliso, puolison vanhempien tietoja (nimet, syntymä- ja kuolinpaikat), lapset (nimet, syntymä- ja kuolinpaikat), pienoiselämänkerta ja mahdolliset lähteet olemassa olevalle tiedolle.

Sovellukseen tulee myös sukuosio ja sukupuukaavio. Sukuosiosta löytyvät kaikki eri suvut, joita sovellukseen on lisätty ja nämä tiedot perustuvat suoraan käyttäjän syöttämiin henkilötietoihin. Sukupuukaavio perustuu suoraan perhetauluihin ja kaaviossa pystyy alustavasti tarkastelemaan yhtä perhetaulua kerrallaan. Myöhemmin kaavio tulee

mahdollisesti rakentumaan siten, että se yhdistää perhetauluja isommaksi kaavioksi tai mahdollisesti, että käyttäjä voi joko tarkastella yhtä perhetaulua kerralla tai yhtä suurta kokonaisuutta.

Kehitystyössä pitää pyrkiä ottamaan huomioon erilaiset tilanteet kuten esimerkiksi se, että samassa perheessä saattaa olla täysin saman nimisiä henkilöitä, joillakin henkilöillä ei ole ollenkaan sukunimeä, sukulaiset ovat menneet keskenään naimisiin, joku on nainut itselleen vuorotellen sisarukset ja joku on saattanut kuulua moneen eri sukukuun. Kaikki ei ole sukututkimuksessa tai sukuhistoriassa aina niin suoraviivaista ja yksinkertaista.

Sukututkimus on jatkunut useita vuosia henkilöllä, jolle sovellus tulee käytettäväksi. Tietoa on kerääntynyt arviolta noin 7000–8000 henkilöstä. Sovelluksen pitää pystyä käsittelemään tämän verran dataa sekä tietenkin kaikki data, jota sovellukseen tulevaisuudessa tulee. Tämän ei kuitenkaan pitäisi olla ongelma, sillä sovellus on tarkoitettu alustavasti vain yhden henkilön käytettäväksi. Jos sovellusta laajentaa myös useamman henkilön käytettäväksi, niin silloin pitää alkaa enemmän miettiä sovelluksen optimointia ja muita tämän kaltaisia asioita, jotta sovellus ei esimerkiksi toimi liian hitaasti.

Sovelluksessa käytetään frontendissä ReactJS:ää, backendissä Node.js:ää ja tietokantana MongoDB:tä. Versionhallintaan käytetään GitHubia ja Git Bashia, editorina Visual Code Studiota ja testailuun esimerkiksi Postmania. Sovelluksen tyylittelyyn käytetään osittain Bootstrappia. Käytetyt tekniikat valikoituivat sen perusteella, koska ne ovat moderneja tekniikoita. Sovelluksen lopullinen versio tulee pyörimään DigitalOceanin tarjoamassa pilvipalvelussa, johon se on linkitettynä suoraan GitHubin kautta, jotta mahdolliset päivitykset päivittyvät GitHub muutoksien myötä suoraan sovellukseen ja mahdollisimman paljon toimintoja olisi niin sanotusti automatisoitu.

Koska sovellus on tarkoitettu käytettäväksi lähtökohtaisesti tietokoneella ja on vain tarkoitettu yhden henkilön käytettäväksi, siinä ei tulla huomioimaan esimerkiksi skaalaavuutta suurille käyttäjämäärille, suorituskyvyn arviointia tai käyttöliittymän hiomista täysin mobiiliystävälliseksi. Sovelluksen laajentamismahdollisuus on kuitenkin hyvä pitää mielessä ja näitä asioita on hyvä ottaa hieman huomioon. Vaikka sovellus ei olekaan lähtökohtaisesti tarkoitettu mobiililaitteille, tulee sen silti olla mahdollisimman

responsiivinen, sillä esimerkiksi selaimen kokoa voi muuttaa ja tämä ei saa hajottaa sovellusta täysin.

Opinnäytetyössä kerrotaan sovelluksen toteutustavasta yleisesti ja mitä erilaisia tekniikoita kehitystyössä käytettiin tai olisi voinut käyttää. Näissä kerrotaan myös enemmän esimerkiksi käytetyistä kirjastoista ja riippuvuuksista. Lisäksi käydään läpi itse sovelluksen ohjelmointia ja miltä käyttöliittymä lopulta tuli näyttämään.

2 Toteutustapa yleisesti

Sovellus rakentuu kolmesta eri osasta, jotka ovat frontend, backend ja tietokanta. Näistä frontend ja backend molemmat on toteutettu JavaScriptillä. Frontend on toteutettu Reactilla, joka on JavaScript -kirjasto (React, n.d.-b) ja backend Node.js:llä, joka on alustariippumaton ajoympäristö JavaScript-koodin suorittamiseen palvelimella (Node.js, n.d.). Tietokantana sovelluksessa käytetään MongoDB:tä, joka on niin sanottu dokumenttitietokanta (Full stack open, n.d.-b).

Frontend ja backend kommunikoivat keskenään HTTP-pyyntöjen avulla, jonka jälkeen backend tekee esimerkiksi tarvittavat muutokset palvelimen puolella tietokantaan tai vaihtoehtoisesti hakee halutut tiedot tietokannasta. Sovelluksessa on hyödynnetty valmiita kirjastoja kuten esimerkiksi Axiosta, Dotenviä ja Expressiä, joiden ansiosta sovelluskehitys on helpompaa ja nopeampaa. Käyttöliittymän tyylittelyssä on osittain hyödynnetty Bootstrap CSS-kehystä, jonka avulla sovelluksesta saa esimerkiksi helpommin responsiivisemmat.

Ohjelmistokehityksessä frontend ja backend toiminnallisuudet ovat omia ”kokonaisuuksiaan”, mutta niin sanottuun tuotantoon frontendistä luotiin oma tuotantoversio, joka liitettiin backendin juurihakemiston yhteyteen. Tuotantoversion koodi on vietynä GitHubiin, jonka kautta se on suoraan linkitettynä DigitalOceanin tarjoamaan pilvipalveluun. Tämän takia uusin tuotantoversio päivittyy aina automaattisesti käyttäjälle internettiin, kun GitHubiin päivitetään sovellukseen tulleet muutokset.

Lähtökohtaisesti sovellus on tarkoitettu vain käytettäväksi tietokoneella, eikä niinkään mobiililaitteilla. Tabletille soveltuva responsiivisuus voi olla tulevaisuudessa mahdollisesti tarpeellinen, joten sitä on pyritty hieman ottamaan huomioon sovelluksessa. Käyttöliittymä

on pyritty toteuttamaan mahdollisimman selkeästi ja suoraviivaisesti, jotta sen käyttäminen olisi mahdollisimman yksinkertaista.

Tietoturva on nykyaikana varsin tärkeä ottaa huomioon ja se on otettu huomioon sovelluksessa sekä front- että backend puolella. Kaikki tieto on suojattuna sisäänkirjautumisen taakse, ja ilman sisäänkirjautumista ja myös käyttäjällä voimassa olevaa tokenia ei pysty menemään mihinkään ei-sallittuun URL-polkuun. Sovelluksessa ei tällä hetkellä ole mahdollista luoda uutta käyttäjää suoraan sovelluksessa esimerkiksi rekisteröitymisen kautta, vaan uuden käyttäjän luominen on täysin ylläpitäjän hallussa. Ylläpitäjä voi luoda uuden käyttäjän esimerkiksi Postmanin kautta oikeanlaisen ja voimassa olevan tokenin avulla.

Sovelluksen vaihtoehtoinen toteutustapa olisi voinut olla toteuttaa sovellus tekemällä websovelluksen sijaan siitä työpöytäsovellus. Työpöytäsovellus olisi ollut siinä mielessä hyvä ratkaisu toteuttaa sovellus, sillä lähtökohtaisesti se on tarkoitettu ja suunniteltu vain käytettäväksi yhdelle henkilölle. Haasteet tulevat kuitenkin siinä vaiheessa, kun sovellusta tulee tarve päivittää.

2.1 Käytetyt tekniikat

2.1.1 React (frontend)

React on Metan kehittämä JavaScript-kirjasto, joka käyttää JSX-syntaksia (JavaScript XML) ja se on tarkoitettu käyttöliittymien kehitykseen (Ström, 2019). React on teknologia, joka ratkaisi niin sanottujen SPA-käyttöliittymien (single-page-application) ongelman. Tällä tarkoitetaan esimerkiksi sitä, että sen avulla voidaan kehittää käyttöliittymä, joka koostuu vain yhdestä sivusta ja jolle latautuu erilaisia näkymiä riippuen siitä, mitä käyttäjä sivulla tekee. Reactilla käyttöliittymän rakentaminen on yksinkertaista ja se pilkkoo sivun elementit pienempiin osiin eli niin sanottuihin komponentteihin. Komponentteja pystyy hyödyntämään muualla ilman, että samaa koodia joutuu kirjoittamaan useaan kertaan ja ne toimivat JavaScript funktioina. (Tecci, 2020)

2.1.1.1 Axios

Axios-kirjastoa käytetään siihen, että React sovellus pystyy kommunikoimaan API:en kanssa. Se on lupauspohjainen (promise) kirjasto ja sitä käytetään Node.js:n ja selaimen kanssa asynkronisten HTTP-pyyntöjen tekemiseen. Axiossa on toimintoja, jotka sopivat yhteen sovellusten kanssa, jotka käyttävät Reactia. (KnowledgeHut, 2023)

2.1.1.2 Bootstrap

Bootstrap on CSS-kehys, joka on tarkoitettu nettisivujen ja -sovellusten luomiseen ja/tai tyylittelyyn. Sitä hyödynnetään tekemään nettisivuista ja -sovelluksista mahdollisimman responsiivisia – varsinkin, jos halutaan näiden olevan mahdollisimman responsiivisia myös mobiililaitteilla. Bootstrap perustuu HTML:ään, CSS:ään ja JavaScriptiin sekä sillä pystyy nopeuttamaan omaa kehitystyötä ilman, että tarvitsee tuhlata aikaa peruskomentojen ja –toimintojen miettimiseen. (Zola, 2022)

2.1.1.3 D3, D3-dTree & Loadash

Otsikon mukaiset kolme kirjastoa tarvitsee, jotta pystyy rakentamaan tässä sovelluksessa olevan sukupuukaavion. D3 on kirjasto, joka mahdollistaa D3:n tulostuksen uudelleenohjauksen Reactin virtuaaliseen DOMiin. Kyseinen kirjasto kääntää koodin React-komponentiksi ja se sisältää myös sarjan D3-mallipohjaisia kaavioita, jotka on muunnettu React-komponenteiksi. (React D3 Library, n.d.) D3-dTree on kirjasto, joka visualisoi datapuita useammalla ”vanhemmalla” D3:n päälle rakennettuna (Gärtner, n.d.). Loadash helpottaa JavaScriptin käyttöä poistamalla vaivannäön esimerkiksi taulukoiden, numeroiden, objektien ja merkkijonojen käsittelystä. Sen modulaariset metodit soveltuvat erinomaisesti taulukoiden, objektien, merkkijonojen iterointiin, arvojen manipulointiin ja testaamiseen sekä koostefunktioiden luomiseen. (Lodash, n.d.)

2.1.2 Node.js (backend)

Node.js on selaimesta riippumaton JavaScript -tulkki, joka pohjautuu Googlen Chrome selaimeen kehitettyyn JavaScript -tulkkiin nimeltä V8 (Tuura, 2015). Node.js:n päätarkoitus

on tarjota nopeutta tietojen käsittelyyn sekä asiakkaan (client) ja palvelimen (server) väliseen kommunikointiin. Tästä syystä Node.js tarjoaa tapahtumavetoisia kahdensuuntaisia yhteyksiä asiakkaan ja palvelimen välillä. Node.js on myös kevyt ja sillä on kyky käyttää JavaScriptiä sekä frontendin että backendin puolella. (Ravikiran A S, 2023)

2.1.2.1 Bcrypt

Bcrypt on kirjasto, joka mahdollistaa salasanojen hajauttamisen (hashing) ja suolaamisen (salting). Näillä toiminnoilla salasana muuttuu selkokielisestä salasanasta uniikiksi arvoksi, joka on vaikea kääntää takaisin selkokieliseksi salasanaksi. Salasanan hajauttaminen tarkoittaa salasanan antamista hajautusalgoritmille uniikin arvon luomiseksi ja suolaaminen lisää satunnaisen merkkijonon salasanaan ennen sen hajauttamista. (Gathoni, 2023)

2.1.2.2 CORS

Cross-Origin Resource Sharing (CORS) on HTTP-otsikkoon (header) perustuva mekanismi, joka mahdollistaa palvelimen ilmoittaa selaimelle, mistä alkuperästä muutoin kuin sen omasta, se sallii resurssien lataamisen. Näytä alkuperiä ovat esimerkiksi verkkotunnus, protokolla tai portti. (MDN Web Docs, 2023) Esimerkiksi tämän sovelluksen kohdalla websovelluksen selaimessa suoritettava JavaScript-koodi saa lähtökohtaisesti kommunikoida vain samassa originissa olevan palvelimen kanssa. Eli esimerkiksi, jos frontend on avattuna portissa 3001 ja backend portissa 3003, niiden origin ei ole sama. (Full stack open, n.d.-a)

2.1.2.3 DotEnv

Ympäristömuuttujat ovat muuttujia, jotka määritellään sovellukselle sen ulkopuolelta kuten esimerkiksi pilvipalvelun tai käyttöjärjestelmän kautta. Node.js:ssä ympäristömuuttujat ovat hyvä ja turvallinen tapa määrittää muuttujia, jotka eivät usein muutu. Tällaisia ovat esimerkiksi salasanat, URL-osoitteet ja todennusavaimet, joita ei halua jakaa ulkopuolisille tahoille. DotEnv on itsessään kevyt npm-paketti, joka lataa ympäristömuuttujat tiedostosta .env objektiin process.env. Tätä tiedostoa tai sen sisältöä ei pidä puskea esimerkiksi GitHubiin tai minnekään muuhun julkiseen varastoon esimerkiksi mahdollisten väärinkäytösten vuoksi. (Stork, 2021)

2.1.2.4 Express

Express on Node.js-websovelluskehys, joka tarjoaa ominaisuuksia verkko- ja mobiilisovellusten rakentamiseen. Sitä voidaan käyttää yksisivuisten, monisivuisten tai hybridisovellusten rakentamiseen. Express on rakennettu Node.js:n päälle ja sen avulla pystyy hallitsemaan palvelimia ja reittejä. Sen tarkoituksena on esimerkiksi helpottaa ja nopeuttaa API:en ja verkkosovellusten tekemistä. (Sharma, 2023)

2.1.2.5 Jsonwebtoken

JSON Web Token (JWT) on standardi, joka määrittelee tiiviin ja itsenäisen tavan turvallisesti välittää tietoa osapuolten välillä JSON-objektina. Sen pienen koon ansiosta, JWT-tunnuksia on helppo siirtää joko URL:in, POST-parametrin tai HTTP-otsikon kautta. JWT:tä käytetään pääasiassa autentikointiin ja se voidaan allekirjoittaa digitaalisesti joko julkisella tai yksityisellä avainparilla. (Sufiyan, 2023)

2.1.2.6 Mongoose

Mongoose on MongoDB-objektien mallinnustyökalu, joka on suunniteltu toimimaan asynkronisessa ympäristössä. Se tukee tässä projektissa käytettyä Node.js:ää. Sen avulla MongoDB:tä voidaan käyttää sovelluksen tietokantana. (Npm, 2023)

2.1.2.7 Nodemon

Nodemon on komentorivityökalu, joka helpottaa Node.js-sovelluksen kehittämistä. Se tarkkailee projektihakemistoa ja käynnistää sovelluksen uudelleen, kun se havaitsee muutoksia. (O'Reilly, n.d.) Tämä tarkoittaa sitä, että kun sovellusta kehittää backendin puolella, niin esimerkiksi ilman tätä työkalua sovellus pitää aina sulkea ja käynnistää uudelleen, jos haluaa muutoksien tulevan voimaan.

2.1.3 MongoDB (tietokanta)

MongoDB on dokumenttitietokanta, jota käytetään websovellusten rakentamiseen. Se soveltuu kaikille tunnetuille ohjelmointikielille ja mahdollistaa sovelluksen rakentamisen ilman työlästä tietokannan määrittelyä. MongoDB poikkeaa esimerkiksi SQL-tietokannoista siten, että siinä ei tallenneta tietoa tauluihin riveinä tai sarakkeina, vaan jokainen tietue MongoDB-tietokannassa on dokumentti. Se mahdollistaa sen, että sovellukset voivat hakea tiedon JSON-muodossa. (MongoDB, n.d.)

2.1.4 Muut tekniikat

2.1.4.1 DigitalOcean

DigitalOcean on pilvipalvelutarjoaja, joka tarjoaa pilvipalveluita ja infrastruktuuria palveluna. Sitä käytetään droppien (droplets) hallintaan ja seurantaan, jossa jokainen droppi on Linux-pohjainen virtuaalikone. (Perforce, 2022) DigitalOceanin voi esimerkiksi linkittää GitHubin kanssa yhteen ja aina, kun kehittäjä puskee muutokset tietokoneeltaan GitHubiin, niin silloin sovellus hakee uudet tiedot GitHubista ja päivittää ne pilvessä pyörivään sovellukseen.

2.1.4.2 Git Bash

Git Bash on Microsoft Windows -sovellus, jota käytetään Gitin suorittamiseen komentoriviltä (Git for Windows, n.d.). Se emuloi bash-ympäristöä Windowsilla, mikä mahdollistaa Unix-komentojen käytön Windows-käyttöjärjestelmässä. Sitä voidaan käyttää komentojen suorittamiseen Git-repositorioiden ja Git-elementtien kanssa. (Marijan, n.d.)

2.1.4.3 GitHub

Git itsessään on komentoriviohjelma ja GitHub on koodin säilytys- ja yhteistyöalusta versionhallinnan ja yhteistyöntueksi (GitHub, n.d.). Se Helpottaa yhteistyötä kehittäjien kesken ja tarjoaa hajautetun versionhallinnan. Sitä käytetään esimerkiksi ohjelmistoprojektien tallentamiseen, seurantaan ja yhteistyöhön useissa eri yhteyksissä. GitHubin käyttäjiin lukeutuu esimerkiksi erilaiset yritykset, oppilaitokset, yhteisöt ja

ohjelmistokehittäjät. Tärkeitä termejä sen käyttäjille ovat esimerkiksi pull request, merge, push, commit ja clone. (Lutkevich, 2023)

2.1.4.4 Postman

Postman on API-alusta, jonka avulla voidaan rakentaa ja käyttää API-rajapintoja. Se mahdollistaa API:en tutkimisen, virheenjäljityksen ja testauksen. (Postman, n.d.) Lisäksi Postman tarjoaa mahdollisuuden synkronoida tietoja muiden käyttäjien kanssa, mikä helpottaa ja nopeuttaa tiimityöskentelyä. (Swiatkowski, 2022)

2.1.4.5 Visual Studio Code

Visual Studio Code on lähdekoodieditori, joka on saatavilla Windowsille, macOS:lle ja Linuxille. Siihen on sisäänrakennettu tuki JavaScriptille, TypeScriptille ja Node.js:lle sekä laaja ekosysteemi laajennuksia muille mielille ja suoritusympäristöille kuten esimerkiksi C#:lle, Javalle ja Pythonille. Visual Studio Codeen on mahdollista asentaa monipuolisesti erilaisia laajennuksia, joiden avulla omaa työskentelyä pystyy helpottamaan huomattavasti. Se tukee myös Gitiä, joten versionhallinta onnistuu myös sen kautta esimerkiksi GitHubiin. (Visual Studio Code, 2023)

2.2 Vaihtoehtoiset tekniikat

2.2.1 Frontend

Sovelluksen frontendin toteuttamiseen olisi voinut käyttää Reactin sijaan jotakin toista modernia ohjelmistokehystä kuten esimerkiksi Angularia tai Vueta. Nämä molemmat kehykset pohjautuvat JavaScriptiin ja soveltuvat hyvin websovelluksen kehittämiseen ja toteuttamiseen. Jokaisessa kehyksessä on omat hyvät ja huonot puolensa verrattuna toisiinsa (Technostacks, 2023).

Angular on rakennettu TypeScriptin päälle ja se sisältää esimerkiksi komponenttipohjaisen kehyksen skaalautuvien verkkosovellusten rakentamiseen, kokoelman hyvin integroituja kirjastoja ja kehittäjätyökaluja (Angular, 2022). Sen vahvuuksia ovat esimerkiksi virheetön

synkronisointi näkymän ja mallin välillä, muutoksien näkyminen välittömästi suoraan näkymässä ja mahdollisuus eristää näkymä ja tietojoukot. Sen sijaan heikkouksia ovat esimerkiksi monimutkainen MVC-malli, joka kehittäjän täytyy oppia sekä se on raskaampi kehys kuin esimerkiksi React. (Technostacks, 2023)

Vue perustuu standardiin HTML:ään, CSS:ään ja JavaScriptiin ja sitä käytetään käyttöliittymien rakentamiseen. Se tarjoaa deklaratiivisen ja komponenttipohjaisen ohjelmointimallin. (Vue.js, n.d.) Sen vahvuuksia ovat esimerkiksi helppo integroitavuus, pieni koko ja sen tarjoama kaksisuuntainen viestintä. Heikkouksia taas ovat puutteelliset lisäosat, kielirajoitukset ja se ei sovellu laajaan skaalaamiseen. (Technostacks, 2023)

2.2.2 Backend

Sovelluksen backendin toteuttamiseen olisi voinut käyttää Noden sijaan esimerkiksi C# tai Javaa. C# (Sharp) on ohjelmointikieli, jonka on kehittänyt Microsoft, ja joka toimii .NET kehyksen päällä. Sitä käytetään esimerkiksi verkko-, työpöytä- ja mobiilisovellusten kehittämiseen. (W3Schools, n.d.-a) Java on ohjelmointikieli ja alusta, joka on objektipohjainen. (Javapoint, n.d.) Sen omistaa Oracle ja sitä käytetään yli kolmessa miljardissa laitteessa. Javaa pystyy ja on käytetty esimerkiksi Android mobiili-, työpöytä- ja verkkosovelluksissa. (W3Schools, n.d.-b)

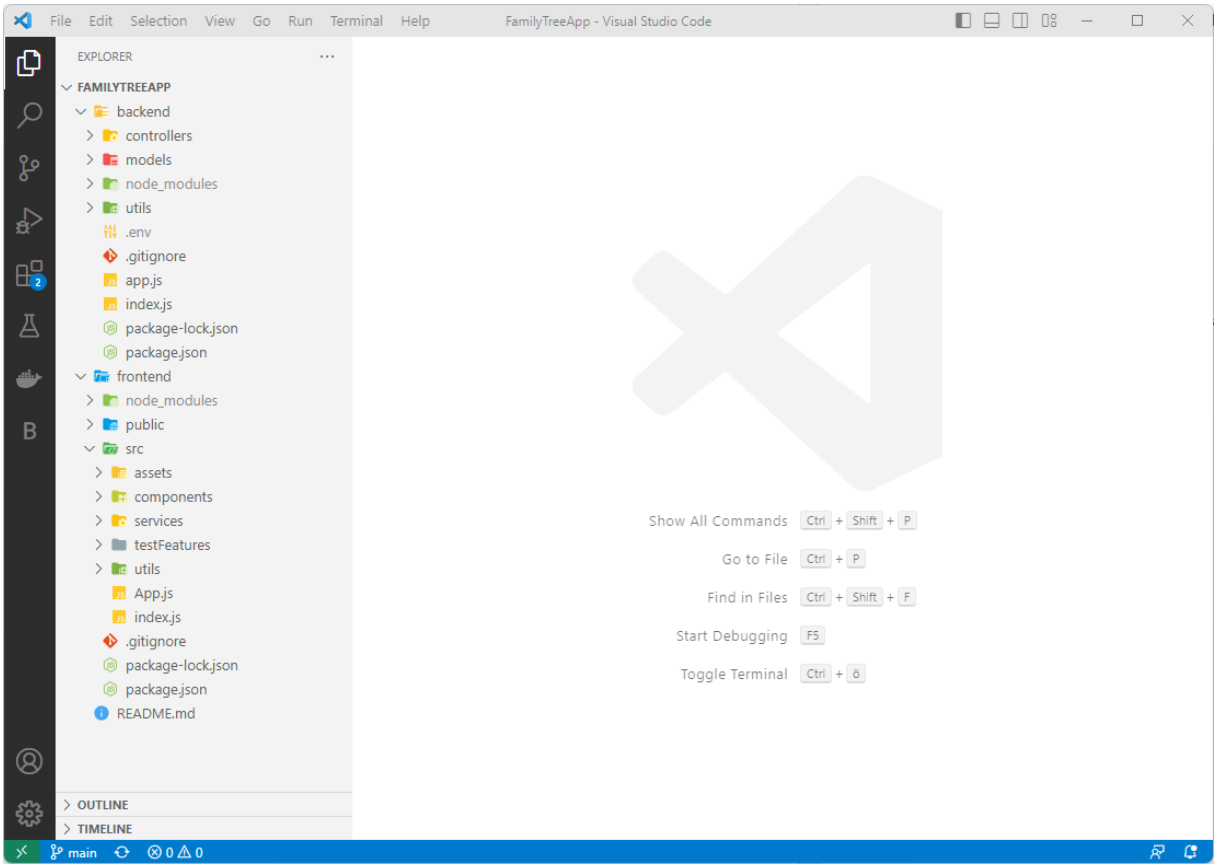
2.2.3 Tietokanta

Sovelluksen tietokantana olisi voinut käyttää MongoDB:n sijaan esimerkiksi SQLiteä, MariaDB:tä tai MySQL:ää. Siinä missä MongoDB on NoSQL-dokumenttipohjainen tietokanta, SQLite on ohjelmistokirjasto, joka tarjoaa relaatiotietokannan hallintajärjestelmän (Geeksforgeeks, 2020). MariaDB on avoimen lähdekoodin relaatiotietokantaratkaisu, joka tarjoaa SQL-ominaisuuksia tietojen käyttämiseen. Se on haarautunut (fork) MySQL:stä ja se on kirjoitettu useille eri ohjelmointikielille. (Francois, 2022) MySQL on avoimen lähdekoodin relaatiotietokannan hallintajärjestelmä, joka tallentaa dataa taulukoihin, jotka koostuvat riveistä ja sarakkeista. MySQL:n käyttäjät voivat esimerkiksi määritellä, muokata, hallita ja hakea dataa kyselykielen avulla. Se on yksi maailman suosituimpia avoimen lähdekoodin tietokantajärjestelmiä. (Drake, 2020)

3 Ohjelmointi

Sovelluksen ohjelmistokehityksessä hakemistorakenne on juuritasolla jaettu front- ja backendiin (kuva 1). Näistä molemmat ovat jo itsessään omia projekteja, jotka ovat sovelluksessa yhdistetty toimimaan yhtenä kokonaisuutena. Backendin alta löytyvät kaikki backendin toiminnallisuuteen liittyvät kuten esimerkiksi moduulit ja muut tiedostot. Frontendin alta taas vastaavasti löytyvät kaikki frontendin toiminnallisuuteen liittyvät kuten esimerkiksi moduulit ja muut tiedostot.

Kuva 1. Projektin hakemistorakenne.



3.1 Frontend

Frontendin hakemiston juuritasolta löytyvät Noden moduulit, julkinen (public) ja lähdehakemisto (src). Julkinen hakemisto sisältää nimensä mukaisesti julkisia tiedostoja kuten esimerkiksi HTML-tiedoston ja muita staattisia tiedostoja, jotka eivät muutu React-sovelluksen suorituksen aikana. Lähdehakemisto sisältää sovelluksen lähdekoodin eli

esimerkiksi komponentit, tyylitiedostot ja muut resurssit, jotka ovat tarpeen sovelluksen toiminnan kannalta. Lisäksi juuritasolta löytyvät .gitignore, package-lock.json ja package.json tiedostot. Näistä gitignoreen lisätään nimensä mukaisesti kaikki tiedostot, jotka jätetään gitin ulkopuolelle. Package.json sisältää sovelluksen tiedot ja riippuvuudet, kun taas package-lock.json varmistaa, että kaikki riippuvuudet ovat yhteensopivia ja asennettu oikein.

Lähdehakemiston juuresta löytyvät esimerkiksi muiden komponenttien ja/tai hakemistojen lisäksi App.js ja index.js tiedostot. App.js on React-sovelluksen pääkomponentti, joka määrittelee sovelluksen käyttöliittymän rakenteen ja käyttäjän näkemän sisällön. Index.js on puolestaan Reactin aloitustiedosto, joka aloittaa sovelluksen suorittamisen.

Kuva 2. Reactin index.js tiedoston sisältö.

frontend > src > index.js

```
1  import React from "react";
2  import ReactDOM from "react-dom/client";
3  import { BrowserRouter as Router } from "react-router-dom";
4  import "bootstrap/dist/css/bootstrap.min.css";
5
6  import App from "./App";
7
8  ReactDOM.createRoot(document.getElementById("root")).render(
9    <Router>
10     <App />
11   </Router>
12 );
13
```

Kuva 3. Reactin App.js tiedoston sisältö.

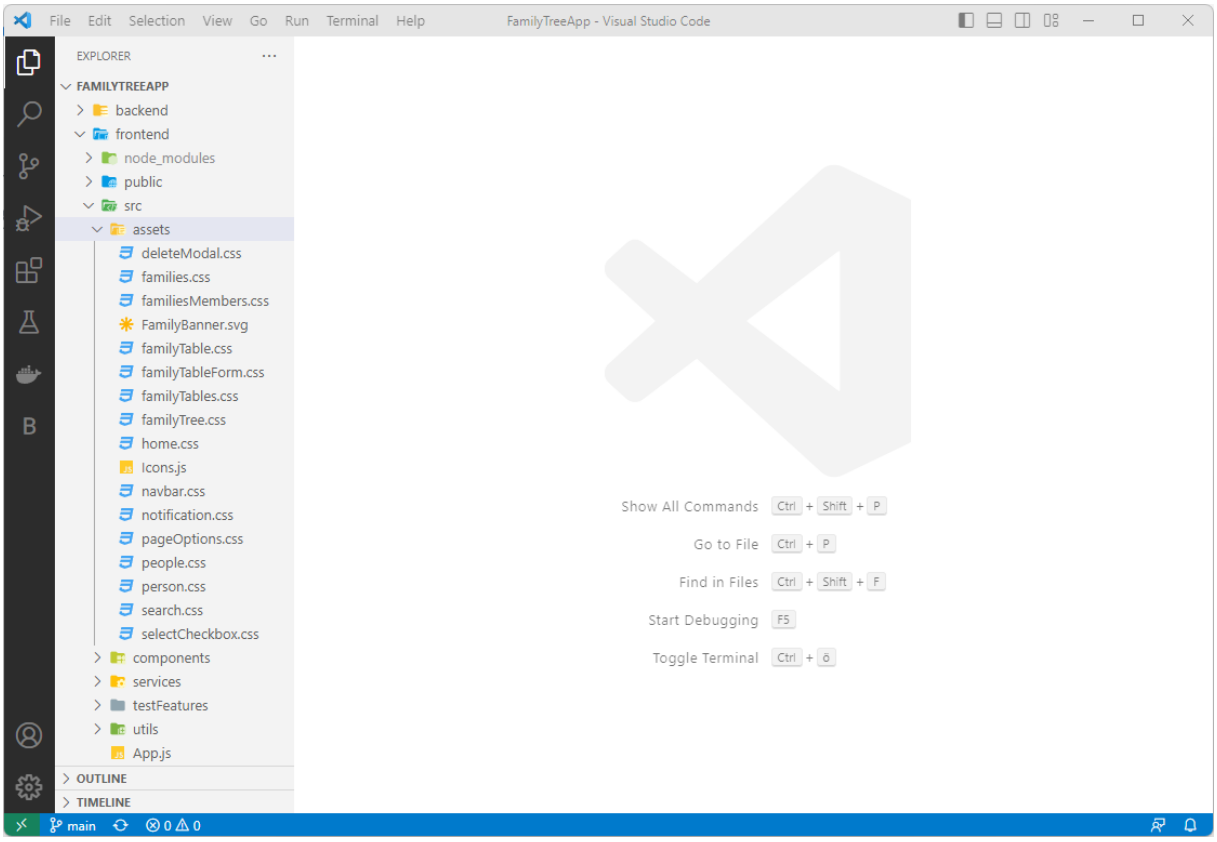
frontend > src > App.js > ...

```
import { Route, Routes } from "react-router-dom";
import PrivateRoutes from "./utils/PrivateRoutes";
import Families from "./components/Families";
import FamilyTree from "./components/FamilyTree";
import Home from "./components/Home";
import People from "./components/People";
import Person from "./components/Person";
import NewPerson from "./components/NewPerson";
import FamilyTables from "./components/FamilyTables";
import FamilyTable from "./components/FamilyTable";
import NewFamilyTable from "./components/NewFamilyTable";
import EditPerson from "./components/EditPerson";
import FamiliesMembers from "./components/FamiliesMembers";
import EditFamilyTable from "./components/EditFamilyTable";
import Login from "./components/Login";
import SearchResults from "./components/SearchResults";

const App = () => {
  return (
    <>
      <Routes>
        <Route element={<PrivateRoutes />}>
          <Route path="/families" exact element={<Families />} />
          <Route path="/families/members" exact element={<FamiliesMembers />} />
          <Route path="/people" exact element={<People />} />
          <Route path="/people/create" exact element={<NewPerson />} />
          <Route path="/people/:id" exact element={<Person />} />
          <Route path="/people/edit/:id" exact element={<EditPerson />} />
          <Route path="/familytables" exact element={<FamilyTables />} />
          <Route
            path="/familytables/create"
            exact
            element={<NewFamilyTable />}
          />
          <Route path="/familytables/:id" exact element={<FamilyTable />} />
          <Route
            path="/familytables/edit/:id"
            exact
            element={<EditFamilyTable />}
          />
          <Route path="/familytree" exact element={<FamilyTree />} />
          <Route path="/search" exact element={<SearchResults />} />
          <Route path="/" exact element={<Home />} />
        </Route>
        <Route path="/login" exact element={<Login />} />
      </Routes>
    </>
  );
};

export default App;

```

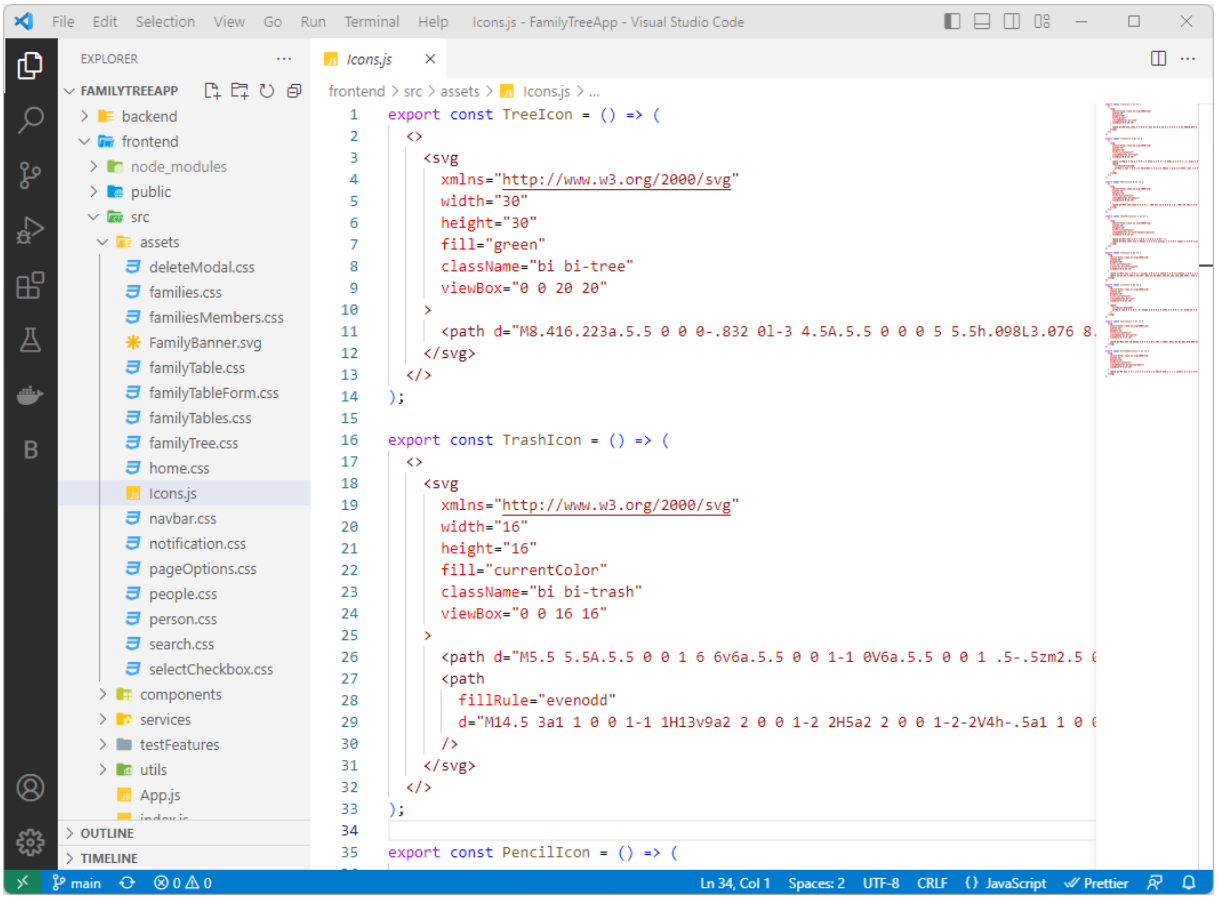
3.1.1 Resurssit (assets)

Resurssit voivat sisältää erilaisia tiedostoja kuten esimerkiksi kuvia, fontteja, tyylitiedostoja ja muita staattisia tiedostoja, joita sovellus käyttää ja lataa tarpeen mukaan (Expo, n.d.). Tässä sovelluksessa resurssit sisältävät sovelluksen käyttämät ikonit, kuvat ja tyylitiedostot (kuva 4). Tyylitiedostot on jaettu ja nimetty omiksi tiedostoiksi sen mukaan, mihin sivuun tai paikkaan ne liittyvät ja kaikki ikonit on sijoitettu yhteen tiedostoon omiksi komponenteikseen.

Kuva 4. Reactin resurssit.



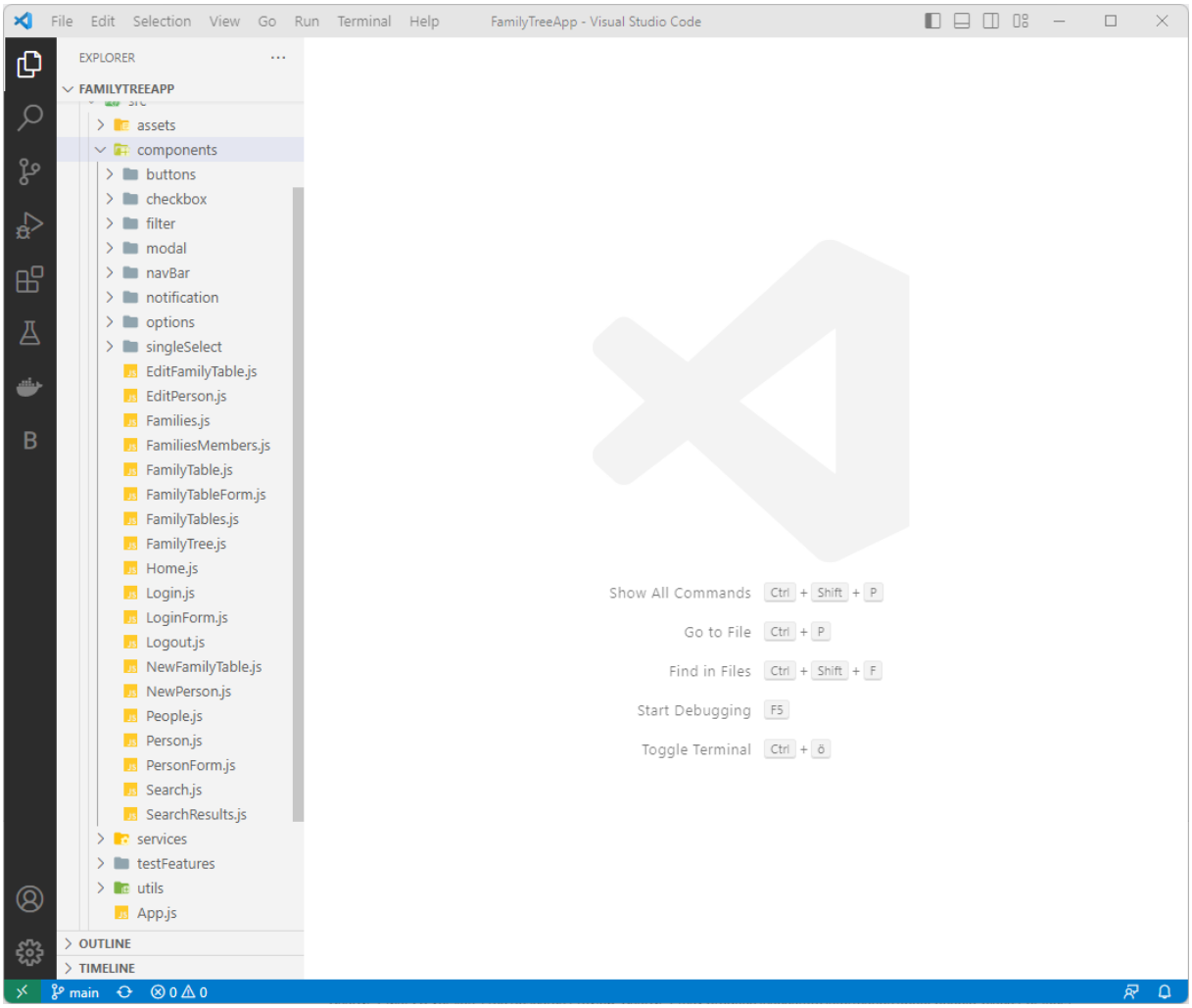
Kuva 5. Sovelluksen käyttämiä ikoneita.



3.1.2 Komponentit (components)

Komponentit ovat itsenäisiä ja uudelleenkäytettäviä koodin osia. Käsitteellisesti komponentit ovat kuin JavaScript-funktioita ja ne hyväksyvät mielivaltaisia syötteitä ja palauttavat elementtejä, jotka kuvaavat, mitä näytöllä pitäisi näkyä. (React, n.d.) Tässä sovelluksessa on pyritty siihen, että samaa asiaa ei olisi tarvetta tehdä kahteen kertaan ja sitä mukaan on pyritty jakamaan koodia useaksi eri komponentiksi. Sovelluksesta löytyvät esimerkiksi nappi-, filtteri- ja navigointikomponentit tai komponentteja. Lisäksi jokainen sivu on jaettu omaksi komponentikseen.

Kuva 6. Reactin komponentit.



Tarkastellaan esimerkkinä kotisivulle vievän napin komponenttia (kuva 7). Tuodaan React Router -kirjastosta ensimmäiseksi useNavigate -koukku (hooks) komponentti. Koukku antaa komponentille mahdollisuuden navigoida eri reittien välillä ilman, että se tarvitsee historiatietoja.

Koodipätkä 1. useNavigaten tuominen.

```
import { useNavigate } from "react-router";
```

Luodaan HomeButton -komponentti, joka näyttää napin ”Etusivulle” -tekstillä, ja joka käyttää aiemmin tuotua ”useNavigate” -koukkua navigoimaan käyttäjän takaisin etusivulle. Käyttäjän painaessa nappia, ”onClick” -tapahtumakäsittelijä kutsuu anonyymia

nuolifunktiota, joka kutsuu ”navigate” -funktiota ”useNavigate” -koukusta. Näin käyttäjä siirtyy polkuun ”/” eli etusivulle, kun nappia painetaan.

Koodipätkä 2. HomeButton -komponentti.

```
const HomeButton = () => {
  const navigate = useNavigate();

  return (
    <button
      className="btn btn-outline-warning searchPageButton"
      onClick={() => navigate("/")}
    >{`<- Etusivulle`}</button>
  );
};
```

Määritellään komponentille vientimahdollisuus (export), jotta sitä voidaan käyttää muissa tiedostoissa tai komponenteissa. Oletusarvoisena (default) viedään haluttu nappikomponentti muualle käytettäväksi.

Koodipätkä 3. HomeButton -komponentin vieminen.

```
export default HomeButton;
```

Kuva 7. Kotisivunapin komponentin koodi kokonaisuudessaan.

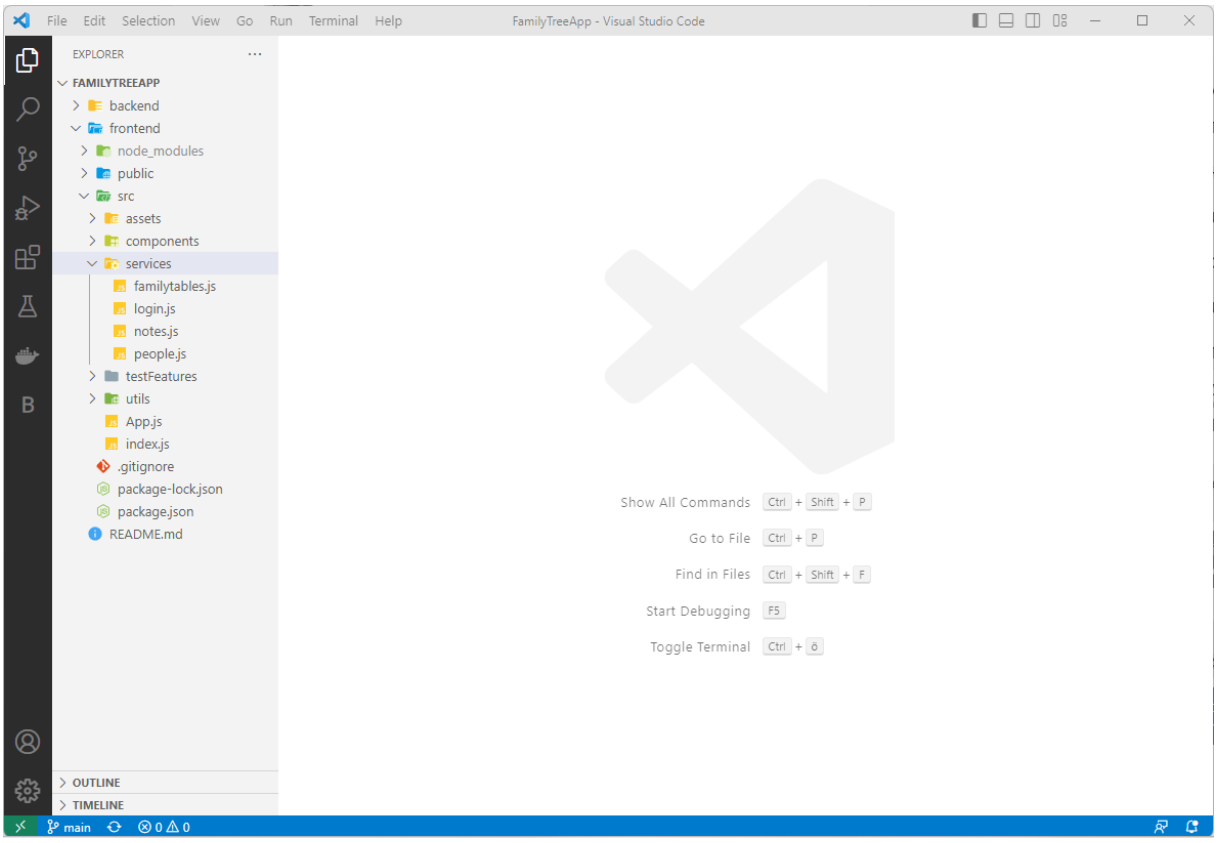
frontend > src > components > buttons > HomeButton.js > ...

```
1   import { useNavigate } from "react-router";
2
3   const HomeButton = () => {
4     const navigate = useNavigate();
5
6     return (
7       <button
8         className="btn btn-outline-warning searchPageButton"
9         onClick={() => navigate("/")}
10      >{`<- Etusivulle`}</button>
11    );
12  };
13
14  export default HomeButton;
15
```

3.1.3 Palvelut (services)

Palvelut mahdollistavat viestinnän käyttöliittymä- ja taustateknologian välillä (Arsenault, 2017). Tässä sovelluksessa on omat palvelut perhetauluille, sisäänkirjautumiselle, muistiinpanoille ja henkilöille (kuva 8). Nämä sisältävät sellaisia toiminnallisuuksia kuten kaikkien tietojen hakemisen, yksittäisen tiedon hakemisen, uuden tiedon luomisen, tiedon poistamisen ja tiedon päivittämisen.

Kuva 8. Reactin käyttämät palvelut.



Tarkastellaan esimerkkinä henkilöpalvelun koodia ja/tai tiedostoa (kuva 9). Tuodaan ensimmäisenä Axios-kirjaston axios-moduuli, jota käytetään HTTP-pyyntöjen lähettämiseen. Tämä voi sisältää esimerkiksi tietojen noutamisen API:sta. Lisäksi määritellään muuttujalle vakioarvo.

Koodipätkä 4. Axios-kirjasto ja muuttujan vakioarvo.

```
import axios from "axios";
const baseURL = "/api/people";
```

Luodaan funktio, joka palauttaa HTTP-pyynnön otsikot Authorization-avaimella. Funktion avulla tarkistetaan, onko tokeni tallennettuna selaimen muistiin (local storage). Jos tokeni on olemassa, funktio palauttaa objektin, joka sisältää Authorization-avaimen ja sen arvona on ”Bearer”-merkkijono ja tokeni välilyönnillä erotettuna. Jos tokenia ei ole olemassa, funktio palauttaa tyhjän objektin. Tällä varmistetaan ja suojataan se, että vain kirjautuneet käyttäjät pääsevät käsiksi tiettyihin reitteihin.

Koodipätkä 5. GetHeaders-funktio.

```
const getHeaders = () => {
  const token = JSON.parse(localStorage.getItem("token"))?.token;
  return token ? { Authorization: `Bearer ${token}` } : {};
};
```

Määritellään funktio, joka käyttää Axios-kirjastoa lähettämään GET-pyyntö aikaisemmin määritetyn muuttujan vakioarvon osoitteeseen. Funktio sisältää myös otsikkoparametrin, joka välitetään pyynnön mukana ja joka sisältää getHeaders-funktion paluuarvon, joka antaa pyynnölle oikean tokenin oikeassa muodossa. Kun pyyntö on lähetetty, odotetaan vastausta. Kun vastaus on vastaanotettu, funktion paluuarvo on vastauksen sisältö JSON-muodossa. Funktio käyttää asynkronista ohjelmointia varmistaakseen, että vastaus on vastaanotettu ennen kuin se palauttaa vastauksen sisällön. Tässä tapauksessa vastaus sisältää kaikki henkilötiedot, jotka löytyvät tietokannasta.

Koodipätkä 6. Haetaan palvelimelta kaikki tiedot.

```
const getAll = async () => {
  const response = await axios.get(baseURL, { headers: getHeaders() });
  return response.data;
};
```

Määritellään funktio, jonka toimintaperiaate on lähes sama kuin edellisen, mutta tällä kertaa funktio lähettää GET-pyynnön tietyn henkilön tietoihin ID-arvon perusteella. Sen sijaan, että palautettaisiin kaikki henkilötiedot tietokannasta, niin tämä funktio palauttaa vain yhden henkilön tiedot tietokannasta.

Koodipätkä 7. Haetaan yksittäisen henkilön tiedot.

```
const getById = async (id) => {
  const response = await axios.get(`${baseURL}/${id}`, {
    headers: getHeaders(),
  });
  return response.data;
};
```

Määritellään funktio, jonka toimintaperiaate on sama kuin aiemmin, mutta se poikkeaa pyynnön osalta aikaisempiin. Funktio lähettää POST-pyynnön uuden henkilön lisäämiseksi määritettyyn osoitteeseen. Eli aina, kun halutaan lisätä uusi henkilö tietokantaan, niin tehdään pyyntö tämän funktion kautta.

Koodipätkä 8. POST-pyyntö uudelle henkilölle.

```
const create = async (personObject) => {
  const response = await axios.post(baseURL, personObject, {
    headers: getHeaders(),
  });
  return response.data;
};
```

Määritellään funktio, jonka toimintaperiaate poikkeaa muista pyynnön osalta. Funktio lähettää DELETE-pyynnön tietyn henkilön henkilötietoihin ID-arvon perusteella. Tämän funktion avulla tietokannasta saadaan poistettua tietyn henkilön tiedot.

Koodipätkä 9. DELETE-pyyntö henkilön poistamiseen.

```
const remove = async (id) => {
  const response = await axios.delete(`${baseURL}/${id}`, {
    headers: getHeaders(),
  });
  return response.data;
};
```

Määritellään funktio, jonka toimintaperiaate poikkeaa jälleen muista. Funktio lähettää PUT-pyynnön tietyn henkilön henkilötietoihin ID-arvon perusteella. Tämän funktion avulla henkilön tietoja pystytään muokkaamaan tietokantaan.

Koodipätkä 10. PUT-pyyntö henkilön muokkaamiseen.

```
const update = async (id, personObject) => {
```

```
  const response = await axios.put(`${baseURL}/${id}`, personObject, {
    headers: getHeaders(),
  });
  return response.data;
};
```

Lopuksi viedään aikaisemmin määritetyt funktiot yhdessä objektissa moduulista muualle käytettäväksi.

Koodipätkä 11. Viedään palvelut käytettäväksi muualle.

```
export default {
  getAll,
  getById,
  create,
  remove,
  update,
};
```

Kuva 9. Henkilöpalveluiden koodi kokonaisuudessaan.

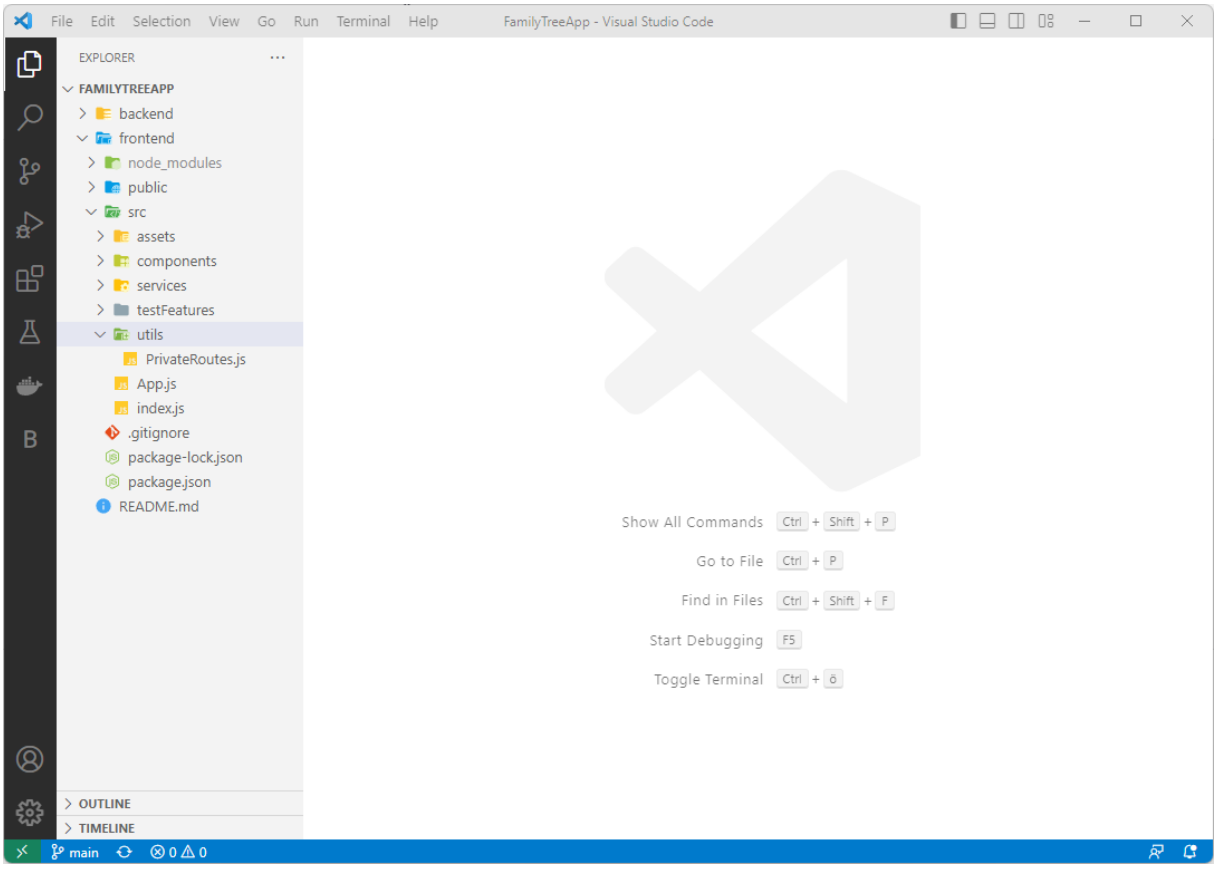
frontend > src > services > people.js > ...

```
import axios from "axios";
const baseURL = "/api/people";

const getHeaders = () => {
  const token = JSON.parse(localStorage.getItem("token"))?.token;
  return token ? { Authorization: `Bearer ${token}` } : {};
};

const getAll = async () => {
  const response = await axios.get(baseURL, { headers: getHeaders() });
  return response.data;
};

const getById = async (id) => {
  const response = await axios.get(`${baseURL}/${id}`, {
    headers: getHeaders(),
  });
  return response.data;
};

const create = async (personObject) => {
  const response = await axios.post(baseURL, personObject, {
    headers: getHeaders(),
  });
  return response.data;
};

const remove = async (id) => {
  const response = await axios.delete(`${baseURL}/${id}`, {
    headers: getHeaders(),
  });
  return response.data;
};

const update = async (id, personObject) => {
  const response = await axios.put(`${baseURL}/${id}`, personObject, {
    headers: getHeaders(),
  });
  return response.data;
};

export default {
  getAll,
  getById,
  create,
  remove,
  update,
};
```

3.1.4 Apufunktiot ja -työkalut (utils)

Apufunktioiden ja/tai -työkalujen avulla ohjelmoinnissa voidaan suorittaa tietyt toiminnot helpommin ja tehokkaammin. Ne voivat olla esimerkiksi erilaisia yleiskäyttöisiä funktioita, luokkia tai moduuleja, jotka auttavat esimerkiksi tietojen muuntamisessa, virheiden käsittelyssä tai käyttöliittymän luomisessa. Tässä sovelluksessa käytetään apufunktiota, joka tarkistaa, onko käyttäjä kirjautunut sisään vai ei. Tämä toiminnallisuus aktivoituu silloin, kun käyttäjä yrittää käyttää jotakin yksityistä reittiä sovelluksessa.

Kuva 10. Sovelluksen frontendin käyttämät apufunktiot.



Tarkastellaan esimerkkinä yksityisen reitin tarkistukseen käytettyä komponenttia (kuva 11). Ensimmäisenä tuodaan React Router -kirjaston Outlet ja Navigate -komponentit sekä navigointimenun komponentti. Outlet on komponentti, joka näyttää reittien sisällön vasta, kun se vastaa reittiä, jota käyttäjä navigoi sovelluksessa. Navigate komponentti taas mahdollistaa reitityksen dynaamisen ohjaamisen.

Koodipätkä 12. Tuodaan Outlet, Navigate ja NavBar.

```
import { Outlet, Navigate } from "react-router-dom";
import NavBar from "../components/navBar/NavBar";
```

Luodaan itse komponentti, joka tarkistaa onko käyttäjä kirjautunut sisään tarkistamalla, onko tokeni tallennettuna selaimen muistiin. Jos tokeni on olemassa, näytetään käyttäjän haluaman sivun (Outletin) sisältö. Jos tokenia taas ei ole olemassa, ohjataan käyttäjä kirjautumissivulle.

Koodipätkä 13. PrivateRoutes -komponentti.

```
const PrivateRoutes = () => {
  const token = JSON.parse(localStorage.getItem("token"))?.token;

  return token ? (
    <div>
      <NavBar />
      <Outlet />
    </div>
  ) : (
    <Navigate to="/login" />
  );
};
```

Mahdollistetaan komponentin käyttäminen myös muissa moduuleissa.

Koodipätkä 14. Viedään PrivateRoutes-komponentti.

```
export default PrivateRoutes;
```

Kuva 11. Yksityisen reitin tarkistuksen koodi kokonaisuudessaan.

frontend > src > utils > PrivateRoutes.js > ...

```
1   import { Outlet, Navigate } from "react-router-dom";
2   import NavBar from "../components/navBar/NavBar";
3
4   const PrivateRoutes = () => {
5     const token = JSON.parse(localStorage.getItem("token"))?.token;
6
7     return token ? (
8       <div>
9         <NavBar />
10        <Outlet />
11      </div>
12    ) : (
13      <Navigate to="/login" />
14    );
15  };
16
17  export default PrivateRoutes;
18
```

3.2 Backend

Backendin hakemiston juuritasolta löytyvät Noden moduulit ja omat luodut hakemistot kuten esimerkiksi ohjaimet (controllers) ja mallit (moldels). Juuritasolta löytyvät myös .env, .gitignore, app.js ja index.js, package-lock.json ja package.json -tiedostot. App.js tiedosto on Node.js-sovelluksessa pääkomponentti, joka määrittelee sovelluksen reitityksen ja toiminnallisuuden ja index.js toimii sovelluksen aloitustiedostona. Muut tiedostot toimivat tai ovat samankaltaisia kuin frontendin puolella lukuun ottamatta .env-tiedostoa, jota ei ole frontendin puolella. Tässä kyseisessä tiedostossa määritellään ympäristömuuttujat kuten esimerkiksi MongoDB-tietokannan käyttämä URI, sovelluksen portti ja SECRET-muuttuja.

Kuva 12. Backendin index.js tiedoston sisältö.

backend > index.js > ...

```
1  const app = require("./app");
2  const config = require("./utils/config");
3  const logger = require("./utils/logger");
4
5  app.listen(config.PORT, () => {
6    logger.info(`Server running on port ${config.PORT}`);
7  });
8
```

Kuva 13. Backendin app.js tiedoston sisältö.

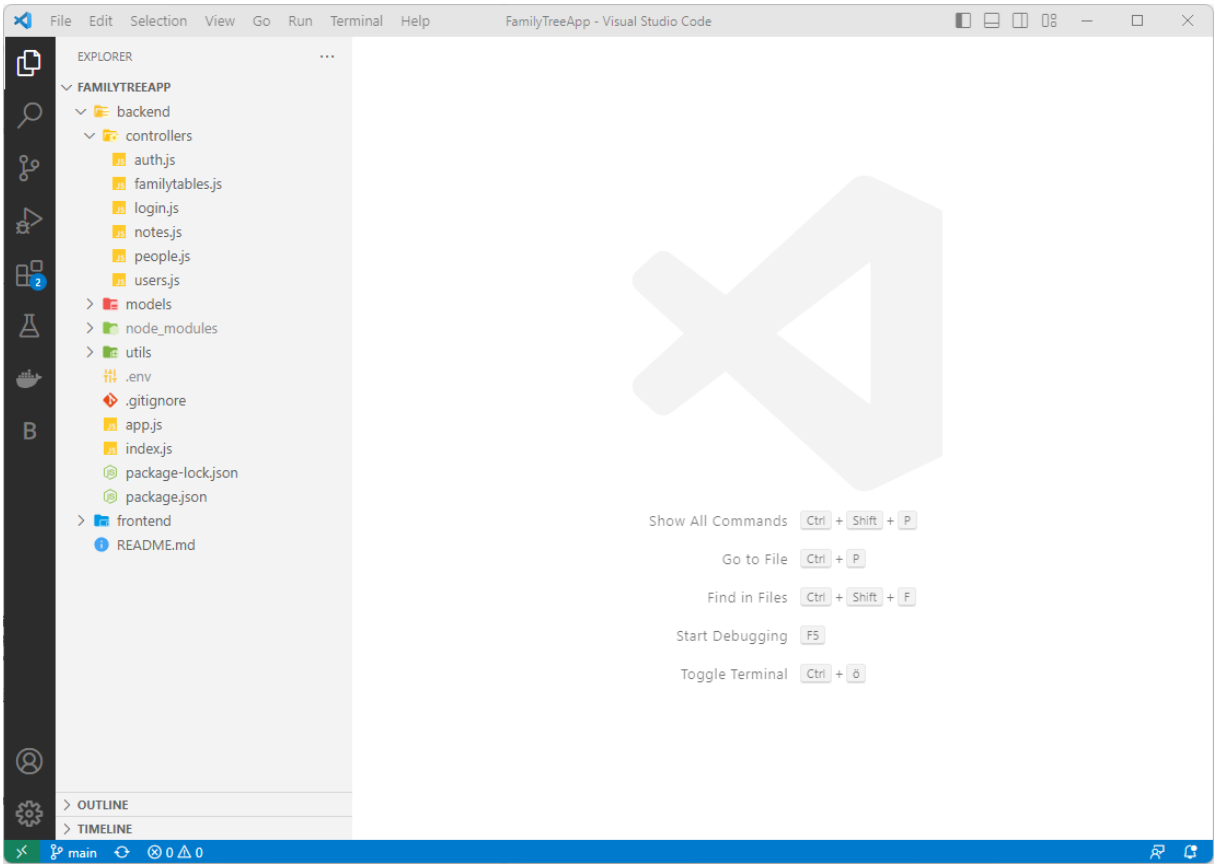
backend > app.js > ...

```
1   const config = require("./utils/config");
2   const express = require("express");
3   require("express-async-errors");
4   const app = express();
5   const cors = require("cors");
6   const peopleRouter = require("./controllers/people");
7   const usersRouter = require("./controllers/users");
8   const loginRouter = require("./controllers/login");
9   const notesRouter = require("./controllers/notes");
10  const familytableRouter = require("./controllers/familytables");
11  const middleware = require("./utils/middleware");
12  const logger = require("./utils/logger");
13  const mongoose = require("mongoose");
14  const { authRequired } = require("./utils/authRequired");
15
16  mongoose.set("strictQuery", false);
17
18  logger.info("connecting to", config.MONGODB_URI);
19
20  mongoose
21    .connect(config.MONGODB_URI)
22    .then(() => {
23      logger.info("connected to MongoDB");
24    })
25    .catch((error) => {
26      logger.error("error connection to MongoDB:", error.message);
27    });
28
29  app.use(cors());
30  app.use(express.json());
31  app.use(middleware.requestLogger);
32  app.use(express.static("build"));
33
34  app.use("/api/login", loginRouter);
35  app.use("/api/people", authRequired, peopleRouter);
36  app.use("/api/users", authRequired, usersRouter);
37  app.use("/api/familytables", authRequired, familytableRouter);
38  app.use("/api/notes", authRequired, notesRouter);
39
40  app.use(middleware.unknownEndpoint);
41  app.use(middleware.errorHandler);
42
43  module.exports = app;
44
```

3.2.1 Ohjaimet (controllers)

Ohjaimien (controllers) tehtävänä on vastaanottaa esimerkiksi käyttäjän syötteitä ja hallita ohjelman suoritusta niiden perusteella (W3Schools, n.d.). Sovelluksen backendistä löytyvät ohjaimet autentikoinnille, perhetauluille, sisäänkirjautumiselle, muistiinpanoille, henkilöille ja käyttäjille (kuva 14). Autentikoinnin ja sisäänkirjautumisen ohjaimet tarkistavat käyttäjän antamat kirjautumistiedot, generoitavat JWT-tokenin ja palauttavat sen käyttäjälle vastauksena, jos sisäänkirjautuminen onnistuu. Molemmat toteuttavat kirjautumistoiminnallisuutta, mutta molemmilla on oma tarkoituksensa ja käyttötapansa. Käytännössä autentikointia käytetään siihen, että tarkistetaan, onko käyttäjällä oikeus esimerkiksi johonkin polkuun ja sitä voidaan käyttää, missä tahansa paikassa. Sisäänkirjautumisen ohjaimen tarkoituksena on pelkästään käsitellä kirjautumispyyntöjä "/login"-polun kautta. Perhetaulu-, muistiinpano- ja henkilöohjaimet määrittelevät REST-rajapinnat, jotka käsittelevät niitä ja niiden sisältöä tietokannassa. Tämä mahdollistaa HTTP-pyyntöjen lähettämisen sovellukseen, joka suorittaa tarvittavat toiminnot tietokannan kanssa ja vastaa HTTP-vastauksilla. Ohjaimien toiminnallisuuksia ovat datan hakeminen, luominen, päivittäminen ja poistaminen tietokannasta. Käyttäjien ohjain tehtävänä on käyttäjätunnuksen luominen, hashata salasana, tallentaa käyttäjä tietokantaan ja palauttaa käyttäjä JSON-muotoisena vastauksena. Ohjaimen toiminnallinen tehtävä on toteuttaa käyttäjähallintaa.

Kuva 14. Sovelluksen ohjaimet.



Tarkastellaan esimerkkinä tarkemmin sisäänkirjautumisen käsittelystä vastaavan ohjaimen koodia (kuva 15). Ensimmäisenä tuodaan neljä eri moduulia tai riippuvuutta. Jsonwebtoken moduuli mahdollistaa JWT:n generoinnin ja tarkistamisen, Bcrypt-moduulin tarjoaa salauksen ja salasanan tarkistamisen, Express-moduuli mahdollistaa reittien määrittämisen ja hallinnan sekä User on sovelluksen oma moduuli, joka määrittelee käyttäjätietomallin.

Koodipätkä 15. Sisäänkirjautumisen ohjaimen tuodut moduulit tai riippuvuudet.

```
const jwt = require("jsonwebtoken");
const bcrypt = require("bcrypt");
const loginRouter = require("express").Router();
const User = require("../models/user");
```

Seuraavaksi määritellään Express-sovelluksessa käytettävä reitti, joka vastaa HTTP POST-pyyntöön, jonka polku on ”/”. Reitti käsittelee käyttäjän sisäänkirjautumisyrityksen, joka tapahtuu lähettämällä käyttäjätunnus ja salasana pyynnön rungossa. Kun pyyntö vastaanotetaan, etsitään tietokannasta kyseistä käyttäjätietoa käyttäjätunnuksen

perusteella. Jos käyttäjä löytyy tietokannasta, verrataan tallennetun käyttäjän salasanaa saapuneeseen salasanaan käyttäen bcrypt-kirjastoa. Jos käyttäjää ei löydy tai salasana on virheellinen, reitti vastaa 401-virheellä, jossa kerrotaan käyttäjälle, että joko käyttäjätunnus tai salasana on väärin. Jos käyttäjätunnus ja salasana ovat oikeat, generoidaan JWT-tokeni, joka sisältää käyttäjätunnuksen ja käyttäjän tunnisteen. Allekirjoitusavaimena käytetään ympäristömuuttujaa SECRET. Lopuksi lähetetään vastaus statuskoodilla 200, joka sisältää käyttäjätunnuksen, käyttäjän nimen ja tokenin. Statuskoodi 200 tarkoittaa onnistunutta kirjautumista.

Koodipätkä 16. Sisäänkirjautumisen ohjain.

```
loginRouter.post("/", async (request, response) => {
  const { username, password } = request.body;

  const user = await User.findOne({ username });
  const passwordCorrect =
    user === null ? false : await bcrypt.compare(password, user.passwordHash);

  if (!(user && passwordCorrect)) {
    return response.status(401).json({
      error: "invalid username or password",
    });
  }

  const userForToken = {
    username: user.username,
    id: user._id,
  };

  const token = jwt.sign(userForToken, process.env.SECRET);

  response
    .status(200)
    .send({ token, username: user.username, name: user.name });
});
```

Viimeisenä reitti viedään käytettäväksi myös muihin sovelluksen moduuleihin.

Koodipätkä 17. Sisäänkirjautumisen ohjaimen vienti.

```
module.exports = loginRouter;
```

Kuva 15. Sisäänkirjautumisen ohjaimen koodi kokonaisuudessaan.

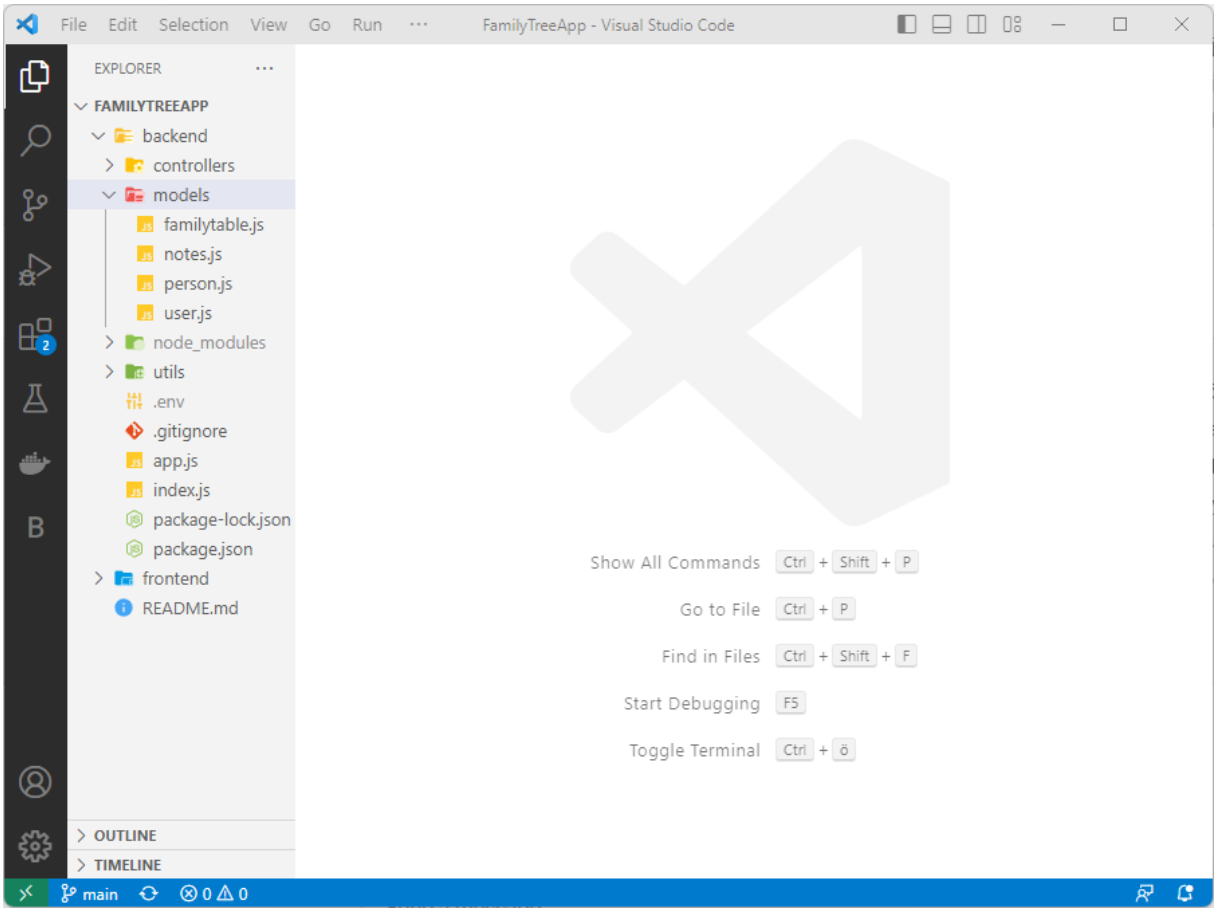
backend > controllers > login.js > ...

```
1   const jwt = require("jsonwebtoken");
2   const bcrypt = require("bcrypt");
3   const loginRouter = require("express").Router();
4   const User = require("../models/user");
5
6   loginRouter.post("/", async (request, response) => {
7     const { username, password } = request.body;
8
9     const user = await User.findOne({ username });
10    const passwordCorrect =
11      user === null ? false : await bcrypt.compare(password, user.passwordHash);
12
13    if (!(user && passwordCorrect)) {
14      return response.status(401).json({
15        error: "invalid username or password",
16      });
17    }
18
19    const userForToken = {
20      username: user.username,
21      id: user._id,
22    };
23
24    const token = jwt.sign(userForToken, process.env.SECRET);
25
26    response
27      .status(200)
28      .send({ token, username: user.username, name: user.name });
29  });
30
31  module.exports = loginRouter;
```

3.2.2 Mallit (models)

Mallit ovat tietorakenteita, joita sovellus käyttää datan muodon määrittämiseen (Anderson, n.d.). Sovelluksen backendistä löytyvät mallit perhetauluille, muistiinpanoille, henkilöille ja käyttäjille (kuva 16). Jokainen malli kuvaa nimensä mukaisesti sen mallin ominaisuuksia, mistä mallista on kyse. Esimerkiksi henkilömalli kuvaa henkilön ominaisuuksia kuten esimerkiksi etunimi, sukunimi, syntymä- ja kuolinpaikka sekä elämänkerta. Koska sovellus käyttää tietokantana MongoDB:tä, mallit määritellään käyttäen Mongoosea, joka on MongoDB:n objektimallinnustyökalu Node.js:lle.

Kuva 16. Sovelluksen mallit.



Tarkastellaan esimerkkinä tarkemmin käyttäjämallin koodia (kuva 17). Ensimmäisenä luodaan muuttuja nimellä ”mongoose” ja otetaan siihen käyttöön mongoose -moduuli.

Koodipätkä 18. Tuodaan mongoose.

```
const mongoose = require("mongoose");
```

Määritellään Mongoose-skeema, joka kuvaa käyttäjän ominaisuuksia ja niiden tietotyyppiä. Skeemalla on kolme kenttää, jotka ovat käyttäjätunnus, nimi ja salasanan hashaus. Käyttäjätunnukselle määritetään, että se on merkkijono, sen minimipituus on 3 merkkiä ja se on pakollinen tieto. Nimelle määritetään ainoastaan, että se on merkkijono. Salasanan hashaus määritellään merkkijonoksi ja se on myös käyttäjätunnuksen tavoin pakollinen tieto. Tämä skeema määrittelee sen, millaisia tietoja käyttäjistä tallennetaan tietokantaan ja mitkä niiden tietotyypit ovat.

Koodipätkä 19. Mongoose-skeema.

```
const userSchema = mongoose.Schema({
  username: {
    type: String,
    minlength: 3,
    required: true,
  },
  name: String,
  passwordHash: {
    type: String,
    required: true,
  },
});
```

Seuraava koodipätkä muuntaa tiedot käyttäjäobjektista JSON-muotoon. Kun käyttäjäobjekti muunnetaan JSON-muotoon, käyttäjän ”-id”-kentän arvo muunnetaan merkkijonoksi ”id”-kentässä ja samalla poistetaan ”_id”, ”__v” ja ”passwordHash” kentät JSON-oliosta. Tällä tavalla salasanatietoja ei paljasteta JSON-muodossa ja ”_id”-kenttä saadaan helpommin käyttöön JSON-muodossa.

Koodipätkä 20. Mongoose-skeeman muutokset.

```
userSchema.set("toJSON", {
  transform: (document, returnedObject) => {
    returnedObject.id = returnedObject._id.toString();
    delete returnedObject._id;
    delete returnedObject.__v;
    delete returnedObject.passwordHash;
  },
});
```

Luodaan käyttäjäolioille Mongoose-malli, joka perustuu ”userSchema”-skeemaan. Mongoose-malli on tietokannan käyttöliittymä, joka mahdollistaa esimerkiksi tietojen hakemisen, tallentamisen ja päivittäisen MongoDB-tietokantaan.

Koodipätkä 21. Käyttäjäolion Mongoose -malli.

```
const User = mongoose.model("User", userSchema);
```

Viimeisenä malli viedään käytettäväksi myös muihin sovelluksen moduuleihin.

Koodipätkä 22. Viedään käyttäjämalli.

```
module.exports = User;
```

Kuva 17. Käyttäjämallin koodi kokonaisuudessaan.

backend > models > user.js > ...

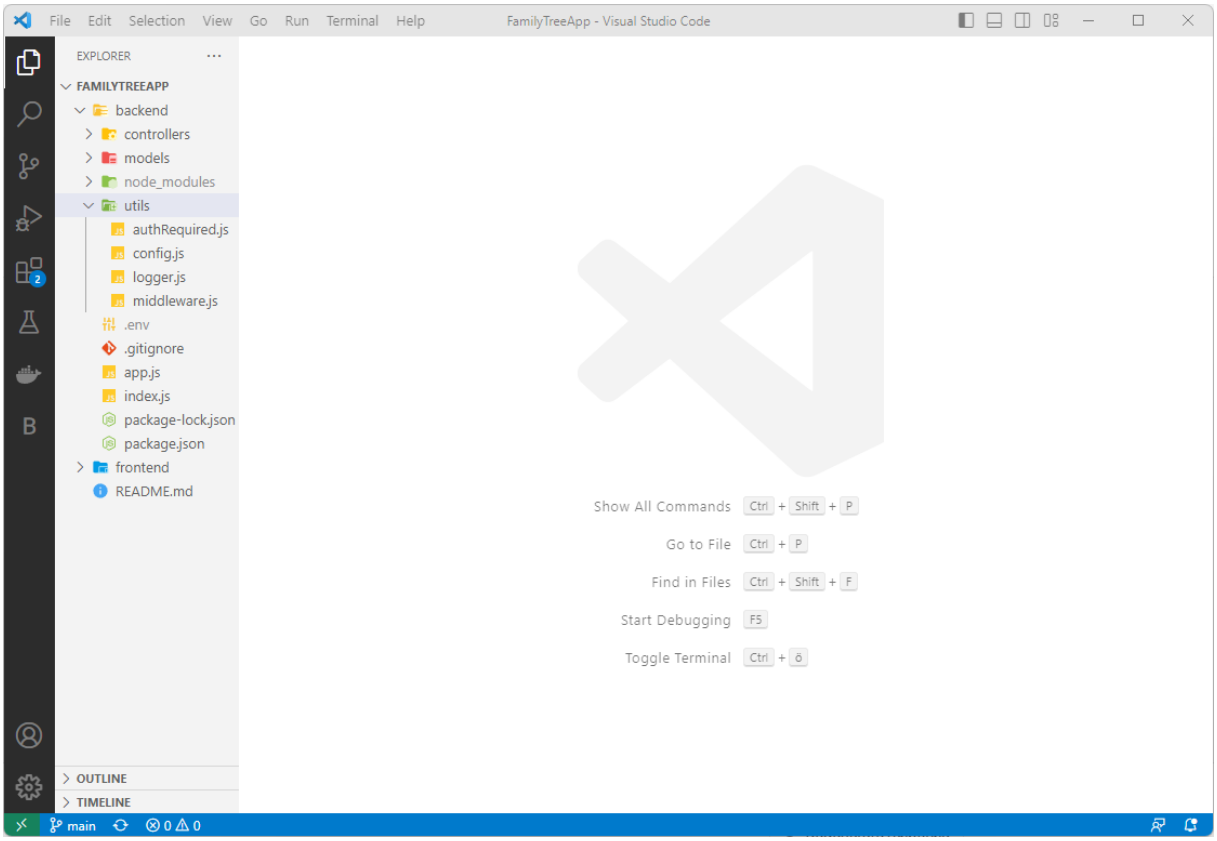
```
1   const mongoose = require("mongoose");
2
3   const userSchema = mongoose.Schema({
4     username: {
5       type: String,
6       minlength: 3,
7       required: true,
8     },
9     name: String,
10    passwordHash: {
11      type: String,
12      required: true,
13    },
14  });
15
16  userSchema.set("toJSON", {
17    transform: (document, returnedObject) => {
18      returnedObject.id = returnedObject._id.toString();
19      delete returnedObject._id;
20      delete returnedObject.__v;
21      delete returnedObject.passwordHash;
22    },
23  });
24
25  const User = mongoose.model("User", userSchema);
26
27  module.exports = User;
```

3.2.3 Apufunktiot ja -työkalut (utils)

Apufunktioiden ja/tai -työkalujen avulla ohjelmoinnissa voidaan suorittaa tietyt toiminnot helpommin ja tehokkaammin. Ne voivat olla esimerkiksi erilaisia yleiskäyttöisiä funktioita, luokkia tai moduuleja, jotka auttavat esimerkiksi tietojen muuntamisessa, virheiden käsittelyssä tai käyttöliittymän luomisessa. Tässä sovelluksessa on käytetty seuraavia apufunktioita tai -työkaluja: autentikointi pakollinen, konfigurointi, erilaisia konsolitulostuksia, tuntematon polku ja virheenkäsittely. Autentikointi pakollinen varmistaa sen, että käyttäjä on kirjautunut, ennen kuin hän voi käyttää tiettyjä sovelluksen resursseja. Konfiguroinnissa on määritettynä muuttujiin sovelluksen käyttämä portti ja MongoDB-tietokannan URI. Erilaisilla konsolituloksilla tulostellaan eri parametrejä esimerkiksi

virhetilanteiden yhteydessä sekä, kun tehdään jokin pyyntö kuten esimerkiksi POST, niin silloin konsoliin tulostuu kaikki tieto tästä tapahtumasta. Tuntematon polku on nimensä mukaisesti virhetilanne, jossa käyttäjä on yrittänyt polkuun, jota ei ole olemassa. Virheidenkäsittelyssä annetaan erilaisia virheilmoituksia esimerkiksi validointivirheen yhteydessä.

Kuva 18. Backendin apufunktiot ja -työkalut.



Tarkastellaan esimerkkinä autentikoinnin pakollisuuden määrittävää apufunktiota ja/tai -työkalua (kuva 19). Ensimmäisenä otetaan muuttujaan käyttöön jsonwebtoken-kirjasto, jolla voidaan luoda ja tarkistaa JWT-avaimia sekä toiseen muuttujaan mallihakemistosta löytyvä käyttäjämalli.

Koodipätkä 23. Tuodaan JWT-kirjasto ja käyttäjämalli.

```
const jwt = require("jsonwebtoken");
const User = require("../models/user");
```

Määritellään authRequired-funktio, joka suoritetaan aina, kun tietyn reitityspolun käyttö vaatii todennusta. Funktio tarkistaa, onko pyynnön otsikkotiedossa ”Authorization”-kenttää, joka alkaa ”Bearer”-merkkijonolla. Jos tämä puuttuu pyynnöstä, palautetaan virheilmoituksena tieto siitä, että käyttäjältä puuttuu tokeni tai se on väärä. Jos pyynnön otsikossa on haluttu kenttä, pyynnön arvo puretaan ja se yritetään tarkistaa JWT-kirjaston ”verify”-toiminnolla. Jos tokeni on oikea, puretaan se ja sen sisältämän tiedon perusteella etsitään käyttäjätunnusta tietokannasta. Jos käyttäjä löytyy, tallennetaan se pyyntöobjektiin ja siirrytään suorittamaan seuraavaa middleware-funktiota. Jos tokeni on viallinen tai käyttäjää ei löydy, palautetaan virheilmoitus statuskoodilla 401, joka sisältää viestin ”väärä tokeni” tai ”käyttäjää ei löytynyt”.

Koodipätkä 24. Autentikoinnin pakollisuus.

```
const authRequired = async (request, response, next) => {
  const authorization = request.get("Authorization");

  if (!authorization || !authorization.toLowerCase().startsWith("bearer ")) {
    return response.status(401).json({ error: "missing or invalid token" });
  }

  const token = authorization.substring(7);
  try {
    const decodedToken = jwt.verify(token, process.env.SECRET);
    const user = await User.findById(decodedToken.id);
    if (!user) {
      return response.status(401).json({ error: "user not found" });
    }
    request.user = user;
    next();
  } catch (error) {
    return response.status(401).json({ error: "invalid token" });
  }
};
```

Viedään funktio käytettäväksi myös muihin sovelluksen moduuleihin tai sovelluksen osiin.

Koodipätkä 25. Autentikoinnin pakollisuuden vienti.

```
module.exports = { authRequired };
```

Kuva 19. AuthRequired-moduulin koodi kokonaisuudessaan.

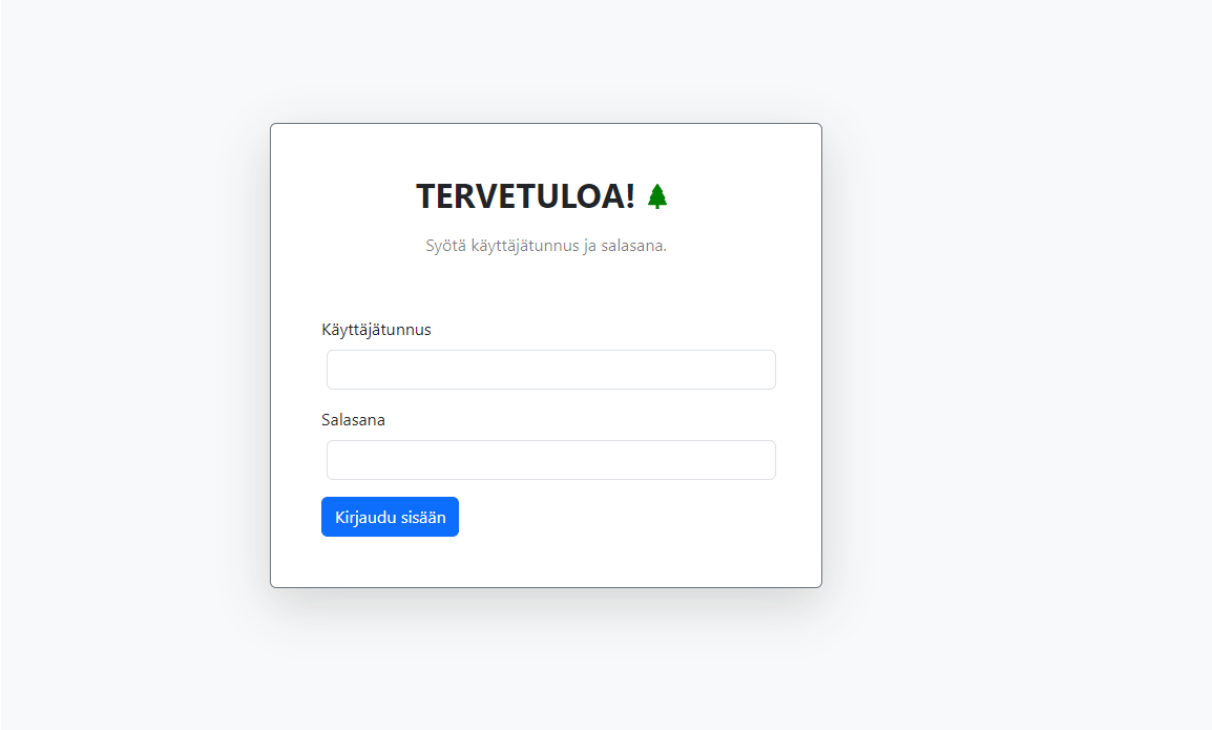
backend > utils > authRequired.js > ...

```
1   const jwt = require("jsonwebtoken");
2   const User = require("../models/user");
3
4   const authRequired = async (request, response, next) => {
5     const authorization = request.get("Authorization");
6
7     if (!authorization || !authorization.toLowerCase().startsWith("bearer ")) {
8       return response.status(401).json({ error: "missing or invalid token" });
9     }
10
11    const token = authorization.substring(7);
12    try {
13      const decodedToken = jwt.verify(token, process.env.SECRET);
14      const user = await User.findById(decodedToken.id);
15      if (!user) {
16        return response.status(401).json({ error: "user not found" });
17      }
18      request.user = user;
19      next();
20    } catch (error) {
21      return response.status(401).json({ error: "invalid token" });
22    }
23  };
24
25  module.exports = { authRequired };
26
```

4 Käyttöliittymä

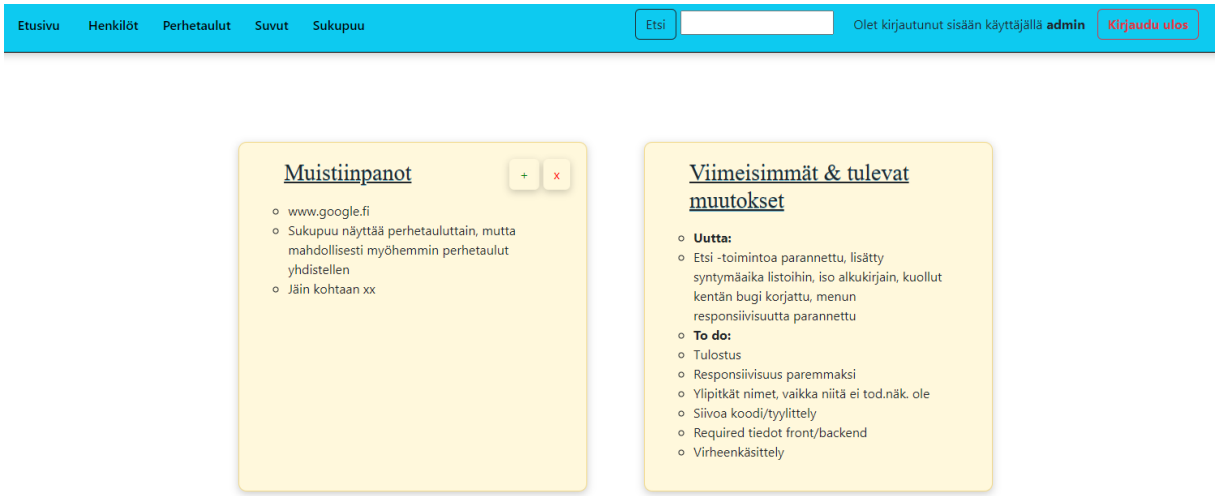
Lisähuomiona todettakoon, että kaikki kuvissa näkyvät tiedot ovat mielikuvituksen tuotetta. Sovellusta ylläpidetään DigitalOceanin tarjoamassa pilvipalvelussa ja sitä pystyy käyttämään suoraan selaimessa. Käyttäjälle ensimmäinen näkymä on sisäänkirjautumislomake (kuva 20), johon käyttäjän pitää syöttää käyttäjätunnus sekä salasana. Väärä käyttäjätunnus ja/tai salasana johtavat virheilmoitukseen.

Kuva 20. Sisäänkirjautumislomake.



Sisäänkirjautumisen jälkeen käyttäjälle avautuu etusivu, jossa on mahdollisuus lisätä muistiinpanoja kuten esimerkiksi tärkeistä linkeistä tai siitä mihin sukututkimus jäi ja mistä pitää jatkaa (kuva 21). Etusivulta löytyy myös ylläpitäjän informaatiolaatikko, jossa on listattuna uudet ja tulevat muutokset. Navigointivalikko löytyy sivun yläreunasta, joka näkyy sellaisenaan jokaisella sivulla. Valikosta löytyy eri sivut, etsi-ominaisuus sekä mahdollisuus uloskirjautumiselle.

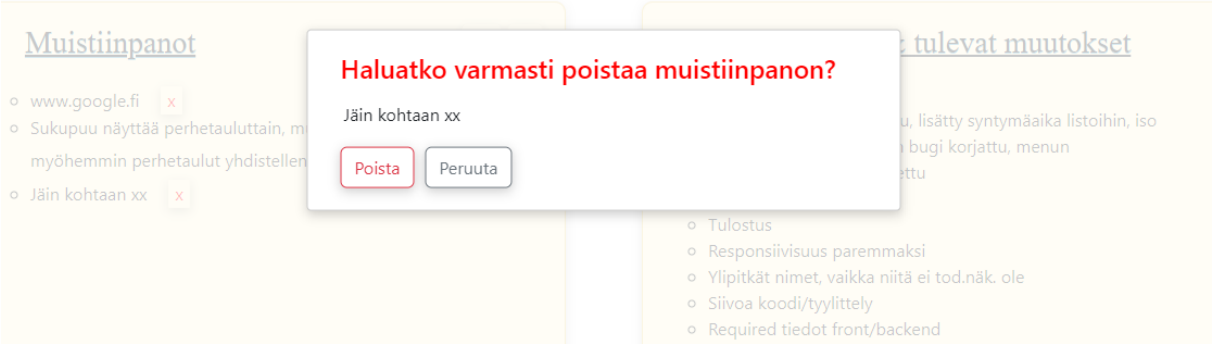
Kuva 21. Sovelluksen etusivu.



Muistiinpanoista löytyy lisäys- ja poistomahdollisuus. Oikeasta yläkulmasta käyttäjä voi joko sallia uuden muistiinpanon lisäämisen plusmerkistä tai vaihtoehtoisesti piilottaa lisäysmahdollisuuden miinusmerkistä. Kun lisäysmahdollisuus on auki, käyttäjälle ilmestyy tekstikenttä, johon käyttäjä voi kirjoittaa. Muistiinpanon poistamistoiminnallisuus on samankaltainen, x-merkillä avataan poistomahdollisuus ja yksittäisen muistiinpanon poistaminen tapahtuu uudelleen x-merkkiä painamalla. Kun käyttäjä on poistamassa muistiinpanoa, käyttäjälle aukeaa vielä varmistusikkuna tästä.

Kuva 22. Muistiinpanojen toiminnallisuudet.

Kuva 23. Poistovarmistusikkuna.



Henkilöt (kuva 24), perhetaulut (kuva 25) ja sukusivut (kuva 32) ovat keskenään melko samankaltaiset visuaalisesti, mutta niissä on kuitenkin hieman eroavaisuuksia. Henkilösivulla on listattuna kaikki sovellukseen lisätyt henkilöt, perhetauluissa kaikki sovellukseen lisätyt henkilöiden perhetaulut ja sukuihin on listattu kaikki henkilöt, joille on määritelty jokin suku tai sukuja. Kaikki muutokset tietoihin tapahtuu joko henkilö- tai perhetaulusivujen kautta.

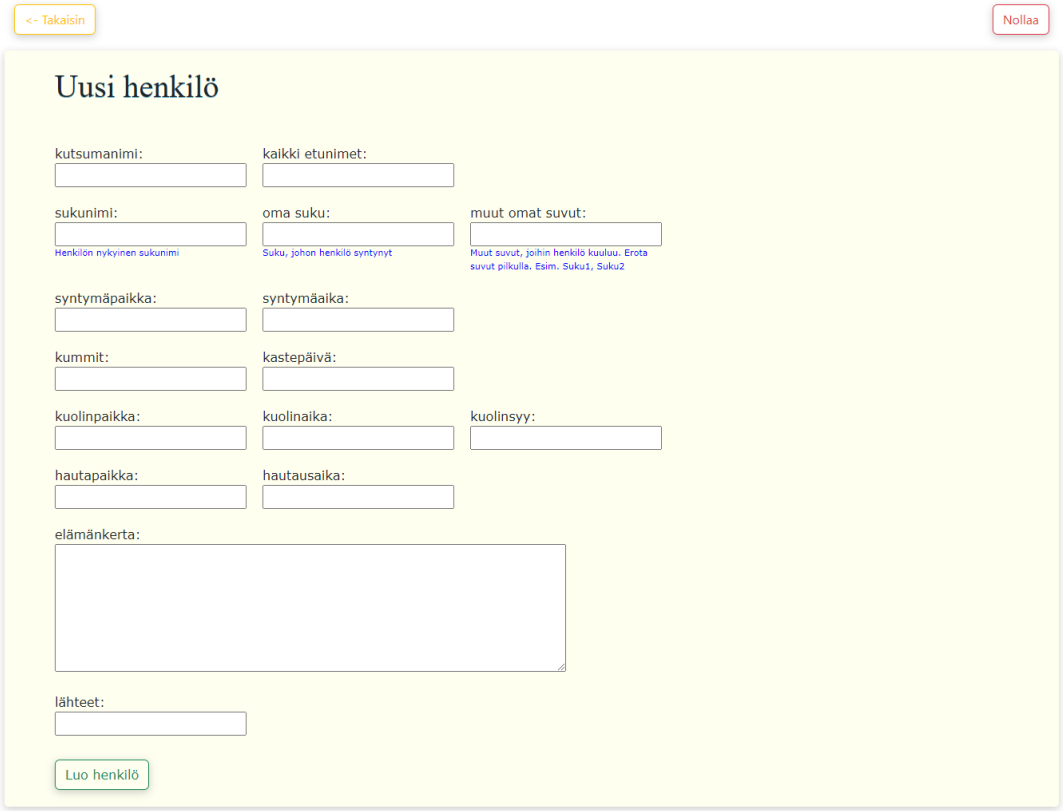
Henkilö- ja perhetaulusivuilla on mahdollisuus lisätä joko uusi henkilö tai perhetaulu. Lisäksi näiltä sivuilta löytyy mahdollisuus suodattaa hakutuloksia. Henkilö- ja perhetaulujen etusivuilla näytetään etunimet, sukunimi, syntymävuosi ja suku. Henkilöiden ja perhetaulujen perässä on mahdollisuus tietojen tarkasteluun, muokkaukseen tai poistamiseen. Sivulla näkyy myös sivunumero, ja käyttäjällä on mahdollisuus näyttää kerralla listassa joko 10, 25, 50, 75 tai 100 henkilöä tai perhetaulua.

Kuva 24. Henkilöt-sivu.

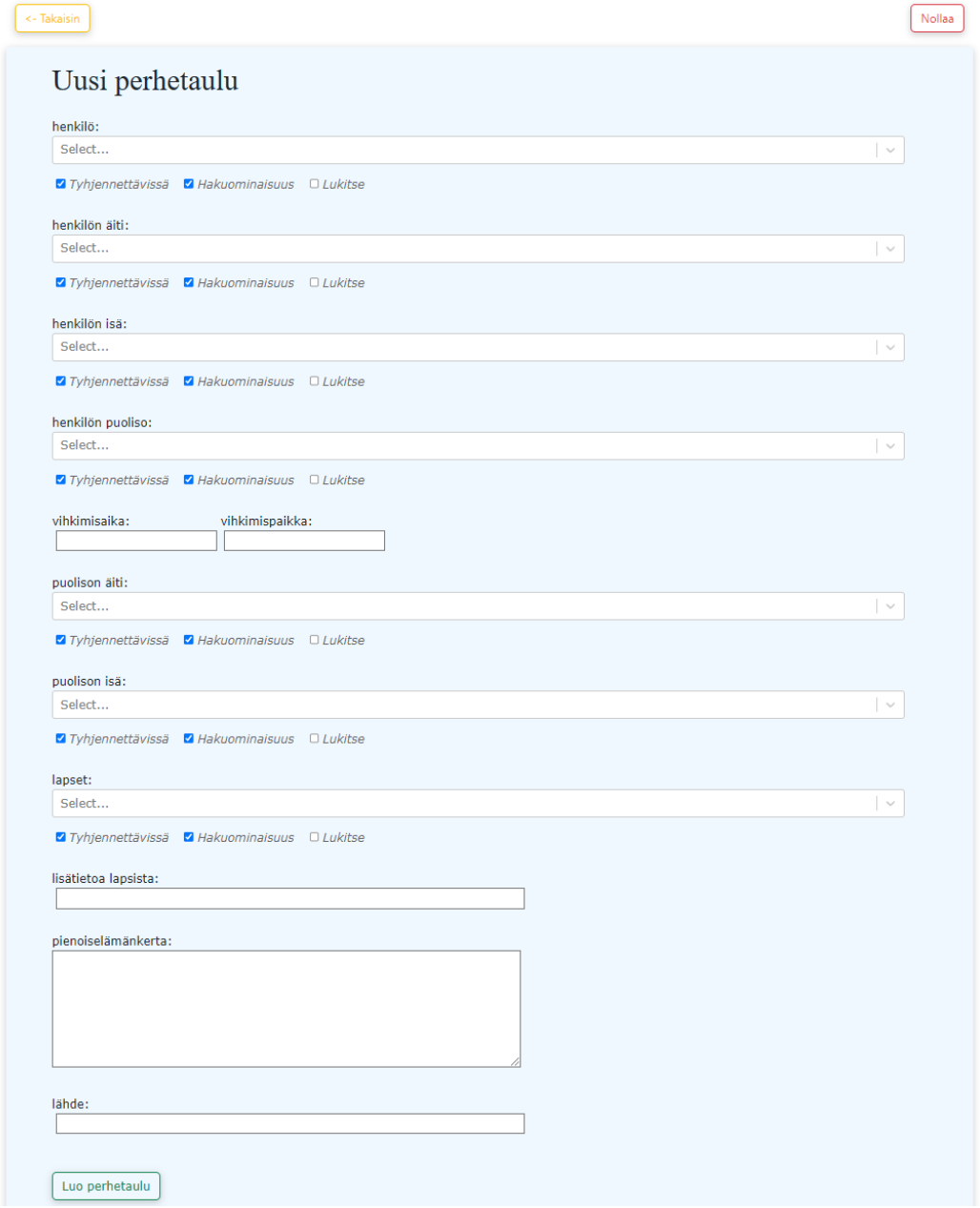
Kuva 25. Perhetaulut-sivu.

Uuden henkilön lisääminen tapahtuu antamalla henkilölle erilaisia tietoja (kuva 26). Näistä tiedoista pakollinen on vain kaikki etunimet ja sukunimi. Käyttäjällä on myös mahdollisuus nollata syötetyt tiedot, jos haluaa aloittaa alusta. Kun henkilö on luotu, niin silloin käyttäjälle tulee siitä ilmoitus. Uuden perhetaulun lisääminen tapahtuu samalla tavalla (kuva 27), mutta pakollinen valittava tieto on perhetaulussa vain henkilötieto. Kun henkilötietoja (kuva 28) tai perhetauluja muokkaa, se antaa samanlaisen lomakkeen kuin uuden tiedon luomisessa, mutta tällä kertaa siinä on mukana jo olemassa olevat tiedot. Henkilö- (kuva 29) ja perhetaulusivuilla (kuva 30) näitä tietoja voi sitten tarkastella lisää ja halutessaan myös tulostaa nämä paperille.

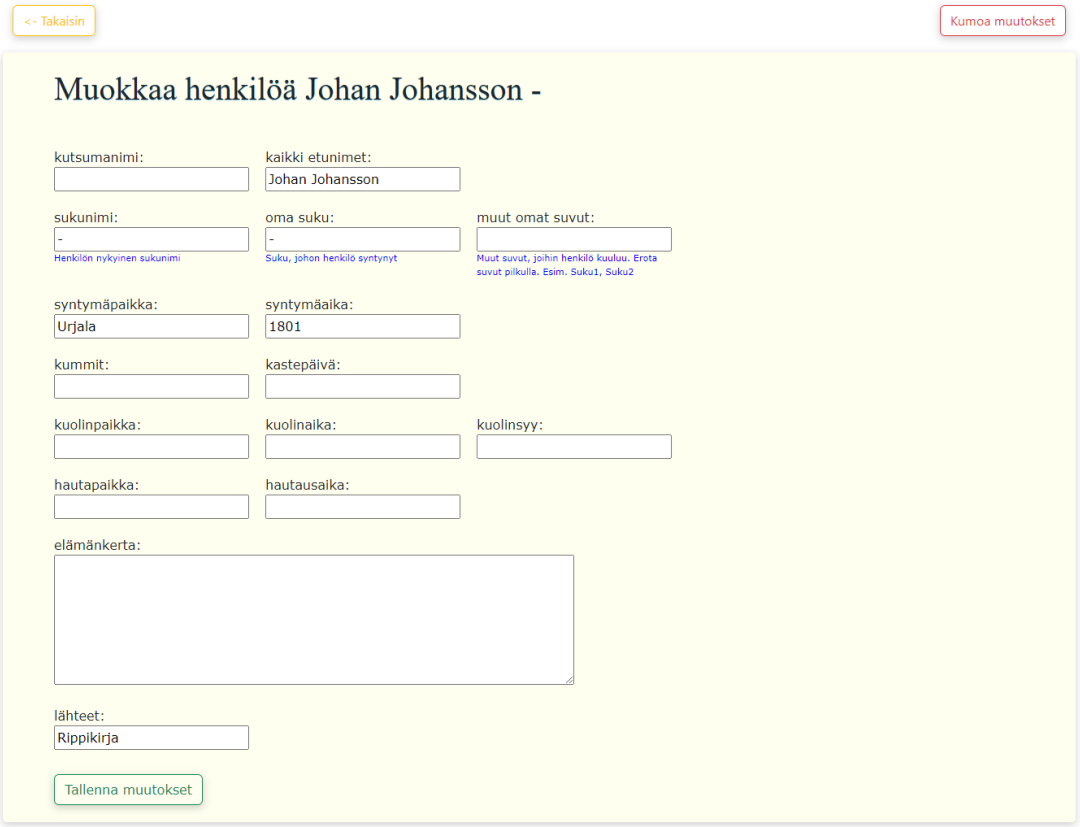
Kuva 26. Lisää uusi henkilö.



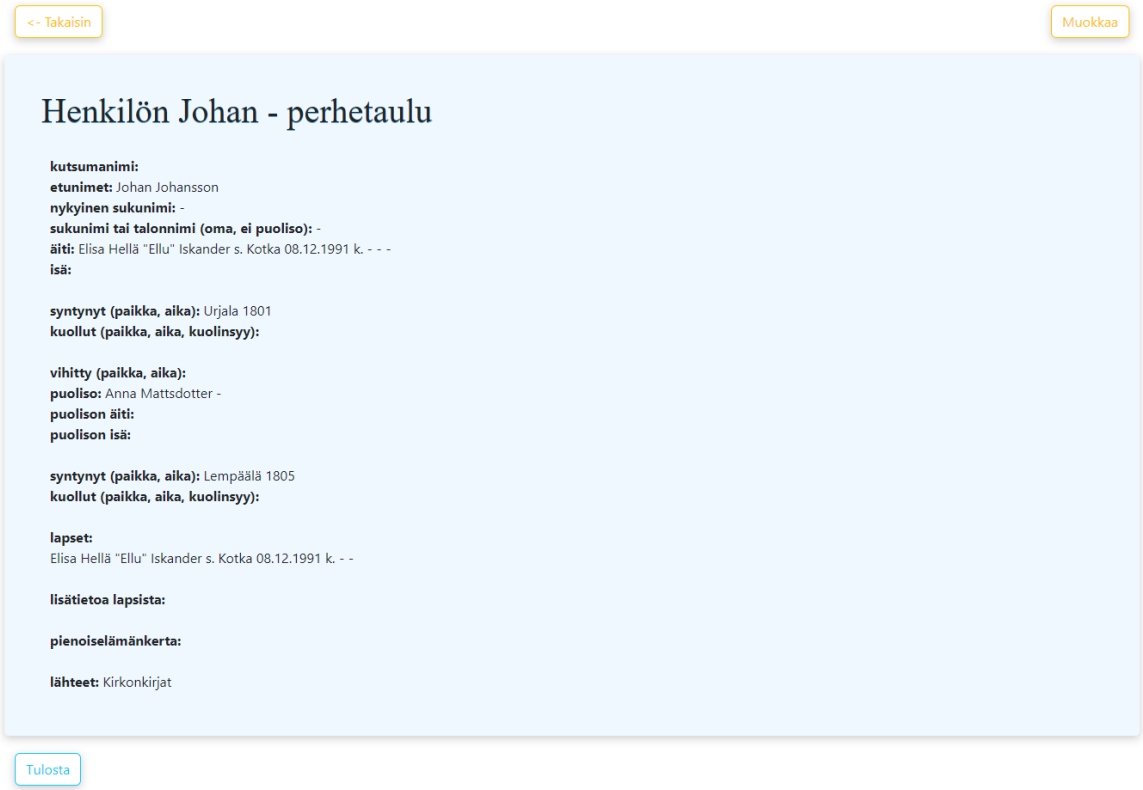
Kuva 27. Lisää uusi perhetaulu.



Kuva 28. Henkilön Johan Johansson -henkilötietojen muokkaus.



Kuva 29. Henkilön Johan -perhetaulu.

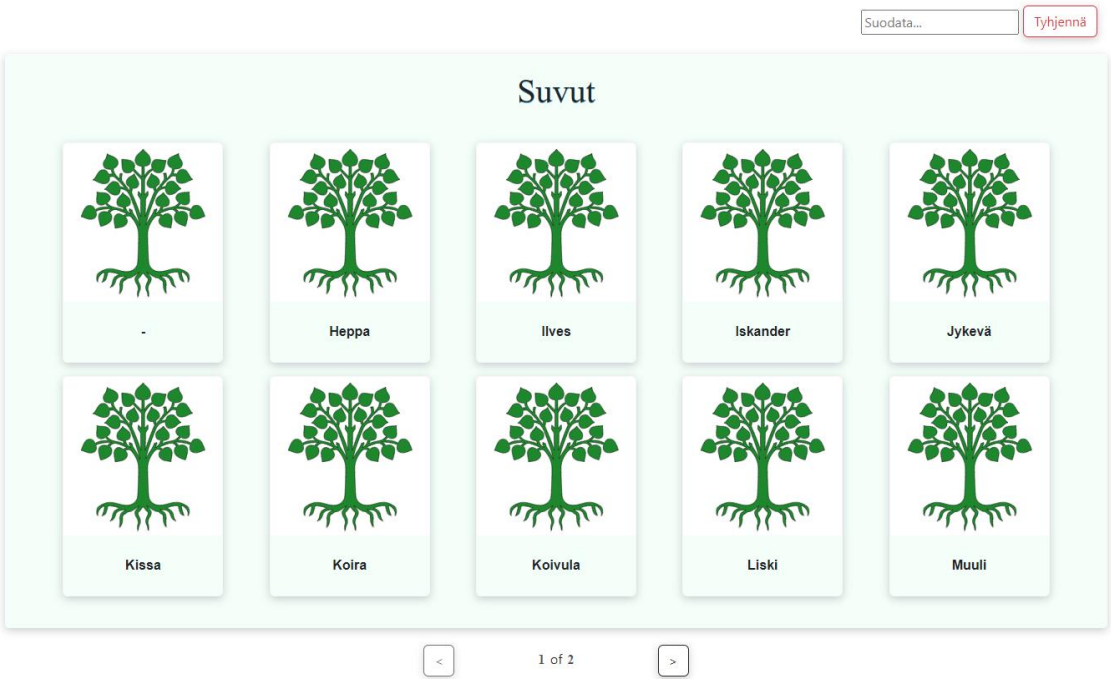


Kuva 30. Henkilön Johan -henkilötiedot.

Sovellus hakee automaattisesti henkilöiden sukutiedot henkilöiden tiedoista. Henkilö voi kuulua kerralla moneen eri sukuun riippuen siitä, mitä sukuja henkilölle on annettuna. Valitun suvun jäsenet näkyvät listauksena samalla tavalla kuin henkilö ja perhetaulut (kuva

32), mutta ilman poistomahdollisuutta. Lisäksi, jos suvussa on myös esimerkiksi ei-sukuun-syntyneitä henkilöitä, niin nämä voi piilottaa tai näyttää halutessaan suvun jäsenten listauksessa.

Kuva 31. Sukujen listaus.



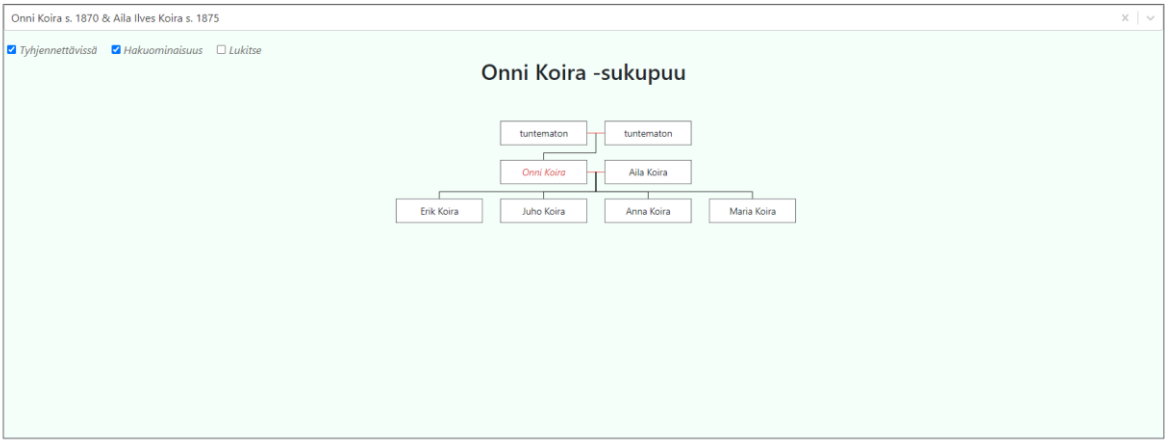
Kuva 32. Valitun suvun jäsenet.

Sukupuu-sivulla pystyy näyttämään tällä hetkellä halutun perhetaulun mukaisen

sukupuukaavion (kuva 33). Sivulta löytyy valikko, jossa on listattuna kaikki sovelluksesta löytyvät perhetaulut. Valinnassa näkyy ensimmäisenä kenen perhetaulu on kyseessä ja tämän henkilön syntymävuosi. Tämän lisäksi valinnassa näkyy henkilön puoliso syntyvuoden kera. Valikko on toteutettu niin, että perhetaulun voi valita joko kirjoittamalla tai selaamalla vaihtoehtoja lävitse.

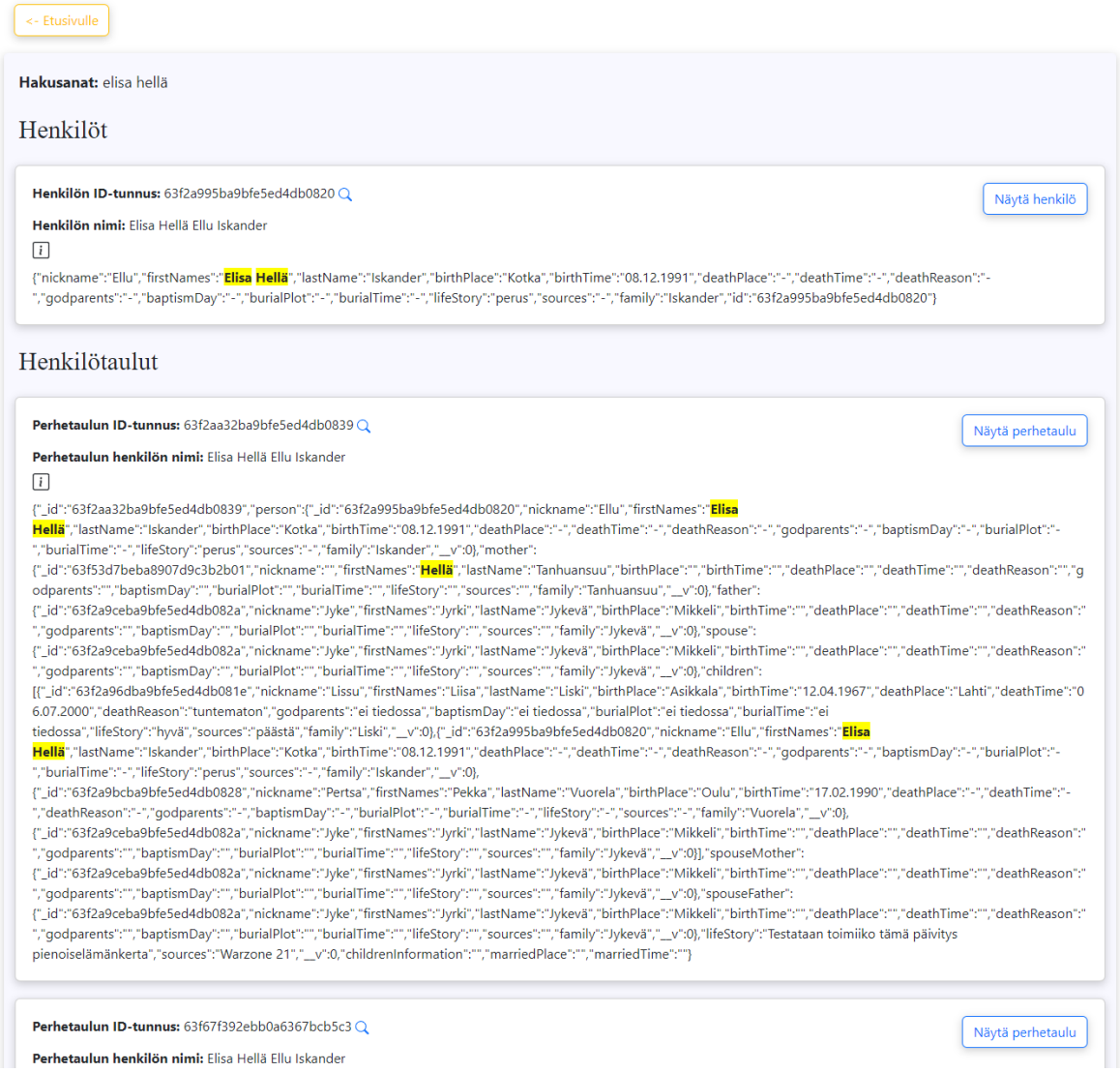
Kun perhetaulu on valittuna, niin näytetään kenen sukupuu on kyseessä. Sukupuukaaviossa näkyy henkilön vanhemmat, puoliso sekä tämän puolison kanssa tehdyt ja/tai hankitut lapset. Sukupuukaavio on tällä hetkellä vielä osittain mietinnän alla ja kaavion logiikka saattaa muuttua niin, että siihen yhdisteltäisiin yhden henkilön kaikki perhetaulut ja näytettäisiin samalla hieman enemmän tietoa.

Kuva 33. Sukupuukaavio.



Sovelluksen etsintätoiminnallisuus on toteutettu niin, että etsitään sekä henkilöistä että perhetauluista annetulla hakusanalla tai -sanoilla (kuva 34). Sovellus näyttää kaikki hakutulokset JSON-muodossa, jossa se korostaa keltaisella kaikki hakusanaan täsmäävät sanat. Hakutuloksiin on lisätty myös [i]-toiminnallisuus, jonka kautta JSON-muotoista dataa pystyy tarkastelemaan selkomuodossa. Lisäksi suurennuslasi-ikonin kautta pystyy hakemaan suoraan joko kyseisen henkilön tai perhetaulun ID-arvolla kaikki henkilö- ja perhetaulutiedot, joissa tämä kyseinen ID-arvo esiintyy.

Kuva 34. Sovelluksen etsintätoiminnallisuuden hakutuloksia.



5 Yhteenveto

Sovellus valmistui halutulla tavalla ja on valmis käytettäväksi sellaisenaan. Sovellus pyörii DigitalOceanin tarjoamassa pilvessä ja sitä pääsee käyttämään suoraan nettiselaimesta. Tulostusominaisuus jäi vielä sovelluksesta puuttumaan, mutta se on seuraavaksi työnalla. Kyseisen ominaisuuden kanssa pitää vielä konsultoida käyttäjää, minkälaisen tulostuksen tai tulostusnäkymän hän haluaa paperille tulostuvan. Jatkokehitystyö jatkuu sitä mukaan, kun sellaiselle tulee tarvetta esimerkiksi käyttäjän toiveesta tai, jos sovelluksessa ilmenee esimerkiksi virheitä. Jatkokehityksen kohteena ovat tällä hetkellä aiemmin mainittu tulostusominaisuus, eri laitteisiin soveltuvan responsiivisuuden lisääminen ja hakutulosten näkymän muokkaaminen. Nämä eivät kuitenkaan ole kiireellisiä tai kriittisiä, ja sovellusta ei

tulla tällä hetkellä todennäköisesti käyttämään esimerkiksi mobiililaitteilla. Sovelluksen URL tullaan vaihtamaan nykyisestä URL:sta toiseen ja sille tullaan hankkimaan myös domain. Sovelluksen tietokanta tullaan työn valmistumisen jälkeen resetoimaan ja sinne pääsee syöttämään oikeaa dataa. Ylläpito on automatisoitu siten, että aina kun koodiin tekee muutoksia ja sen puskee GitHubiin, niin sovellus päivittää automaattisesti muutokset pilveen ja näin ollen sovellukseen.

Tulevaisuudessa sovellusta pystyy halutessaan laajentamaan myös muille käyttäjille, mutta silloin pitää muokata myös hieman esimerkiksi tietokantaan liittyvää toiminnallisuutta. Tällöin myös pitää miettiä antaako käyttäjälle mahdollisuuden rekisteröityä itsenäisesti sovellukseen. Suorituskyvyn ei pitäisi olla tällä hetkellä haasteena, mutta jos tällaista ilmenee, niin silloin pitää kiinnittää myös hieman enemmän huomiota tähän asiaan ja pyrkiä optimoimaan sovellusta enemmän esimerkiksi siirtämällä enemmän toiminnallisuuksia kuten suodatusta backendin puolelle. Tietoturvan pitäisi olla hyvällä mallilla, eikä sovellukseen tallenneta loppupeleissä mitään arkaluontoista, mutta tätäkin on aina hyvä kehittää lisää sekä ottaa asia huomioon kaikessa.

Lähteet

Anderson, BJ. (n.d.). *JavaScript Models.* https://bjanderson.github.io/js-models/

Angular. (28.2.2022). *What is Angular?* Haettu 8.5.2023 osoitteesta https://angular.io/guide/what-is-angular

Arsenault, C. (23.2.2017). *Backend vs Frontend Web Development – Exploring Both Sides.* https://www.keycdn.com/blog/backend-vs-frontend

Drake, M. (14.12.2020). *What is MySQL?* Haettu 8.5.2023 osoitteesta https://www.digitalocean.com/community/tutorials/what-is-mysql

Expo. (n.d.). *Expo Asset.* Haettu 8.5.2023 osoitteesta https://docs.expo.dev/versions/latest/sdk/asset/

Francois, A. (2.2.2022). *MariaDB vs. MongoDB – Which One Should I Choose?* https://www.alibabacloud.com/blog/mariadb-vs--mongodb-which-one-should-i-choose_598549

Full stack open. (n.d.-a). *Same origin policy ja CORS.* Helsingin yliopisto. Haettu 8.5.2023 osoitteesta https://fullstackopen.com/osa3/sovellus_internetiin#same-origin-policy-ja-cors

Full stack open. (n.d.-b). *Tietojen tallentaminen MongoDB-tietokantaan.* Helsingin yliopisto. Haettu 8.5.2023 osoitteesta https://fullstackopen.com/osa3/tietojen_tallettaminen_mongo_db_tietokantaan#mongo-db

Gathoni, M. (13.4.2023). *How to Hash and Verify a Password in Node.js With bcrypt.* Haettu 8.5.2023 osoitteesta https://www.makeuseof.com/nodejs-bcrypt-hash-verify-salt-password/

Geeksforgeeks. (23.5.2020). *Difference between SQLite and MongoDB.* Haettu osoitteesta 8.5.2023 https://www.geeksforgeeks.org/difference-between-sqlite-and-mongodb/

Git for Windows. (n.d.). *Tools & Features*. https://gitforwindows.org/

GitHub. (n.d.). *Hello World*. Haettu 8.5.2023 osoitteesta https://docs.github.com/en/get-started/quickstart/hello-world

Gärtner, E. (n.d.). *d3-dTree*. https://www.jsdelivr.com/package/npm/d3-dtree

Javapoint. (n.d.). *Java Tutorial*. Haettu 8.5.2023 osoitteesta https://www.javatpoint.com/java-tutorial

KnowledgeHut. (1.5.2023). *How To Use Axios with React?* https://www.knowledgehut.com/blog/web-development/axios-in-react

Lodash. (n.d.). *A modern JavaScript utility library delivering modularity, performance & extras.* https://lodash.com/

Lutkevich, B. (2023). *GitHub.* Haettu 8.5.2023 osoitteesta https://www.techtarget.com/searchitoperations/definition/GitHub

Marijan, B. (n.d.). *What is Git Bash; Working with Git Bash Commands.* https://phoenixnap.com/kb/what-is-git-bash

MDN Web Docs. (10.4.2023). *Cross-Origin Resource Sharing (CORS).* Haettu 8.5.2023 osoitteesta https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

MongoDB. (n.d.). *Why Use MongoDB and When to Use it?* Haettu 8.5.2023 osoitteesta https://www.mongodb.com/why-use-mongodb

Nodejs. (n.d.). *Node.js is an open-source, cross-platform JavaScript runtime enviroment*. Haettu 8.5.2023 osoitteesta https://nodejs.org/en

Npm. (17.4.2023). *Mongoose*. https://www.npmjs.com/package/mongoose

O'Reilly. (n.d.). *Using Nodemon.* https://www.oreilly.com/library/view/server-side-development/9781789345391/e2ac19e3-1bb0-4a10-8baf-3ea96b77c5b2.xhtml

Perforce. (4.2.2022). *What Is DigitalOcean?* https://www.perforce.com/blog/vcs/what-is-digital-ocean

Postman. (n.d.). *Tools*. Haettu 8.5.2023 osoitteesta https://www.postman.com/product/tools/

Ravikiran A S. (6.2.2023). *How to use Node Js for Backend Web Development in 2023.* Haettu 8.5.2023 osoitteesta https://www.simplilearn.com/tutorials/nodejs-tutorial/nodejs-backend

React. (n.d.-a). *Components and Props.* https://legacy.reactjs.org/docs/components-and-props.html

React. (n.d.-b). *The library for web and native user interfaces.* Haettu 8.5.2023 osoitteesta https://react.dev/

React D3 Library. (n.d.). *A JavaScript library that allows developers the ability to use D3 in React*. https://react-d3-library.github.io/

Sharma, A. (10.4.2023). *What Is Express JS In Node JS?* Haettu 8.5.2023 osoitteesta https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js

Stork, V. (25.10.2021). *How to Use Node Environment Variables with a DotEnv File for Node.js and npm.* https://www.freecodecamp.org/news/how-to-use-node-environment-variables-with-a-dotenv-file-for-node-js-and-npm/

Ström, C. (9.7.2019). *React pähkinänkuoressa.* https://joinex.fi/react-pahkinankuoressa/

Sufiyan, T. (10.2.2023). *Understanding JWT Authentication With Node.js.* Haettu 8.5.2023 osoitteesta https://www.simplilearn.com/tutorials/nodejs-tutorial/jwt-authentication

Swiatkowski, J. (24.6.2022). *What Is Postman, What Is It Used for, and What are Its Functionalities?* https://www.droptica.com/blog/what-postman-what-it-used-and-what-are-its-functionalities/

Tecci. (9.9.2020). *React – Frontendin kehityksen Tetris!* https://tecci.fi/2020/09/09/react-frontend-kehityksen-tetris/

Technostacks. (12.1.2023). *17 Best JavaScript Frameworks for Robust Web Development.* https://technostacks.com/blog/best-javascript-frameworks/

Tuura, T. (31.3.2015). *Node.js:n käyttö JavaScriptin tulkkaukseen.* Haettu 8.5.2023 osoitteesta https://www.cs.helsinki.fi/u/ttuura/otk-js/nodeshell.html

Visual Studio Code. (30.3.2023). *Why did we build Visual Studio Code?* Haettu 8.5.2023 osoitteesta https://code.visualstudio.com/docs/editor/whyvscode

Vue.js. (n.d.). *What is Vue?* Haettu 8.5.2023 osoitteesta https://vuejs.org/guide/introduction.html

W3Schools. (n.d.-a). *C# Tutorial.* Haettu 8.5.2023 osoitteesta https://www.w3schools.com/cs/index.php

W3Schools. (n.d.-b). *Java Introduction.* Haettu 8.5.2023 osoitteesta https://www.w3schools.com/java/java_intro.asp

W3Schools. (n.d. c). *W3.JS Controllers.* Haettu 8.5.2023 osoitteesta https://www.w3schools.com/w3js/w3js_controllers.asp

Zola, A. (2022). *Bootstrap.* Haettu 8.5.2023 osoitteesta https://www.techtarget.com/whatis/definition/bootstrap