Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

TOMMI VIRTANEN

Verkkokäyttöinen hautakuvasovellus

TIETOJENKÄSITTELYN TUTKINTO-OHJELMA
2023

TIIVISTELMÄ

Virtanen, Tommi: Verkkokäyttöinen hautakuvasovellus
Opinnäytetyö, AMK
Tutkinto-ohjelma: Tietojenkäsittely
Huhtikuu 2023
Sivumäärä: 38

Opinnäytetyön ideana oli kehittää verkkokäyttöinen sovellus käyttäen ReactJS-käyttöliittymäkirjastoa, .NET Corea ja SQL Serveriä.

Työn toimeksiantaja on Hakosalo Innovations Oy ja työn loppukäyttäjä on yrityksen asiakkaana oleva yhdistys. Yhdistys pidetään tässä työssä nimettömänä.

Työssä suunniteltiin ja toteutettiin selaimella käytettävä prototyyppimalli sovelluksesta, jonka avulla voi kuvata hautoja ja ladata kuvat palveluun muiden käyttäjien selattaviksi. Opinnäytetyön aikataulusta johtuen sovellusta ei ehditty toteuttaa loppuun asti, joten työssä kuvataan vaan työn toteutuksessa käydään läpi vain valmiiksi saadut aiheet.

Avainsanat: mobiilisovellus, verkko-ohjelmointi, ReactJS, .NET Core, SQL Server.

Abstract

Virtanen, Tommi: Web based application for gravepictures
Bachelor's thesis
Degree programme: Business Information Systems
April 2023
Number of pages: 38

The goal of this thesis was to develop a web based application using ReactJS, .NET Core and SQL Server.

The client of this thesis was Hakosalo Innovations Oy and the software was made for an anonymous third party customer of the client company. The third party client is kept anonymous in this thesis.

In the thesis the applications requirements were specified and the application was developed. Due to time restrictions on this thesis the application itself wasn't fully completed, so the thesis only covers parts that were completed.

Keywords: mobile application, web development, ReactJS, .NET Core, SQL Server.

LYHENNELUETTELO

C#	Olioihin perustuva ohjelmointikieli
HTML	Standardoitu merkintäkieli selaimessa esitettäville dokumenteille (sivuille).
ReactJS	JavaScript käyttöliittymäkirjasto
.NET Core	Microsoftin kehittämä palvelinohjelmistokehys
JavaScript	Verkkopalveluiden yleisesti käyttämä ohjelmointikieli
API	Application Programming Interface, eli erillinen sovellus palvelimella, joka lähettää käyttöliittymälle tietoa.
REST	Representational state transfer eli rajapintojen toteuttamiseen tarkoitettu arkkitehtuurityyli
Client	Käyttäjän laite, joka suorittaa selaimen lähettämän käyttöliittymäohjelman koodin.
NodeJS	Alusta JavaScript-koodin suorittamiseen verkkoselaimen ulkopuolella
NPM	Node Package Manager on ohjelmisto, jonka kautta voi ladata JavaScript-lisäosia NodeJS alustalla suoritettaviin JavaScript-ohjelmiin.
Vite	Kehitysalusta React- ja Vue-projekteille.

SISÄLLYS

1 JOHDANTO

Opinnäytetyössä käsitellään mobiilikäyttäjille suunnatun verkkopalvelun kehitystä teoriassa, esitellään teknologiset ratkaisut sekä perustellaan ne ja esitetään sovellukselle jo suunniteltua jatkokehitystä sekä erilaiset käyttömahdollisuudet tulevaisuudessa. Opinnäytetyössä käsitellään enimmäkseen sovelluksen käyttöliittymää.

Internetiä käytetään suurenevissa määrin mobiililaitteella. Vuoden 2022 toisella neljänneksellä noin 59% verkon selaajista käytti mobiililaitetta. Kasvu on vuodesta 2015 ollut nopeaa ja mobiilikäyttäjien määrä on lähes tuplaantunut vuosien 2015 – 2022 aikana. (Statista, 2022.) Mobiilikäyttöisyyden lisääntyminen on tuonut uusia vaatimuksia verkkosivujen kehitykseen. Kehittäjien on otettava entistä enemmän huomioon erilaiset käyttäjäryhmät ja niille suunnattu sovellusten käytettävyys. Verkkosivuja kehitettäessä on tehokkaampaa aloittaakin mobiililaitteille suunnittelusta niiden rajoitetun ruudun koon ja pikselien määrän takia. Mobiilinäkymästä aloitettaessa kehittäminen on nopeampaa isommille ruudunkoille, koska kaikki data, joka mahtuu mobiililaitteen ruudulle, mahtuu myös isolle ruudulle.

Nykyaikaisten selainten suorituskyvyn kehittyminen on mahdollistanut selainpohjaisten sovellusten pääsyn mobiililaitteen käyttöjärjestelmätason ominaisuuksiin, kuten kameraan, paikannukseen ja yhteystietoihin. Kehitys on mahdollistanut monipuolisten applikaatioiden toimittamisen käyttäjille ilman erillisen sovelluksen latausta käyttäjän laitteelle. Sovelluskehittäjät ovatkin aiemmin olleet pakotettuja kirjoittamaan saman sovelluksen kaikille alustoille omana koodinaan, mutta nykypäivänä saman koodin voi kirjoittaa verkkopohjaiseksi yhteen sovellukseen.

2 OPINNÄYTETYÖN ESITTELY

Opinnäytetyön ideana on tuottaa verkkoon palvelu hautojen kuvaamiseen. Sovellus toteutetaan Hakosalo Innovationsin toimesta nimettömälle asiakasyhdistykselle ja sen kehitystä dokumentoidaan sekä seminaarityöhön että opinnäytetyöhön. Projekti julkaistaan verkkoon toistaiseksi tuntemattomaan osoitteeseen ja kehittämisvaiheessa ohjelma toteutetaan käyttäen paikallista tietokonetta, jolloin selaimessa käytetään URL-osoitetta http://localhost. Projektin jälkeen sovelluksen kehitystä on tarkoitus jatkaa asiakkaan toiveiden mukaan.

2.1 Hakosalo Innovations

Hakosalo Innovations Oy on porilainen ohjelmistoalan yritys, joka tuottaa verkkopohjaisia järjestelmiä. Yritys on perustettu vuonna 2019 yritysjakautumisen johdosta, mutta alkuperäinen yritys perustettiin vuonna 1995. Yrityksen henkilöstöön kuuluu työn kirjoitushetkellä 4 henkilöä, kirjoittaja mukaan lukien. Yrityksen tuoteportfoliosta löytyy sekä räätälöityjä, asiakkaan tilaamia tuotteita, että tuotteistettuja sovelluksia, kuten CMS-järjestelmä.

2.2 Käyttötarkoitus

Sovellus tuotetaan suomalaiselle sukututkimusta edistävälle yhdistykselle. Sukututkimuksella tarkoitetaan sukujen ja niiden historian tutkimusta käyttäen lähdemateriaalia sukujen menneisyydestä. Yhdistys pyrkii edistämään suomalaista sukututkimusta lisäämällä sen tuntemusta ja tuloksia. Yhdistys ei varsinaisesti itse tee sukututkimusta, vaan sen tehtävä on tarjota sukututkijoille mahdollisimman laaja kattaus lähdemateriaalia ja aineistoja sukututkimuksen välineiksi. Yksi tällainen lähdeaineisto on haudat ja hautakuvat.

Hautakuvasovelluksen tarkoitus on ensisijaisesti yksinkertaistaa ja nopeuttaa hautatietokannan täydentämistä. Yhdistyksen nykyisessä järjestelmässä hautatiedot lähetetään yksitellen Excel-tiedostona järjestelmän ylläpitäjälle, joka lisää tiedot sukututkimusjärjestelmään. Uudessa sovelluksessa sukututkijat voivat lisätä järjestelmään kuvia jo olemassaolevista haudoista, sekä lisätä kokonaan uusia henkilöitä hautakuvan lisäämisen yhteydessä, mikäli hautaa ei järjestelmässä vielä ole. Sovellus nopeuttaa ja helpottaa kuvien lisäämistä palveluun automatisoimalla suuren osan prosessista. Käyttäjä lisää kuvia sovellukseen, jolloin ne siirtyvät palveluun odottamaan ylläpitäjän hyväksyntää. Järjestelmä ei automatisoi koko prosessia, sillä kuvien auditointi eli laaduntarkastaminen vaatii manuaalista työtä. Sovellus helpottaa kuvien lisäämistä, jolloin sukututkijoiden on entistä helpompi lisätä kuvia järjestelmään madaltaen näin kynnystä kuvien lataamiselle.

2.3 Vaatimusmäärittely

Sovelluksen kehittämistä aloittaessa tehdään aina vaatimusmäärittely. Tässä sovelluksessa sellainen tehtiin Hakosalo Innovationsin toimesta.

Vaatimusmäärittely on kommunikaatioväline tilaajan ja asiakkaan välille. Sen avulla toimittaja saa tiedon siitä, minkälainen sovelluksesta on tarkoitus tulla. Toimittaja pystyy vaatimusmäärittelyn perusteella luomaan aikataulun ja kustannusarvion projektin kehittämiseen, mikäli vaatimusmäärittely on hyvin tehty. Määrittelyyn kuuluu tyypillisesti ainakin tavoitteet, lähtökohdat sekä toiminnallisuus ja tekniset vaatimukset. (Web-ostajan opas, N.d.)

Hautakuvasovelluksen vaatimusmäärittely tehtiin sisäisesti toimittajan toimesta. Sovelluksen ollessa yrityksen tarjoama prototyyppi, yritys myös määrittelee itse prototyypin ominaisuudet, vaatimukset ja tekniset ratkaisut. Hautakuvasovelluksen oleellisimmat vaatimukset ovat

1. täysi käytettävyys millä tahansa laitteella
2. helppokäyttöinen käyttöliittymä
3. ohjelman nopeat latausajat.

Yhdistyksen hautatietokanta on kaikille avoin, mutta palvelun käyttäjät ovat yhdistyksen jäseniä. Kuka tahansa voi selata hautoja, mutta vain käyttäjät voivat lisätä hautoja. Näin ollen sovelluksen samanaikaiset käyttäjämäärät pysyvät maltillisena, eikä sovellukseen siksi tarvita skaalautuvuutta suurelle samanaikaiselle käyttäjämäärälle ainakaan prototyyppimalliin. Hautakuvasovelluksen toteutuksen on tarkoitus olla valmiina kesään 2023 mennessä. Tällöin sovelluksesta on tarkoitus julkaista ensimmäinen versio asiakkaan testikäyttöön.

3 TEKNOLOGISET VALINNAT

Verkkosovelluksen suorituskyvyllä tarkoitetaan asioita, joilla tehdään verkkosivusta nopea ja joilla saadaan hitaat asiat tuntumaan nopeammilta. Suorituskyvyllä tarkoitetaan käyttäjän kokemusta palvelusta ja sen nopeudesta. Palvelun suorituskyky luokitellaan yleensä latausajan, toiminnallisuuden nopeuden ja sulavuuden mukaan. (MDN Web Docs, n.d)

Sovelluksen teknologiset valinnat tehtiin osittain arkkitehtuurin ja työn laajuuden sanomina. Esimerkiksi sovelluksen SEO eli Search Engine Optimization jätettiin pois harkinnasta, sillä sovelluksella ei ole varsinaista tarvetta hakukoneoptimoinnille. Varsinaisessa tilaustuotteessa SEO tullaan ottamaan huomioon. Helppokäyttöinen väline sivun nopeuden, käytettävyyden ja SEO:n mittaamiseen on avoimen lähdekoodin Google Lighthouse. Lighthouse-auditoinnin voi suorittaa millä tahansa verkkosivulla. Lighthouse

mittaa muun muassa suorituskykyä, käytettävyyttä ja SEO:ta. (Chrome Developers, 2022.)

3.1 ReactJS

ReactJS on verkkopohjaisten sovellusten tekemiseen tarkoitettu käyttöliittymäkirjasto. React mahdollistaa reaktiivisten verkkosivujen tuottamisen käyttäen JavaScriptiä, HTML:ää (Hypertext Markup Language) ja CSS-tekniikoita. React käyttää JavaScriptin jatkettua syntaksia nimeltään JSX renderöidäkseen HTML-koodia samassa yhteydessä normaalin JavaScriptin kanssa.

React-käyttöliittymät jaetaan komponentteihin. Komponentit ovat itsenäisiä osia, jotka palauttavat tietyn sivulle renderöitävän HTML-koodin ja siihen liitetyn JavaScript-logiikan. React-komponentit ovat JavaScript-funktioita, joilla on mahdollisuus ottaa vastaan parametreja kutsuttaessa. React-komponentteja voi tehdä kahdella eri tavalla, käyttäen luokkaa tai määritellen komponentin funktioksi. Reactin oma dokumentaatio suosittelee kaikkia uusia React-sovelluksia käyttämään funktiokomponentteja, joten myös tässä työssä käytetään funktiokomponentteja. (React documentation, n.d.)

Reactiin on Node-pohjaisuutensa johdosta mahdollista lisätä toiminnallisuutta ja ominaisuuksia käyttäen npm-paketteja. Tällaisia toiminnallisuuksia ovat esimerkiksi tyylit ja valmiit komponentit. Npm-lisäosat lisäävät sovelluksen kokoa, jolla on vaikutuksia sovelluksen latausaikoihin sekä suorituskykyyn.

Reactin valintaa sovelluksen käyttöliittymäksi tuki kirjoittajan kiinnostus Reactiin ja suuri ekosysteemi, joka takaa Reactille hyvät jatkokehitysmahdollisuudet ja hyvän yhteisön tuen. Sen moderni ja nopea kehitys mahdollistavat käyttöliittymän tehokkaan ja nopean kehityksen.

3.2 Material UI

Material UI on React-komponenttikirjasto, joka tarjoaa Googlen Material Design -ohjeistukseen pohjautuvia käyttöliittymäkomponentteja, kuten näppäimiä, liukusäätimiä ja tekstikenttiä. Material UI -komponentit ovat tuotantovalmiita tehdasasetuksiltaan, joten kehittäjän tehtäväksi jää liittää haluttu toiminnallisuus kuhunkin komponenttiin. (Material UI, 2023.)

Material UI sopii sovellukseen helppokäyttöisten komponenttiensa takia. Material UI vähentää kehittäjän työtä, sillä CSS-luokkien kirjoittamiseen kuluu vähemmän aikaa, jolloin kehittäjä voi keskittyä ”liiketoimintalogiikan” tuottamiseen. Material UI:ta pyritään käyttämään sovelluksessa nimenomaan sellaisissa komponenteissa, jotka toistavat itseään. Sen sijaan koko sovelluksen ulkoasua määrittelevät komponentit tehdään käyttäen CSS-kieltä.

3.3 .NET Core

.NET Core on Microsoftin kehittämä ja ylläpitämä avoimen lähdekoodin ”cross-platform” ohjelmistokehys, joka mahdollistaa .NET-sovellusten suorittamisen eri alustoilla, kuten Windowsilla ja Linuxilla. .NET Core:a on mahdollista käyttää monen eri tyyppisen ohjelmiston tai niiden osien kehittämisessä. .Net Core -sovellukset suoritetaan .NET Framework -ohjelmistokehykseen kuuluvan CLR-komponentin alaisuudessa. CLR mahdollistaa .NET -sovelluksissa muun muassa automaattisen muistinhallinnan, tyyppivarmuuden ja usean alustan tuen. (Microsoft Learn, 2023.)

3.4 SQL Server

SQL Server on Microsoftin kehittämä ja ylläpitämä SQL-relaatiotietokannan hallintajärjestelmä. SQL Serveriä käytetään SQL-kielellä ja sen sisältämien tietokantojen data on rakenteellista.
SQL Server on työssä käytössä siksi, että aiemmat sovelluksen versiot on rakennettu sitä käyttäen, joten tarvetta muutokselle ei ollut.

3.5 Microsoft IIS

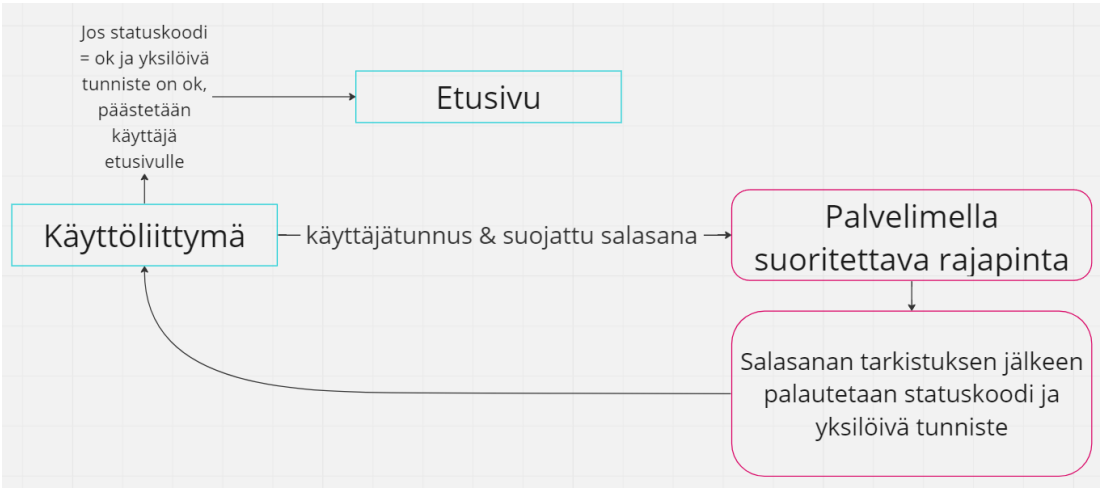
Internet Information Services eli IIS on Microsoftin palvelinalusta verkkopalveluiden isännöintiä varten. (Microsoft, N.d). IIS:n avulla pystyy myös ohjaamaan palvelimen kuormitusta sekä ohjaamaan IP-osoitteita. IIS:ää käyttäen on mahdollista suorittaa natiivisti Microsoftin omia tuotteita, kuten .NET-palveluita, C#- ja VBScript-ohjelmointikieliä. Muita alustalla suoritettavissa olevia kieliä ovat mm. PHP ja Java.

4 SOVELLUKSEN KOOSTUMUS

Hautakuvasovellus koostuu sivuista, joille käyttäjä navigoidaan URL-osoitteen perusteella. Sivut ovat komponentteja, jotka sisältävät toisia komponentteja. Kun käyttäjä navigoi osoitteeseen https://localhost:3000/, käyttäjä ohjataan etusivu-komponenttiin, joka koostuu automaattisesti renderöitävästä navigaatiopalkista sekä komponenttikohtaisesta koodista. Uudelleenkäytettävät komponentit ja sivut sijaitsevat omissa kansioissaan.

4.1 Arkkitehtuuri

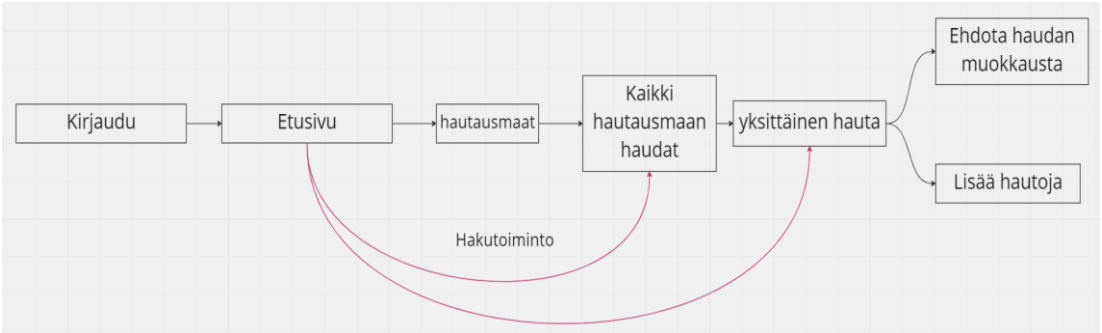
Sovelluksen toteutus perustuu REST API -rajapintaan ja käyttäjän laitteella suoritettavaan client-ohjelmaan. Käyttäjän navigoidessa palvelun etusivulle React lataa käyttäjän laitteelle sovelluksen, jonka navigointi toteutuu sovelluksen sisäisesti. Datan ohjelma hakee REST API:sta käyttäen HTTP-protokollan GET-metodia. Kuvassa 1 kuvataan yksinkertaistettu kaavio sovelluksen kirjautumisesta käyttäen API-kutsua.



Kuva 1. Yksinkertainen API-kutsun mallinnus

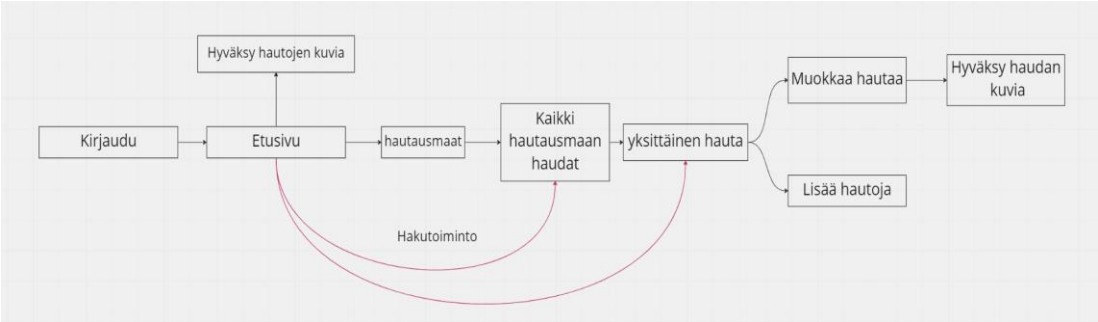
4.2 Sivukartta

Sovelluksen sisältö määritellään kahdeksi eri kategoriaksi, peruskäyttäjän ja hallintakäyttäjän näkymäksi. Peruskäyttäjä on sisään kirjautunut asiakasyhdistyksen jäsen ja hallintakäyttäjä on sisäänkirjautunut henkilö, jolla on hallinnoijan oikeudet asiakasyhdistyksen järjestelmässä. Sovelluksen käyttö vaatii kirjautumisen sähköpostilla ja salasanalla, joiden oikeellisuus vahvistetaan tietokannasta. Onnistuneen sisäänkirjautumisen jälkeen käyttäjä ohjataan etusivulle. Kuva 2 kuvaa peruskäyttäjälle näkyvää sivurakennetta asiakkaalle esiteltävässä ohjelmassa.



Kuva 2. Peruskäyttäjän sivukartta.

Hallintakäyttäjä on asiakasyrityksen jäsen, jolla on peruskäyttäjää enemmän oikeuksia yrityksen palveluissa. Hallintakäyttäjän sivut erotetaan käyttäjän sivuista omaksi projektikseen. Sivuista tehdään IIS:n päälle oma palvelu, joka pyörii osoitteessa https://hallinta.palvelunosoite.fi. Hallintakäyttäjän näkymä käyttää datan hakemiseen ja päivittämiseen samaa rajapintaa kuin peruskäyttäjän käyttöliittymä, mutta hallintakäyttäjällä on käyttöliittymässä enemmän ominaisuuksia kuin peruskäyttäjällä. Tällaisia ominaisuuksia ovat esimerkiksi kuvien ylläpito, peruskäyttäjien oikeuksien hallinnointi ja erilaisten palveluun liittyvien statistiikkojen näkyvyys. Kuvassa 3 on kuvattuna hallintakäyttäjän sivurakennetta.



Kuva 3. Hallintakäyttäjän sivukartta

4.3 Sivujen reititys

Arkkitehtuurivalintojen takia sovelluksen käyttöliittymää ei piirretä palvelimella, vaan sivun piirto tapahtuu loppukäyttäjän laitteella. Käyttäjä lataa koko sivun laitteelleen ensimmäistä kertaa palveluun saapuessaan, jolloin käytännössä kaikki sivut on jo valmiiksi ladattu käyttäjän laitteella. Tässä vaiheessa käyttöön otetaan React Router. React Router mahdollistaa sivujen reitittämisen käyttäjän laitteella, eli se määrittelee, mille sivulle mikäkin osoite johtaa (React Router, N.d). Kuvassa 4 on yksinkertainen esimerkki React Router -käyttötapauksesta hautakuvasovelluksesta.

```
<>
  <Navbar/>


<BrowserRouter>
    <Routes>

      <Route index element={<Etusivu />} />
      <Route path='/hautausmaat' element={<Hautausmaat />} />
      <Route path='/hautausmaa/:uid' element={<Hautausmaa />} />

    </Routes>
</BrowserRouter>


</>
```

Kuva 4. Yksinkertainen React Router -toteutus.

Hautakuvaussovelluksessa toteutetaan käyttöliittymä siten, että Navbar -komponentti piirretään joka sivulle osoitteesta riippumatta. React Routerin BrowserRouter -komponentin sisällä määritellään Route-komponenteilla reitti, joka url-parametrin perusteella renderöidään (W3Schools, N.d.b). Tämä mahdollistaa sovelluksen piirtämisen käyttäjän laitteella ilman uuden sivun pyytämistä palvelimelta. React Routerin avulla komponenttien piirtäminen jokaiselle sivulle on tehty helpoksi.

4.4 Sivujen data

Hautakuvaussovellus käyttää tietokantaa datan lähteenä vanhan REST-rajapinnan kautta. Jokaista sivua varten on tehty oma REST-rajapinnan palvelu (endpoint), josta voidaan hakea haluttu data sivulle. Näin minimoidaan turhan datan määrä sovelluksessa ja nopeutetaan palvelun toimintavalmiutta ja latausnopeutta myös sellaisissa tilanteissa, joissa verkkoyhteyden nopeus on normaalia hitaampi. Datan haku tapahtuu fetch -funktiolla ja palvelin palauttaa datan JSON-muotoisena (JavaScript Object Notation), jolloin Reactin on helppo tulkita dataa ja tulostaa se sivulle. Kuvassa 5 on esitettynä datan hakeminen rajapinnasta.

```
function Hautausmaat() {

  const [state, setState] = useState();

  useEffect(() => {
    const getData = async () => {
        const data = await (
            await fetch (
                "tästä_osoitteesta_haetaan_dataa"
            )
        ).json();
        setState(data)
    }
    getData()

  }, [])
```

Kuva 5. Datan haku rajapinnasta.

5 TIETOKANTA

Sovelluksen tietokanta toteutetaan käyttäen Microsoftin SQL Serveriä. Relaatiotietokanta mahdolllistaa taulujen väliset relaatiot ja suhteet (viiteavaimet), joita tarvitaan sovelluksen datarakenteen muodostamiseen. Sovelluksen ollessa dynaaminen ja datasta riippuvainen on oleellista tehdä tietokannan tauluista toistensa kanssa yhteenliitettäviä ja normalisoituja, jolloin datan muokkaus ja lisäys on mahdollisimman helppo toteuttaa.

Sovelluksen dataa kerätään neljään eri SQL -tauluun. Tietokantaan lisätyt taulut on kuvattu kuvissa 6, 7, 8 ja 9 SQL Serverin Management Studion Design-työkalussa. Kaikkien taulujen nimet päättyvät ”_demo” päätteeseen, jolloin demonstroidaan se, että kyseessä on demoversiossa käytettävät taulut. Sovellukseen lisätään seuraavat taulut:

- Users_demo
- Hautausmaat_demo
- Haudat_demo
- HautojenHenkilot_demo

10184_HSSQL01\H...Hautausmaat_demo | SQLQuery2.sql - 10...mmi.virta

Column Name	Data Type	Allow Nulls
UID	int	☐
Nimi	varchar(MAX)	☑
Paikkakunta	varchar(MAX)	☑
Maa	varchar(MAX)	☑
Koordinaatit	varchar(MAX)	☑
		☐

Kuva 6. Hautausmaat_demo-taulu

10184_HSSQL01\HA...dbo.Haudat_demo

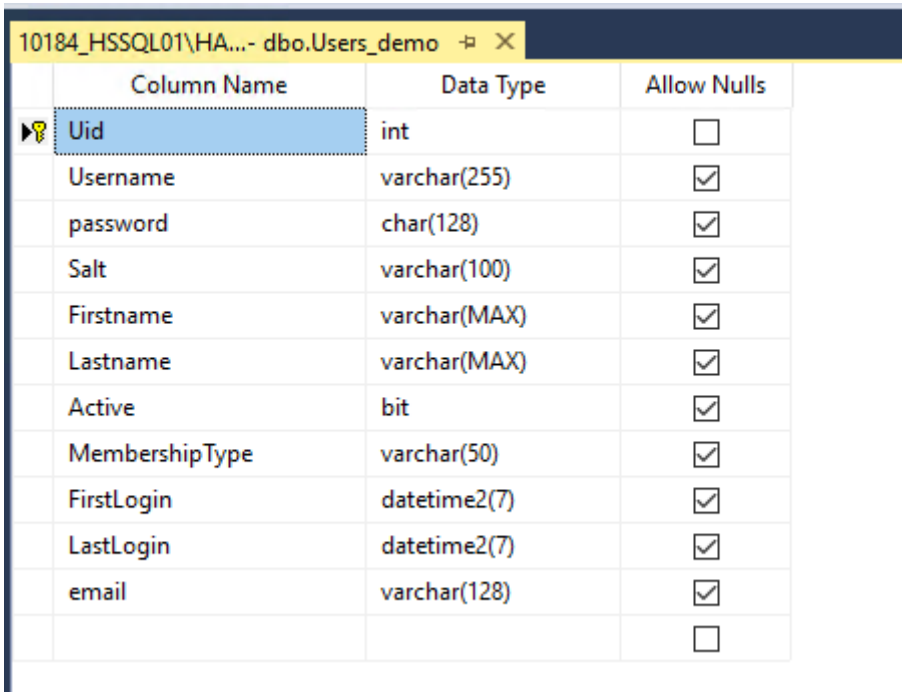
Column Name	Data Type	Allow Nulls
UID	int	☐
ImageName	varchar(255)	☑
HautausmaaUID	int	☑
verifiedImage	bit	☑
		☐

Kuva 7. Haudat_demo-taulu

10184_HSSQL01\HA...jenHenkilot_demo | 10184_HSSQL01\HA...dbo.Hau

Column Name	Data Type	Allow Nulls
UID	int	☐
Etunimi	varchar(MAX)	☑
Sukunimi	varchar(MAX)	☑
BirthDate	datetime2(7)	☑
DeathDate	datetime2(7)	☑
Syntymavuosi	int	☑
Kuolinvuosi	int	☑
HautaUID	int	☑
HautausmaaUID	int	☑
Hautausmaa	varchar(MAX)	☑
		☐

Kuva 8. HautojenHenkilot_demo-taulu



Kuva 9. Users_demo-taulu

6 SOVELLUKSEN TOTEUTUS

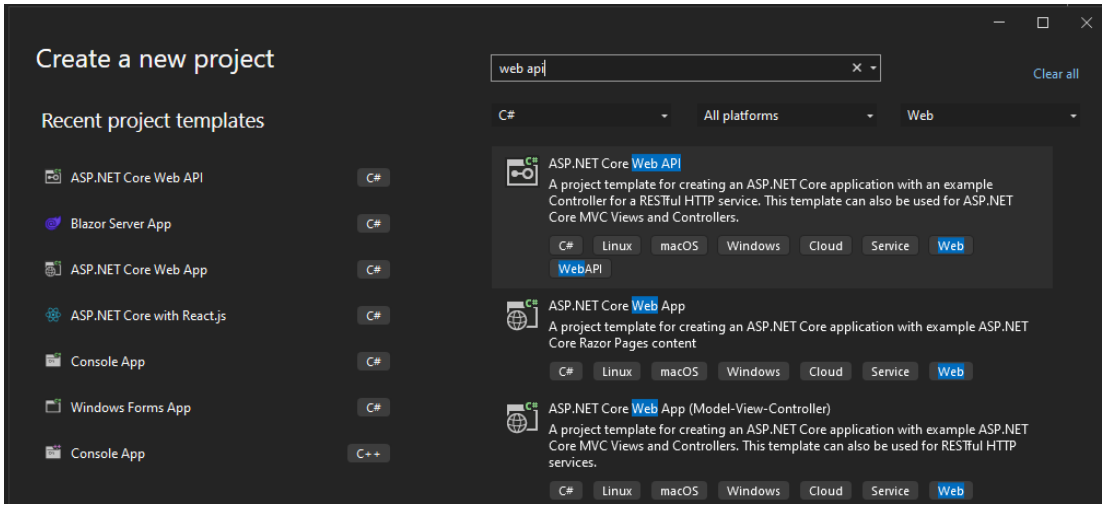
Sovelluksen toteuttamista lähdettiin suorittamaan tässä työssä kuvattujen kaavioiden mukaisesti. Ohjelman tuottaminen aloitettiin palvelinsovelluksesta, sillä näin käyttöjärjestelmän tuottaminen validilla datalla on mahdollista. Tässä työssä käydään läpi oleellisimpien konseptien toteutus, jolla vältetään asioiden toistaminen useaan otteeseen.

6.1 Palvelinohjelma

Sovelluksen toteutuksen ollessa suhteellisen pieni valitaan .NET:n Minimal Web Api rajapintaratkaisuksi. Minimal Web Api on muuten idealtaan sama kuin

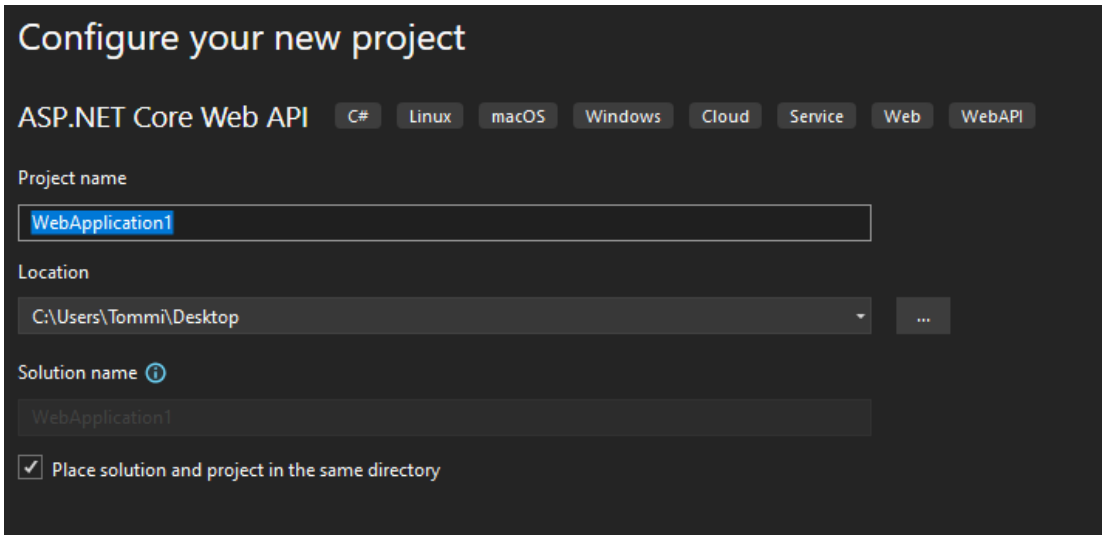
MVC-versiota käyttävä API mutta Minimal-versiossa tietyt MVC:lle tyypilliset ominaisuudet puuttuvat. Minimal Web Api tarjoaa kuitenkin riittävän valikoiman tämän sovelluksen toteutukseen.

Minimal Web Api -sovelluksen voi alustaa käyttäen Visual Studion omaa projektivalikkoa valitsemalla alustettavan projektin tyypiksi ”.NET Core Web Api”. Kuvassa 10 kuvattuna valinta Visual Studio 2022 -ohjelman uutta projektia luotaessa.

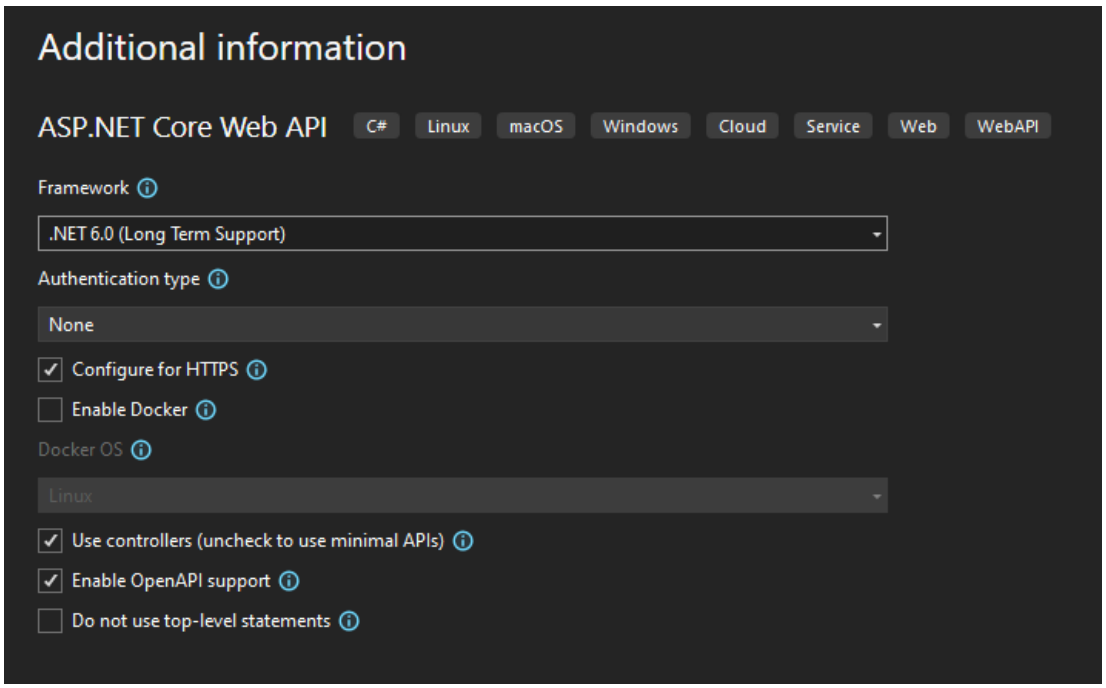


Kuva 10. .NET Core Web Api projektin valinta Visual Studion projektivalikosta.

Projektityypin valinnan jälkeen ohjelma kysyy alustettavan projektin nimeä, sijaintia levyllä ja tallennetaanko ohjelman ”solution” samaan kansioon muun projektin kanssa. Näiden jälkeen Visual Studio kysyy .NET Framework -version, jolla ohjelma ajetaan ja muutamia kysymyksiä. Oleelisena tässä projektissa on valita rasti pois päältä kohdasta ”Use controllers (uncheck to use minimal API.)”. Tämän jälkeen Visual Studio alustaa projektin ja kehittäminen voi alkaa. Kuvissa 11 ja 12 kuvattuna .NET Core Web Api -projektin alustuksen kaksi viimeistä valikkoa.



Kuva 11. .NET Core -projektin alustuksen toinen vaihe.



Kuva 12. .NET Core -projektin alustuksen kolmas vaihe.

API-palvelut .NET Minimal Web Api -projektissa toteutetaan HTTP-metodeilla. App.MapGet-funktio tekee projektiin uuden GET-palveluun, josta käyttäjä voi hakea dataa. App.MapPost on toinen projektissa käytettävä funktio http-kutsujen toteuttamiseen ja sen avulla käyttöliittymä pystyy lähettämään palvelimelle tarvittavaa tietoa, esimerkiksi hautausmaiden lisäämiseen.

.NET -projektin alustuksen jälkeen sovellukseen mallinnettiin EF Core -luokat tietokannan tauluille, jolloin saadaan EF Coren tietokantaoperaatiofunktiot helposti käyttöön ja vaadittavan ohjelmakoodin määrä vähenee. EF Coren kantayhteyden muodostamista varten tehdään kuvassa 13 esiteltävä DbContextClass-luokka, jonka tehtävä on alustaa yhteys tietokantaan sekä muodostaa lista tai listat, joihin tietokannasta haettu data haetaan ja johon tallennettava data asetetaan ennen sen tallentamista tietokantaan. Jokainen REST API:n palvelu, joka käyttää tietokantaa, tarvitsee DbContextClass -luokan objektin, jotta yhteys tietokantaan voidaan muodostaa.

```
using HautaApi.Entities;
using Microsoft.EntityFrameworkCore;

namespace HautaApi
{
    8 references
    public class DbContextClass : DbContext
    {

        //konfiguraatioasetukset
        protected readonly IConfiguration Configuration;

        //konstruktori kontekstille jossa asetetaan konfiguraatiot
        0 references
        public DbContextClass(IConfiguration configuration)
        {
            Configuration = configuration;
        }
        0 references
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            //määritellään käytettävä kanta
            optionsBuilder.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));
        }
        2 references
        public DbSet<Hautausmaat_demo> Hautausmaat_demo { get; set; }
        1 reference
        public DbSet<Haudat_demo> Haudat_demo { get; set; }
        1 reference
        public DbSet<HautojenHenkilot_demo> HautojenHenkilot_demo { get; set; }
        2 references
        public DbSet<Users_demo> Users_demo { get; set; }
    }
}
```

Kuva 13. Kuvattuna DbContextClass -luokan sisältö.

Tietokantaoperaatioita varten ohjelmaan tehdään seuraavat luokat:

- Users.cs
- Hautausmaat.cs
- Haudat.cs
- HautojenHenkilot.cs

Näihin luokiin luotiin kentät tietokannan taulujen mukaisesti ja asetettiin niille tietokannassa olevat asetukset. Sovelluksen jokaisen taulun jokainen rivi yksilöidään juoksevalla numerolla joka nousee yhdellä joka lisäyksessä. Ohjelmakoodin luokissa yksilöivän avaimen tunniste on `[KEY] <property_name>` ja tietokannassa generoitava arvo määritellään `[DatabaseGenerated(DatabaseGeneratedOption.Identity)]`-ominaisuudella. Kuvassa 14 esiteltynä HautojenHenkilot_demo-luokka, jota käytetään tietokannan datan tallentamiseen ja hakemiseen.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace HautaApi.Entities
{
    3 references
    public class HautojenHenkilot_demo
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        [Key]
        0 references
        public int UID { get; set; }
        0 references
        public string? Etunimi { get; set; }

        0 references
        public string? Sukunimi { get; set; }
        0 references
        public DateTime? Birthdate { get; set; }
        0 references
        public DateTime? DeathDate { get; set; }
        0 references
        public int? Syntymavuosi { get; set; }
        0 references
        public int? Kuolinvuosi { get; set; }
        1 reference
        public int? HautaUID { get; set; }
        0 references
        public int? HautausmaaUid { get; set; }
        0 references
        public string? Hautausmaa { get; set; }

    }
}
```

Kuva 14. Kuvattuna HautojenHenkilot_demo-luokka.

Luokkien jälkeen ensimmäisen http-palvelun tekeminen on yksinkertaista. Osoitteessa ”https://localhost:7286/hautausmaat/” haetaan palvelimelta kaikki hautausmaat jotka palveluun on lisätty. Tietokannan SQL-komentona kirjoitettaisiin `”SELECT * FROM HAUTAUSMAAT_DEMO”`, mutta EF Coren avulla sama data voidaan luokkia käyttämällä hakea funktiolla ”`dbContext.Hautausmaat_demo.ToListAsync()`”, joka hakee kaiken datan luokan objekteina ja tallentaa sen muuttujaan joka palautetaan

käyttöliittymälle. Kuva 14 kuvaa ”/hautausmaat”-osoiteessa suoritettavaa koodia, joka palauttaa kaiken datan Hautausmaat_demo-taulusta.

```
app.MapGet("/hautausmaat", async (DbContextClass dbContext) =>
    await dbContext.Hautausmaat_demo.ToListAsync()
);
```

Kuva 15. kuvattuna sovelluksen ”/hautausmaat”-palvelu

POST-komentoon vastaavat palvelut toteutetaan palvelimella käyttäen aiemmin mainittua App.MapPost-funktiota ja ne ovat rakenteeltaan samanlaisia kuin GET-komentojen palvelut. POST-palveluissa tallennuksen tai muun palvelimella suoritettavan operaation jälkeen käyttöliittymälle palautetaan status-koodi ja viesti joiden avulla käyttöliittymä voidaan ohjata tekemään jokin tietty operaatio, esimerkiksi renderöimään tietty animaatio indikoimaan operaation onnistumista tai epäonnistumista.

Uuden hautausmaan luomiseen käytetään ”/lisaahautausmaa”-palvelua. Palvelussa on tarkoituksena muuntaa käyttäjän antaman hautausmaan tiedot Hautausmaat_demo-luokan objektiksi hautausmaa ja lisätä objekti tietokantaan.

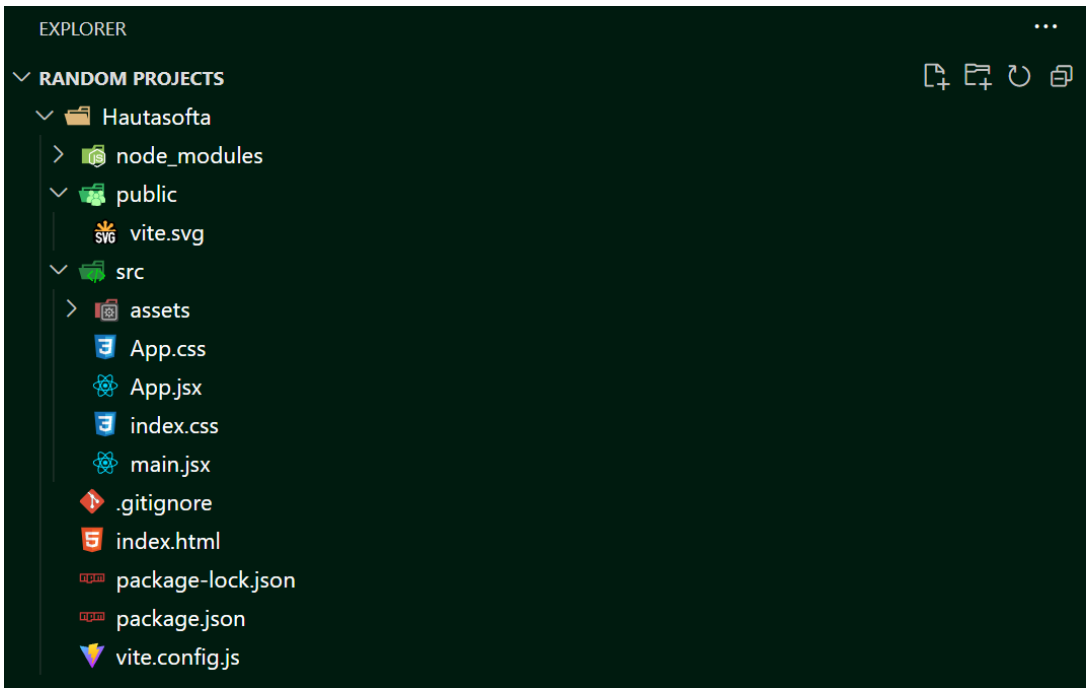
6.2 Käyttöliittymä

Sovelluksen käyttöliittymän alustaminen tapahtuu terminaalin kautta komennolla ”`npm create vite@latest`”. Tällä komennolla npm alustaa Vite-pohjaisen projektin Viten viimeisellä versiolla. Vite kysyy käyttäjältä projektissa käytettävän kirjaston, mahdollisen TypeScript-tuen sekä projektinimen. Tämän projektin kohdalla valittiin React ilman TypeScript-tukea ja projektin nimeksi annettiin ”Hautasofta”.

Tässä opinnäytetyössä IIS:n päälle perustettu verkkosivu palauttaa käyttäjälle index.html-nimisen HTML-tiedoston aina, kun käyttäjä navigoi palvelun

osoitteeseen. Kehitysversion koodista muodostetaan tuotantokäyttöön soveltuva versio React-koodista komennolla ”`npm run build`”. Komento luo dist-nimisen kansion, jossa on optimoitu versio ohjelman koodista, HTML-tiedosto, joka lähetetään käyttäjän laitteelle, sekä web.config-tiedosto, jolla määritellään http-kutsut käännettäviksi https-muotoon. Web.config määrittää myös URL-polut niin, että ohjelman suorittaminen aloitetaan aina tietystä paikasta.

Kun sovellus on alustettu, suoritetaan komento ”`cd Hautasofta`”, jolla siirrytään alustetun projektin kansioon. Tämän jälkeen suoritetaan ”`npm install`”, jolla ladataan vaadittavat npm-paketit projektin node_modules-kansioon. Kuvassa 16 kuvattuna React-projekti, kun komennot on suoritettu. Lopuksi testataan sovelluksen toimivuus suorittamalla ”`npm run dev`”, joka käynnistää sovelluksen paikalliseen kehityspalvelimeen osoitteeseen ”http://localhost:xxxx”, jossa xxxx on portin numero jossa sovellusta suoritetaan.



Kuva 16. Vite-pohjaisen React-projektin kansiorakenne.

Kun ensimmäinen testikäynnistys on suoritettu, voidaan aloittaa projektille oleellisten npm-pakettien lisääminen. Lisättävät lisäosat ovat:

- React Router
- Material UI.

React Routerin lisäys tapahtuu komennolla ”`npm i -D react-router-dom`”. Lisäosan lisäyksen onnistumisen jälkeen otetaan React Routerista löytyvä BrowserRouter käyttöön koko projektissa lisäämällä se App.jsx-tiedostoon ja lisäämällä ohjelman reitit BrowserRouter-tagien sisälle. Kuvassa 17 kuvattuna BrowserRouter-komponenttia käyttävä yksinkertainen reititys. Reittejä lisätään Route-tagilla, jotka lisätään Routes-tagin sisään. Route-tagille lisätään to-arvo ja element-arvo, missä ”to” viittaa URL-reititykseen ja ”element” renderöitävään komponenttiin. Material UI on käyttövalmis, kun paketti on asennettu komennolla ”`npm install @mui/material @emotion/react @emotion/styled`”

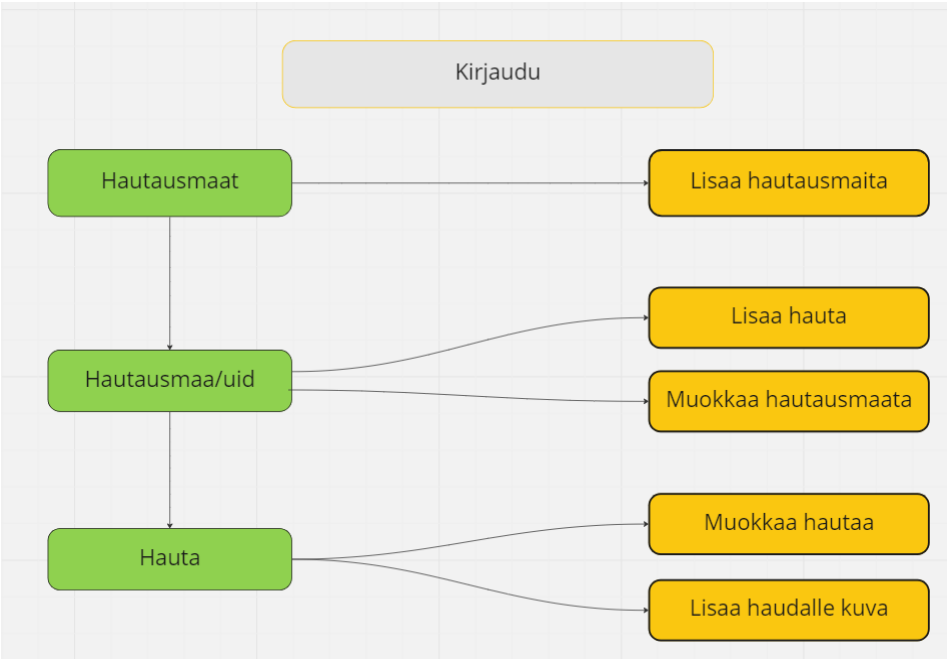
```jsx
import './App.css'
import Hautausmaat from './pages/Hautausmaat';
import Hautausmaa from './pages/Hautausmaa';
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Etusivu from './pages/Etusivu';
import Navbar from './components/Navbar.jsx';
import LisaaKuva from './pages/LisaaKuva';

function App() {
  return (
    <>
      <BrowserRouter>
        <Navbar/>
        <div style={{margin: "Auto", maxWidth: "1920px"}}>
          <Routes>
            <Route index element={<Etusivu />} />
            <Route path='/hautausmaat' element={<Hautausmaat />} />
            <Route path='/hautausmaa/:uid' element={<Hautausmaa />} />
            <Route path='/lisaakuva' element={<LisaaKuva/>} />
          </Routes>
        </div>
      </BrowserRouter>
    </>
  );
}

export default App
```

Kuva 17. Ohjelman App.jsx-tiedosto, jossa importattuna React Routerin tarvitsemat komponentit ja toteutettuna sovelluksen rakenne React Routeria käyttäen.

Pakettien asennuksen jälkeen sovelluksen kehittämistä jatketaan tekemällä sovelluksesta modulaarinen. Käyttöliittymässä Navbar-komponentti on ainoa

oletuksena jokaisella sivulla oleva komponentti. Muut komponentit ovat sivuja tai sivukohtaisia toistuvia komponentteja.

Sovelluksen käyttöliittymään tehtiin yhteensä 9 erillistä sivua. Näistä sivuista 5 on sivuja, joille vaaditaan kirjautuminen sovellukseen tunnuksilla. Kuva 18 kuvaa vihreällä sivut joille kuka tahansa pääsee ilman kirjautumista ja oranssilla kuvatut sivut ovat niitä, jotka vaativat käyttäjätunnuksen ja kirjautumisen.

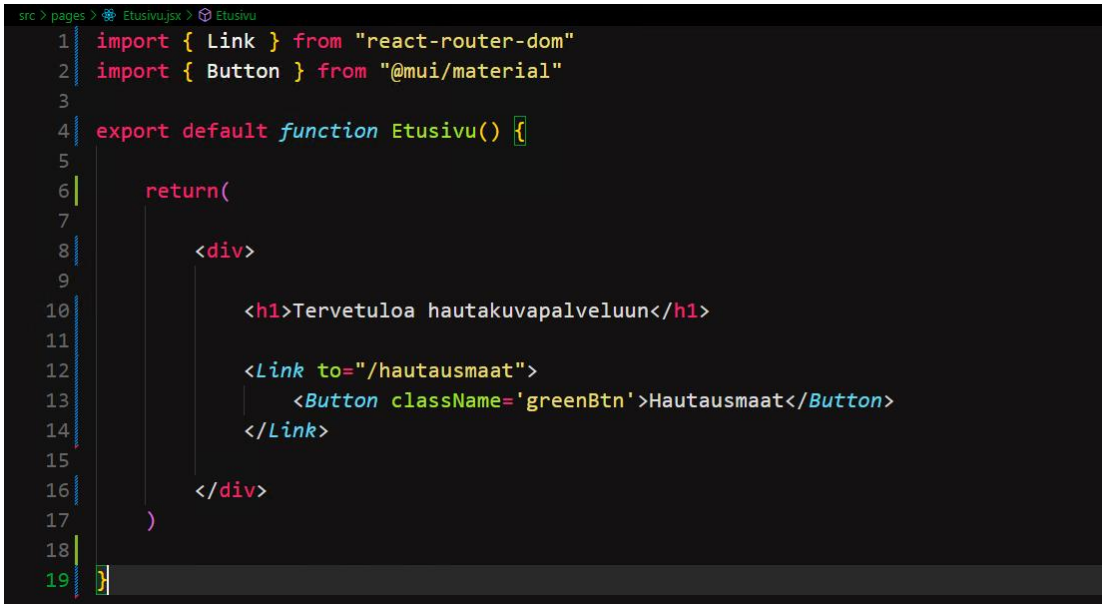


Kuva 18. Sovelluksen vapaat sivut ja kirjautumisen vaativat sivut.

Käyttöliittymä toteutettiin kronologisessa järjestyksessä aiemmin kuvattujen kaavioiden mukaan. Käyttöliittymään toteutetaan ensimmäisenä Etusivu, jonka pohjalta looginen eteneminen sivukartassa on järkevää.

Käyttöliittymän sivua varten luodaan uusi tiedosto .jsx-päätteellä. Sivun nimi voi olla mitä tahansa, mutta nimeämisessä suositeltavaa on mahdollisimman looginen ja selkeä malli, jotta ohjelmakoodin lukeminen on mahdollisimman helppoa. Etusivu-sivua varten tehdään siis Etusivu.jsx tiedosto. Tiedoston luonnin jälkeen siihen lisätään tyhjä JavaScript funktio, jonka return-lauseessa palautetaan käyttäjälle renderöitävä sivu. Kuvan 19 return-lause palauttaa

html-koodina div-elementin, jonka sisällä renderöidään h1-elementti, Link-komponentti ja Button-komponentti.



Kuva 19. Kuvattuna Etusivu-komponentin sisältö.

Sivut, joilla listataan dataa käyttäjän ruudulle, käyttävät JavaScriptin Fetch API:a tiedon hakemiseen REST-rajapinnasta HTTP:n GET-metodilla. Tällaisia sivuja ovat käyttöliittymässä:

- /hautausmaat
- /hautausmaat/:uid

Rajapinta palauttaa datan rajapinnasta JSON-formaatissa, jolloin React-käyttöliittymä pystyy helposti piirtämään datan halutussa muodossa ruudulle. JavaScriptin Map-funktio mahdollistaa taulukkomuotoisen datan piirtämisen ruudulle. Map-funktio mahdollistaa yksittäisen JSON-objektin sisältämien ominaisuuksien piirtäminen yksittäisenä komponenttina. React kuitenkin tuottaa virheen, mikäli piirrettävässä taulukossa ei ole alkioita. Koska data hankitaan vasta, kun komponentti on ensimmäisen kerran renderöity, tuottaa React virheen vaikka datan hakeminen muuten onnistuisikin. Tämän välttämiseksi ohjelmaan tehtiin ”loadereita” eli ohjelma määriteltiin renderöimään ruudulle jotain muuta niin kauan kuin Map-funktiossa piirrettävässä taulukossa ei ole alkiota. Heti, kun taulukossa on alkiota,

piirretään Map-funktiossa määriteltävät komponentit normaalisti. Kuva 20 sisältää return-lauseessa palautettavan div-elementin, jonka sisällä toteutetaan ehtolauseet, jotka määräävät kulloinkin renderöitävän elementin.

```
<div className='dataDiv'>
  {
    pageData != undefined &&

      pageData.map((data, index) => {

          if (haku.length == 0 ){

            return(
                <GraveyardCard data={data} key={index} />

            )
          }

          if (data.nimi.toLowerCase().includes(haku.toLowerCase()) && haku.length > 0) {

            return(
                <GraveyardCard data={data} key={index} />

            )
          }

      })
    }


  {
    pageData === undefined &&
      <div>
        <img style={{height: "2em"}} src={loading}/>
      </div>

  }

</div>
```

Kuva 20. Kuvattuna ”/hautausmaat” sivun datan renderöinti ja loader.

Komponentit, joissa lähetetään dataa rajapinnan POST-palveluille, käyttävät datan lähettämiseen axios-nimistä JavaScript-kirjastoa. Käyttöliittymään määritellään funktio, joka muodostaa palvelimelle lähetettävästä tiedosta objektin ja lähettää sen POST-komennon BODY-osassa JSON-datana. Funktiota kutsutaan button-elementin onClick-tapahtuman jälkeen. Palvelimen palauttaman datan perusteella renderöidään ruudulle joko tilakoodiin perustuva virheilmoitus tai ilmoitus onnistuneesta latauksesta.

Esimerkkinä POST-palvelua hyödyntävästä sivusta käytetään hautausmaiden lisäämiseen tarkoitettua LisaaHautausmaa-komponenttia, joka suoritetaan palvelussa osoitteessa ”/lisaahautausmaa/”. Sivun komponentissa käyttäjä syöttää lisättävän hautausmaan tiedot kenttiin. Tallennuksen jälkeen ”Tallenna”-painikkeen onClick-tapahtumassa kutsutaan funktiota, joka tarkistaa kentissä olevat arvot oikeiksi. Tämän jälkeen funktio suorittaa lähetettävän objektin luomisen ja lähettämisen POST-kutsuna palvelimelle käyttäen axios-kirjastoa.

Palvelimelle lähetettävät arvot saadaan kentistä luomalla jokaiselle kentälle oma useRef-kytkentä. UseRef-kytkentä toimii samantyyppisesti kuin muuttuja normaalia JavaScriptia käytettäessä, sen avulla on mahdollista viitata tiettyyn HTML-elementtiin dokumentissa. LisaaHautausmaa-komponentissa hautausmaan nimen saa useRef-kytkennällä hautausmaan nimen sisältävästä kentästä attribuutilla ”nimi.current.value”. Kuva 21 sisältää axios:ia käyttäen toteutetun POST-kutsun ja siihen tarvittavan datan valmistelun. Palvelin palauttaa käyttöliittymälle JSON-objektin, joka sisältää käyttöliittymän tarvitseman datan ohjelman jatkamiseen. LisaaHautausmaa-komponentissa palvelimen palauttaman statuksen ja operaation tilan vahvistavalla totuusarvolla renderöidään käyttäjälle joko virheilmoitus, mikäli haudan tallennus epäonnistui, tai kenttien tyhjennys ja onnistumisen ilmaiseva ilmoitus käyttäjälle.

```
const nimi = useRef(undefined);
const paikkakunta = useRef(undefined);
const [ok, setOk] = useState(false);
const [virhe, setVirhe] = useState("");
const virhediv = useRef();
const maa = useRef(undefined);

async function handleSubmit() {

    var obj = new Object()

    obj.nimi = nimi.current.value
    obj.paikkakunta = paikkakunta.current.value
    obj.maa = maa.current.value

    if (obj.nimi.length > 0 & obj.paikkakunta.length > 0 & obj.maa.length > 0) {
        setVirhe("")

        const response = await axios.post("https://localhost:7286/lisaahautausmaa", obj)
        .then(response => {
            console.log(response.data.value.teksti)
            if (response.data.value.status == "ok"){

                setOk(true)
            }
            else{
            }
        })
    }
    else {
        setVirhe("Täytä kaikki kentät")
    }
}
```

Kuva 21. LisaaHautausmaa-komponentin tallennus funktio.

Käyttöliittymän ulkonäköön käytettiin Material UI -käyttöliittymäkirjastoa ja kirjoitettiin osa tyyleistä tyhjältä pöydältä. Sovellukseen pyrittiin toteuttamaan mahdollisimman yhteneväinen käyttöliittymän ulkonäkö. CSS-tyylejä käytettiin pääasiassa sovelluksen layoutin määrittelyyn. Tällaisia ovat esimerkiksi elementtien järjestys ruudulla sekä sovelluksen mediakyselyiden keskeytyspisteet, joilla määritellään millaisena tietty elementtityyppi esitetään ruudulla.

Sovelluksen layoutin toteutuksessa käytetään CSS:n Grid-ominaisuutta. Grid muuttaa elementin ruudukoksi, jonka ruutujen kokoa ja muotoa pystyy muokkaamaan. Gridiä käytettiin tehdessä sovellusta eri ruuduille, jolloin pikselien määrä ja ruudun muoto ja koko muuttuvat. Gridillä layoutin määritteleminen on helppoa muuttamalla CSS-tiedostossa keskeytyspisteen

sisällä olevaa grid-template-columns-ominaisuuden arvoa. Kuvan 22 CSS-koodissa jokaisen dataDiv-luokan elementin sarakkeiden määrää vaihdetaan käyttäen keskeytyspisteitä. Keskeytyspisteiden pikselimäärät kehittäjä voi valita itse. Sovelluksen listat, kuten hautausmaat ja haudat, käyttävät gridiä datan listaamisessa ruudulle. Grid-ruudukon koko kasvaa isommissa keskeytyspisteissä isommilla ruuduilla. Mobiilissa listat piirretään yhden sarakkeen levyisiksi, kun taas 1920-pikselin leveydellä sarakkeita piirretään kolme. Tällaista layoutia käytetään siksi, että mobiilissa ruudun tilaa on vähemmän, jolloin sarakkeita mahtuu ruudulle vähemmän ja vastaavasti isolla ruudulla dataa pystyy piirtämään enemmän.

```
@media (width > 767px){
  .dataDiv{
    grid-template-columns: auto auto ;
  }

}

@media (width > 1920px){
  .dataDiv{
    grid-template-columns: auto auto auto ;
  }

}
```

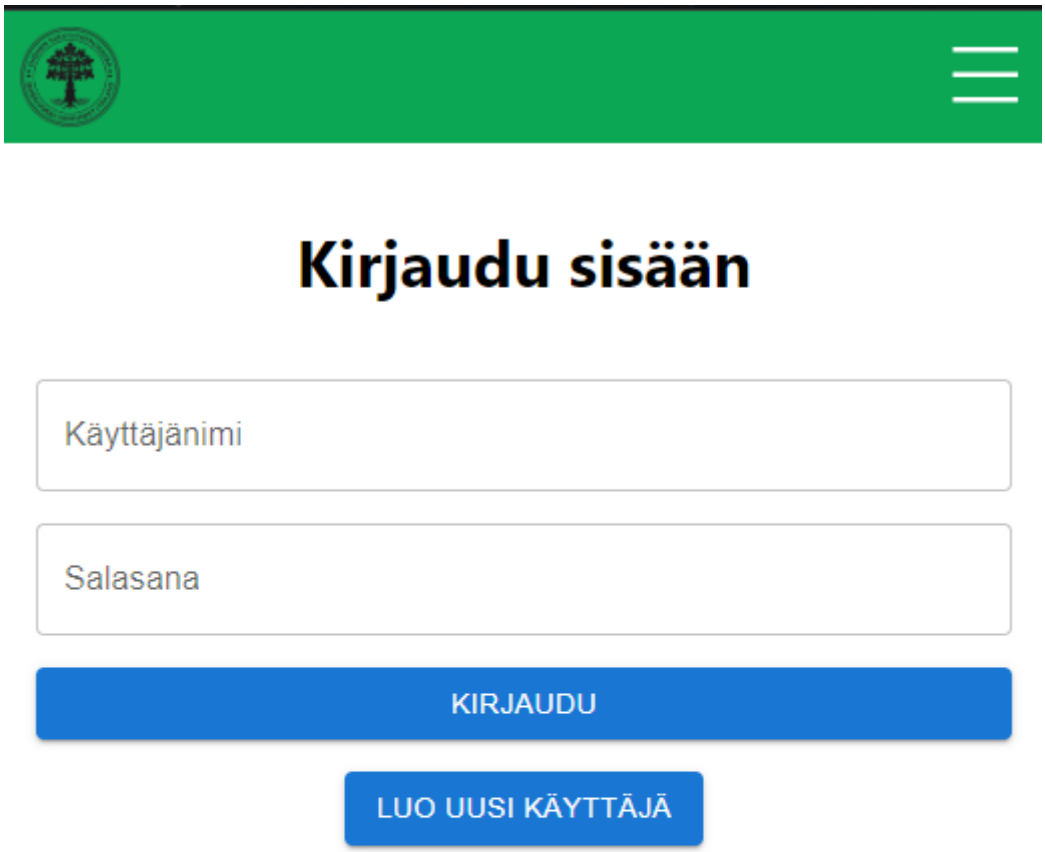
Kuva 22. Esimerkki CSS-keskeytyspisteestä.

6.3 Sovelluksen ulkonäkö

6.3.1 Kirjautumissivu

Käyttäjä ohjataan kirjautumissivulle jos sovelluksen sessiomuistista ei löydy käyttäjän yksilöivää user-objektia. Jos user-objekti löytyy, käyttäjälle osoitetaan ”/kirjaudu” osoitteessa ”Tervetuloa! <käyttäjänimi>” teksti ja nappi

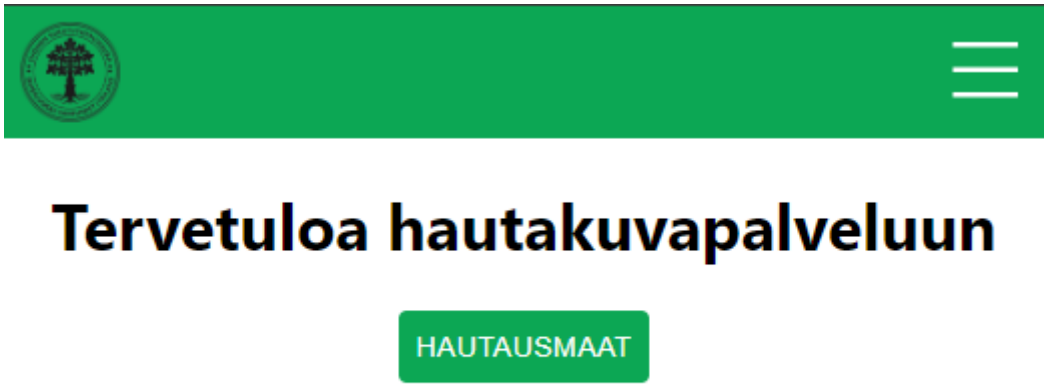
josta käyttäjä pääsee selaamaan hautausmaita. Mikäli user-objektia ei löydy, käyttäjälle renderöidään kirjautumiseen vaadittavat input-kentät, uuden käyttäjän luomisen avaava painike ja painike, jota painamalla käyttäjän tunnus ja salasana varmennetaan oikeellisiksi. Tällaisia tarkistuksia ovat tyypillisesti sovelluksissa esimerkiksi salasanan pituus ja eri merkkityypit, mutta tässä toteutuksessa ainoana tarkistuksena toimii käyttäjän tunnuksen ja salasanan pituus, joiden pitää olla vähintään yhden merkin mittaisia. Sovellus muistaa sessiossa käyttäjän kunnes selain suljetaan. Kuva 23 kuvaa kirjautumissivun ulkonäön



Kuva 23. Kirjautumissivu-komponentti

6.3.2 Etusivu

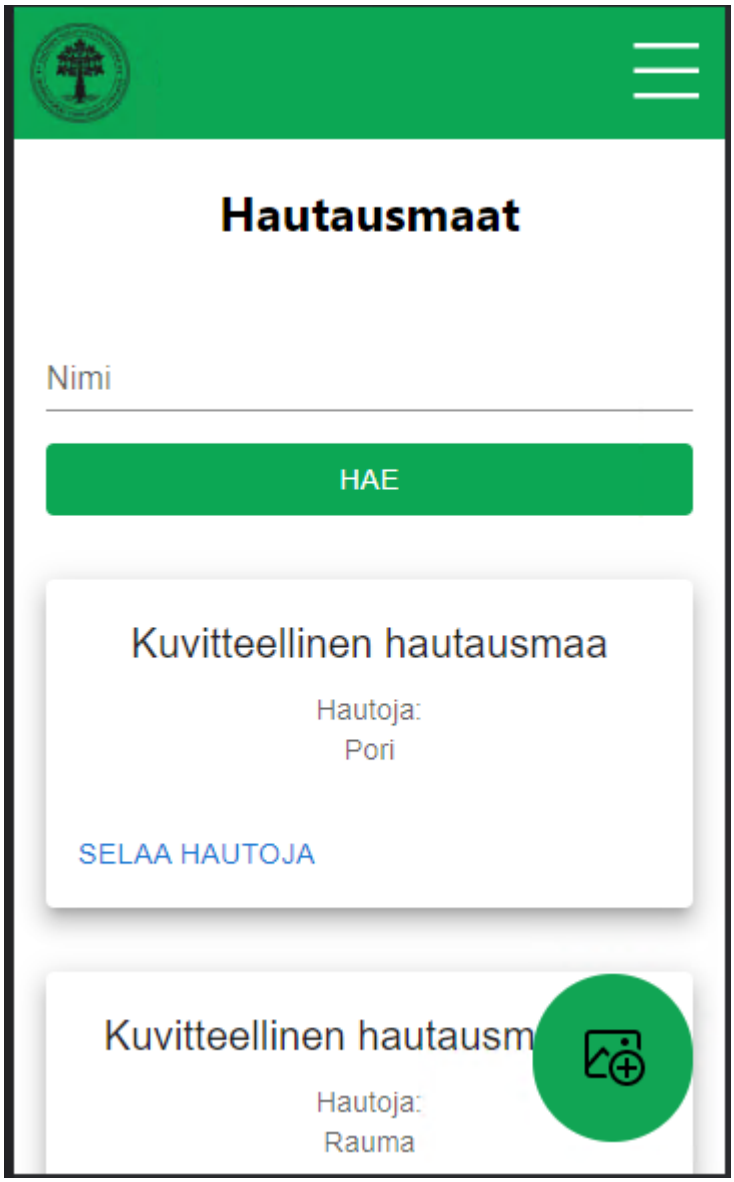
Etusivu-komponentti vastaa käyttäjälle osoitteessa ”/”, joten se toimii sivuna, jonka kautta kaikki muu operointi tapahtuu ja jonne käyttäjä ensimmäisenä lähetetään palveluun siirryttäessä. Etusivu-komponentti on hyvin yksinkertainen ja se sisältää vain tekstin ”Tervetuloa hautakuvapalveluun” ja painikkeen, josta käyttäjä pääsee selaamaan hautausmaita. Sivulle tullaan lisäämään ominaisuuksia tulevaisuudessa, mutta prototyyppiversiossa ei esitellä etusivua vaan muita sivuja, joten sen toiminnallisuus on jätetty vähäiseksi tähän työhön. Etusivun ulkonäkö on kuvattu kuvassa 24.



Kuva 24. Etusivu-komponentti

6.3.3 Hautausmaat

Hautausmaat-komponentti renderöi käyttäjän ruudulle kaikki tietokannasta löytyvät hautausmaat ja niiden perustiedot, kuten paikkakunnan ja hautamäärän, mikäli sellaiset on hautausmaalle kirjattu. Komponentti hakee API:sta datan GET-kutsulla aiemmin kuvatulla tavalla, jonka jälkeen datan jokainen objekti mallinnetaan Material UI:n Card-komponentiksi. Card-komponenttiin lisätään painike, jota painamalla käyttäjä pääsee selaamaan hautausmaahan linkattuja hautoja. Ladatun datan lisäksi sivun alareunaan piirretään painike, jota painamalla käyttäjä siirretään hautausmaan lisäämiseen tai kirjautumissivulle, riippuen siitä onko käyttäjä kirjautunut sisään. Hautausmaat-komponentin ulkonäkö on kuvattu kuvassa 25 niin, että siinä näkyy sivun hakutoiminto, hautausmaiden lisäämisen painike, sekä Card-komponentti, jollaisena haettu data renderöidään käyttäjälle.

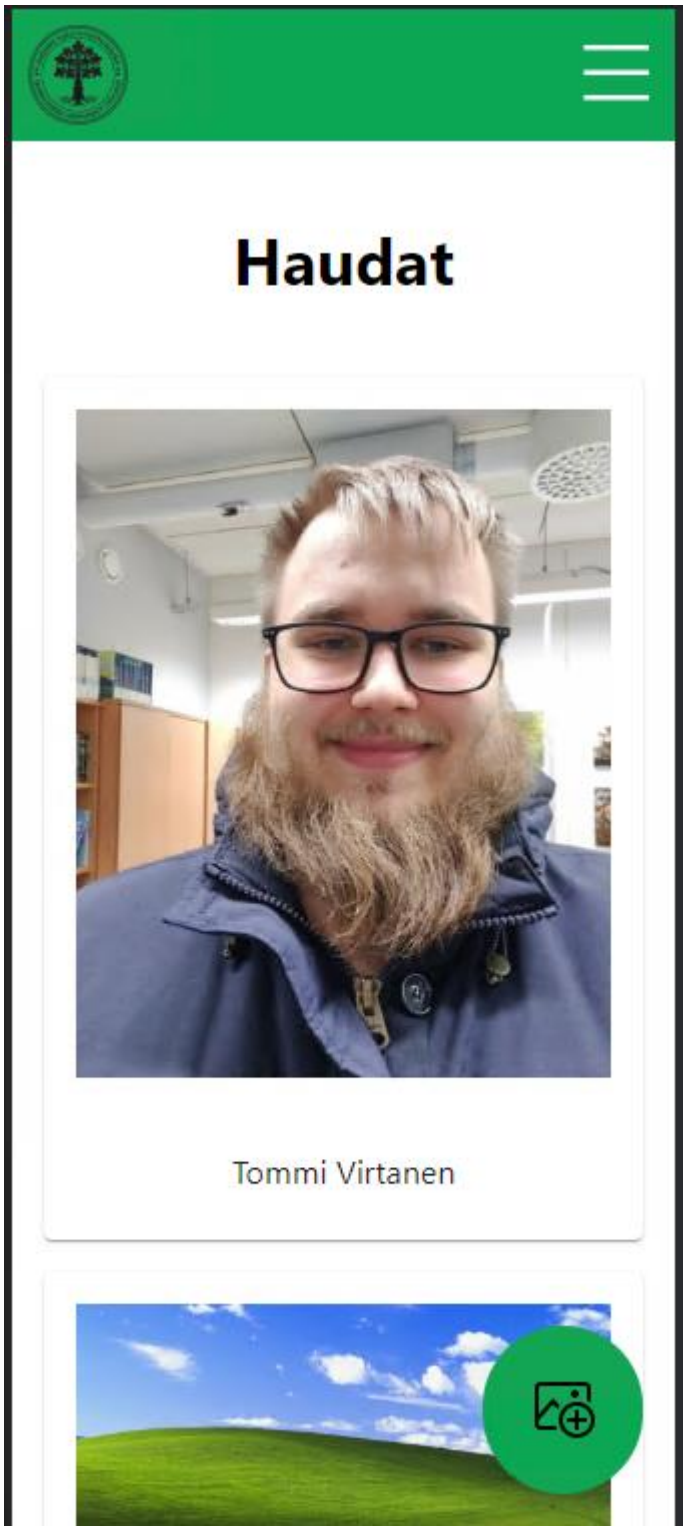


Kuva 25. Hautausmaat-komponentti

6.3.4 Hautausmaa

Hautausmaa-komponentti toimii identtisesti Hautausmaat-komponenttiin verrattuna. Komponentti hakee dataa API:sta käyttäen GET-metodia ja piirtää haetun datan ruudulle. Hautausmaa-komponentissa haettava data on hautausmaiden dataa ja kuvia, jotka haetaan erillisestä tiedostokansiosta palvelimella, jolloin GET-metodin kuormaa vähennetään ja mahdollistetaan kuvien osittainen lataaminen hakemalla tekstimuotoinen data ensin. Komponentissa käytettävä data haetaan tietokannasta kahdesta eri taulusta, Haudat_demo ja HautojenHenkilot_demo-tauluista.

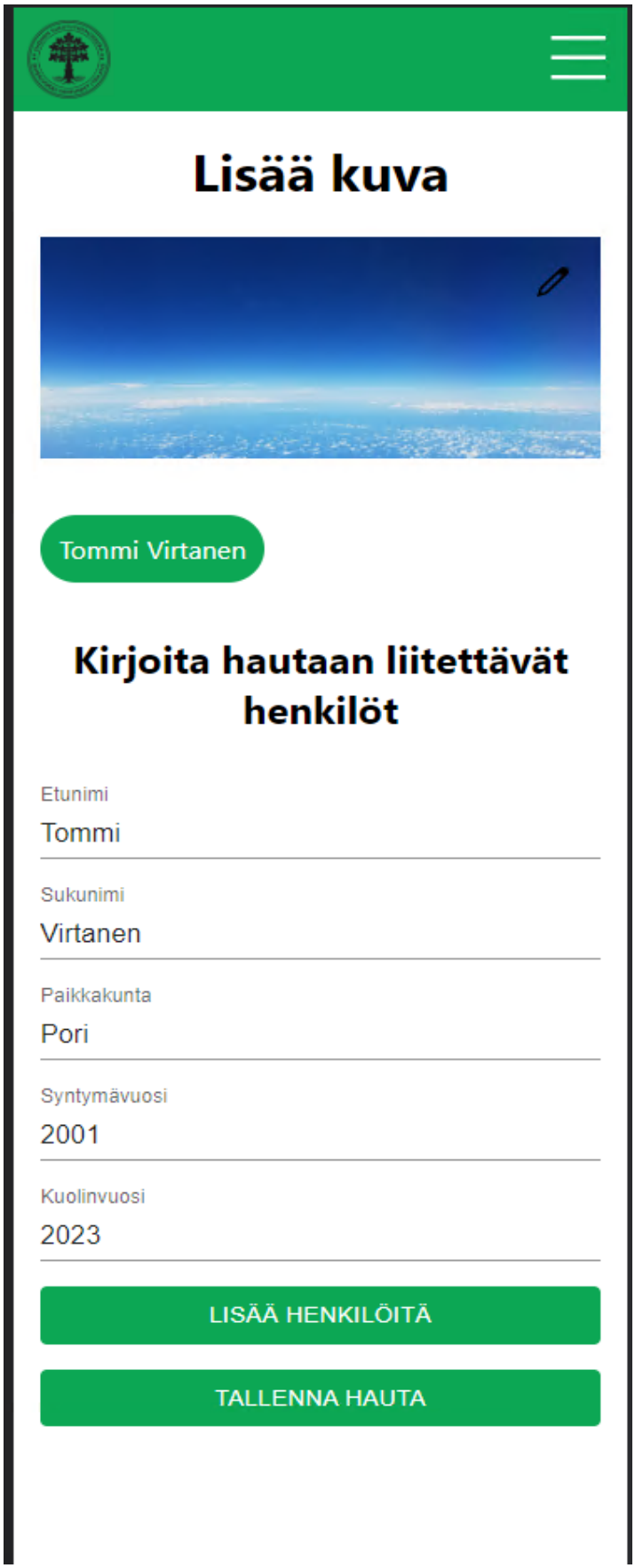
Komponentti piirtää kuvan ja hautaan liitettyjen henkilöiden nimen tai nimet kuvan alle, jolloin hautoja katsova käyttäjä näkee helposti kuvan hautaan haudattujen henkilöiden nimet. Kuten Hautausmaat-komponentissa, myös tässä komponentissa on painike vasemmassa alakulmassa. Hautausmaa-komponentin painikkeesta käyttäjä pystyy sisäänkirjautuneena lisäämään hautausmaalle haudan ja kirjautumattomana hänet ohjataan kirjautumissivulle. Kuva 26 kuvaa yksittäisen hautausmaan haudat, niihin liitetyn kuvan ja henkilöt sekä hautausmaan lisäämiseen johtavan painikkeen. Tämän työn vaiheessa käytetyt kuvat ovat testidataa ja ne tallennetaan kehityksessä käytettävän laitteen muistiin kehityksen aikana.



Kuva 26. Hautausmaa-komponentti

6.3.5 LisaaHauta

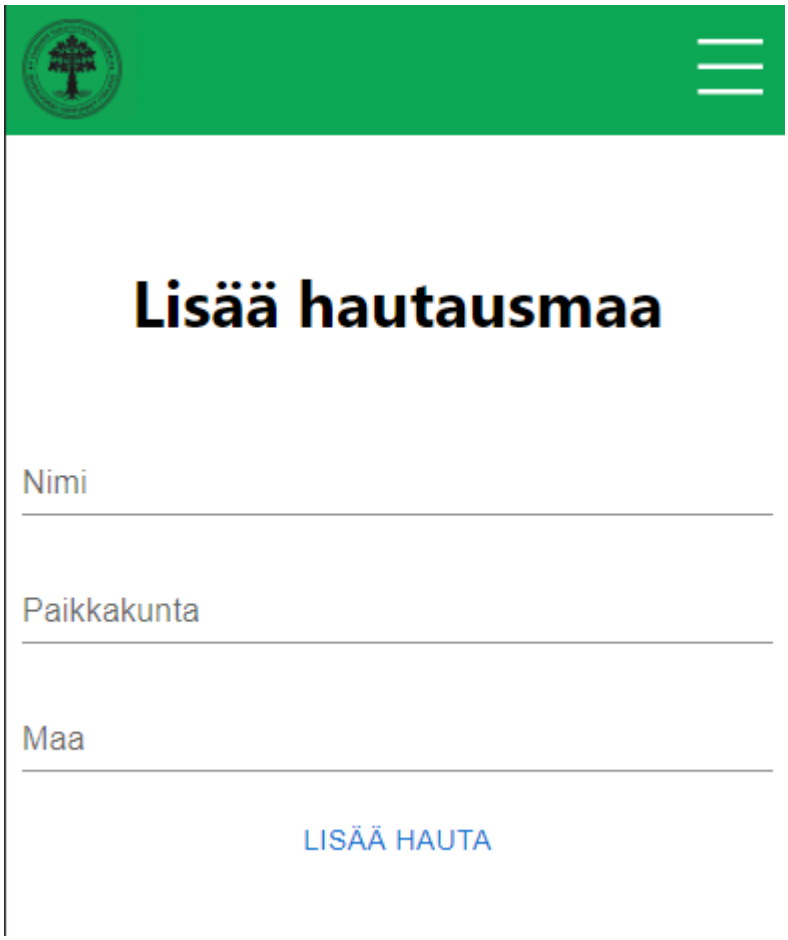
LisaaHauta-komponenttiin käyttäjä siirtyy painettuaan Hautausmaa-komponentin alareunassa olevaa painiketta ja valittuaan tiedostosta lisättävän haudan kuvan. Käyttäjälle aukeavassa sivussa näkyy käyttäjän lisäämä kuva ja sen alla hautaan lisättävien henkilöiden lisäämiseen tarvittavat tekstikentät. Kuvassa 27 kuvattuna LisaaHauta-komponentti ja sen ulkonäkö. ”LISÄÄ HENKILÖITÄ” -painikkeesta käyttäjä liittää tekstikenttiin kirjoitetun käyttäjän hautaan, jonka jälkeen lisätyn henkilön nimestä muodostuu merkintä tekstikenttien yläpuolelle. Käyttäjän on mahdollista lisätä hautaan useita henkilöitä kirjoittamalla jokaisen tiedot ja painamalla ”LISÄÄ HENKILÖITÄ” -painiketta. ”TALLENNA HAUTA” -painikkeesta sovellus lähettää lisätyt tiedot palvelimelle ja tallentaa ne tietokantaan. Tämän jälkeen käyttäjälle palautetaan objektissa status, jonka perusteella käyttäjälle joko piirretään virheilmoitus jos ohjelma kohtasi virheen tai käyttäjä siirretään aiempaan Hautausmaa-komponenttiin, josta käyttäjä on haudan lisännyt.



Kuva 27. LisaaHauta-komponentti

6.3.6 LisaaHautausmaa

LisaaHautausmaa-komponentti mahdollistaa hautausmaiden lisäämisen tietokantaan. Kuvassa 28 kuvattuna LisaaHautausmaa-komponentti ja sen elementit. Lisäämisen jälkeen hautausmaa tulee näkyväksi Hautausmaat-komponenttiin. LisaaHautausmaa-komponenttiin on pääsy vain kirjautuneilla käyttäjillä.



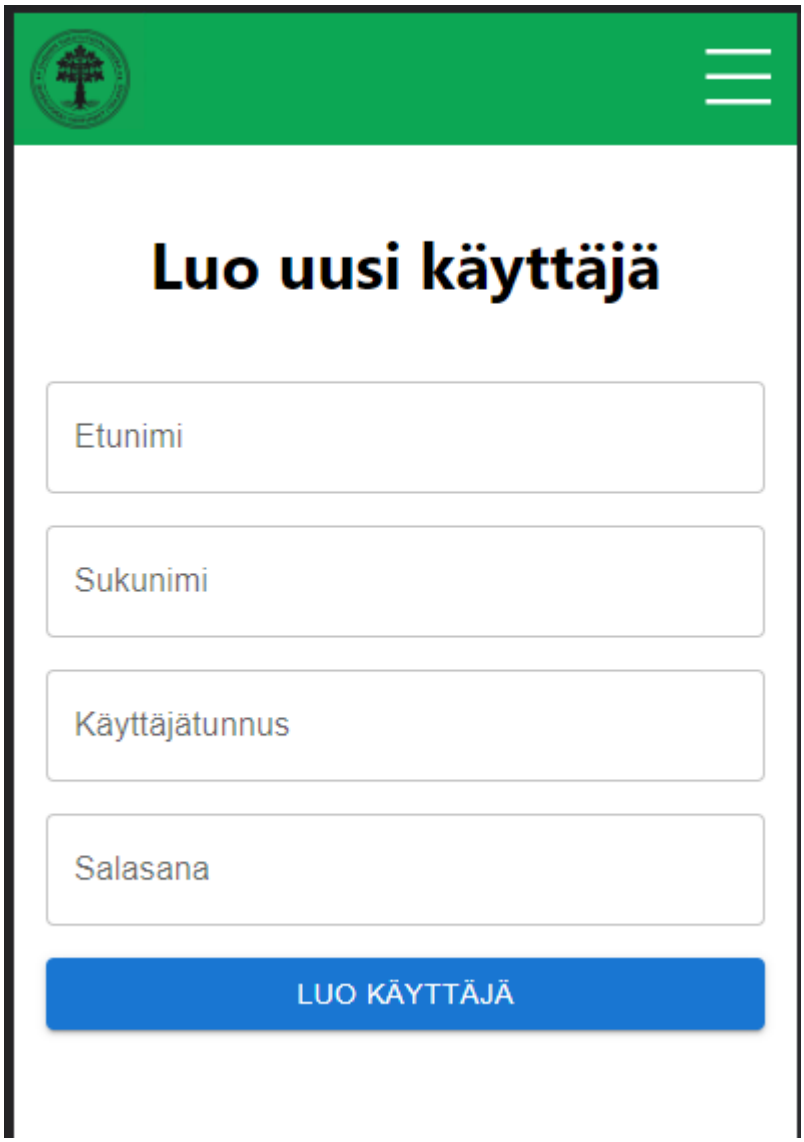
Kuva 28. LisaaHautausmaa-komponentti

6.3.7 LuoUusiKayttaja

LuoUusiKayttaja-komponentti vastaa käyttäjälle ”/luokayttaja”-osoitteessa. Komponentissa käyttäjälle piirretään tekstikentät, joihin käyttäjä syöttää luotavan käyttäjän tiedot. Tiedot, jotka käyttäjän luonnissa kysytään ovat:

- Etunimi
- Sukunimi
- Käyttäjätunnus, jolla palveluun kirjaudutaan sisään
- Salasana

Kuva 29 kuvattuna käyttäjän luomiseen käytettävä komponentti. Kun käyttäjä lähettää tiedot palvelimelle ”LUO KÄYTTÄJÄ” -painiketta painamalla, palvelin salaa salasanan käyttäen bcrypt-kirjastoa, jolloin käyttäjän salasana ei ole tietokannassa selväkielisenä, vaan suojattuna. Lähetetyt tiedot tallennetaan tietokantaan, jonka jälkeen käyttäjä ohjataan Etusivu-komponenttiin.



Kuva 29. LuoUusiKayttaja-komponentti

7 LOPUKSI

Ohjelmistoalalla sovellusten julkaisun jälkeen sovellusta ylläpidetään toimittajan toimesta ja ylläpidosta löytyy merkintä vaatimusmäärittelyssä sekä itse sopimuksessa lähes poikkeuksetta. Ylläpitotoimien jälkeen toimittaja yleensä tarjoaa maksettuna palveluna lisäominaisuuksien kehittämistä sovellukseen. Lisäominaisuuksilla tarkoitetaan sellaisia toiminnallisuuksia tai ominaisuuksia, joita ei vaatimusmäärittelyssä tai tilaussopimuksessa

määritelty, mutta joita asiakas myöhemmin pyytää toteutettavaksi sovellukseen. Tämän opinnäytetyön osalta jatkokehitystä tehdään siinä vaiheessa, kun sovelluksesta on päätetty tehdä tarjous asiakkaan toimesta.

Sovelluksen prototyyppimallin päälle on kuitenkin jo visioitu erilaisia mahdollisia lisäominaisuuksia. Tällaisia ominaisuuksia olisi esimerkiksi OCR (Optical Character Recognition) eli tekstintunnistus. Tekstintunnistuksen mahdollista hyödyntämistä on harkittu hautakuvien tekstin tutkimisen avuksi.

Opinnäytetyön aikana sovellusta ei saatu valmiiksi johtuen sovelluksen laajuudesta sekä opinnäytetyön aikataulusta. Sovelluksen kehittämistä jatketaan normaalisti opinnäytetyön jälkeen. Opinnäytetyön tarkoitus oli lisätä tietoisuutta verkkosovelluksen kehittämiseen liittyvistä asioista. Työn aikana keräsin hyvän määrän tietoa verkkopalveluiden kehittämisestä ja huomionarvoisista asioista niitä kehitettäessä. Työ opetti hankkimaan tietoa Internetistä tehokkaammin, sekä kirjoittamaan aiheesta tieteellisestä näkökulmasta. Kehitystyö alkoi käytännössä tyhjältä pöydältä aiheiden ja tekniikoiden tuntemattomuuden takia. Työn aikana kuitenkin opein mielestäni hyvin tiettyjä .NET Coren ja Reactin ominaisuuksia ja niiden käyttämisen mahdollisimman omatoimisesti. Myös REST-arkkitehtuurin ominaisuudet tulivat tutuiksi alkuvaiheiden epäonnistumisten ja yritysten jälkeen. Työn aihe itsessään on erittäin laaja ja sen käsittely kokonaisuudessaan olisi ylittänyt tämän työn pituuden, joten työssä pysyttiin enemmän teoreettisella tasolla kertoen sovelluksen kehityksestä.

Uskon sovelluksen tulevan tuotantokäyttöön. Sovellusta esiteltiin asiakkaalle alkukeväästä 2023, jolloin reaktio asiakkaan henkilöstön puolesta oli positiivinen. Jatkokehityksen osalta sovellukselle on mietitty muutamia lisäominaisuuksia valmiiksi. Näiden toteutuminen on kuitenkin riippuvainen siitä tilaataanko sovellus asiakkaan toimesta vai ei. Oman uskomukseni ja kokemusteni mukaan sovellus helpottaa ja nopeuttaa asiakkaan toimintaa automatisoimalla aikaa vieviä toimintoja asiakkaan aikataulusta. Tästä syystä uskon sovelluksen päätyvän loppukäyttäjien käytettäväksi tuotantoon. Sovelluksen asiakkaaseen ollaan yhteydessä siinä vaiheessa, kun

sovelluksesta on esittää täysin toimiva ja ehjä ohjelmakokonaisuus. Aikataulua sovelluksen valmistumiselle ei ole asetettu, mutta oma tavoitteeni on saada kesän 2023 aikana sovellus testattavaksi toimeksiantajan muille työntekijöille.

LÄHTEET

Chrome Developers. (24.5.2022). Overview. Haettu 23.2.2023 osoitteesta https://developer.chrome.com/docs/lighthouse/overview/

MDN Web Docs. (N.d). What is web performance. Haettu 8.2.2023 osoitteesta https://developer.mozilla.org/en-US/docs/Learn/Performance/What_is_web_performance

Mui. (N.d.) Material UI – Overview. Haettu 5.2.2023 osoitteesta https://mui.com/material-ui/getting-started/overview/

React Router. (N.d.). Feature Overview. Haettu 1.3.2023 osoitteesta https://reactrouter.com/en/main/start/overview

React. (N.d.) Components and Props. Haettu 4.2.2023 osoitteesta https://reactjs.org/docs/components-and-props.html

Microsoft. (N.d.) IIS Home. Haettu 1.3.2023 osoitteesta https://www.iis.net/

Microsoft. (13.2.2023) What is .NET? Introduction and overview. Haettu 14.3.2023 osoitteesta https://learn.microsoft.com/en-us/dotnet/core/introduction

Statista (2023). Percentage of mobile device website traffic worldwide from 1st quarter 2015 to 4th quarter 2022. Haettu 30.01.2023 osoitteesta https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices.

Web-Ostajan Opas. (N.d.). Hyvä vaatimusmäärittely kuvaa mitä halutaan, muttei kerro miten. Haettu 8.2.2023 osoitteesta https://web-ostajanopas.fi/2022/11/16/hyva-vaatimusmaarittely-kuvaa-mita-halutaan-muttei-kerro-miten/

W3Schools.com. (N.d.a). ASP and ASP.NET Tutorials. Haettu 18.2.2023 osoitteesta https://www.w3schools.com/asp/default.asp

W3Schools.com. (N.d.b). React Router. Haettu 18.2.2023 osoitteesta https://www.w3schools.com/react/react_router.asp