
Roni Sirén

# Hautauspalvelualan digitaalinen hautapaikan hakutyökalu

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikka, insinööri (AMK)

Insinöörityö

8.4.2025

## Tiivistelmä


| | |
|---|---|
| Tekijä: | Roni Sirén |
| Otsikko: | Hautauspalvelualan digitaalinen hautapaikan haku-työkalu |
| Sivumäärä: | 34 sivua + liitteet |
| Aika: | 8.4.2025 |
| Tutkinto: | Insinööri (AMK) |
| Tutkinto-ohjelma: | Sähkö- ja automaatiotekniikka, insinööri (AMK) |
| Ammatillinen pääaine: | Automaatiotekniikka |
| Ohjaajat: | Lehtori Timo Tuominen |


---


Avainsanat: Verkko-ohjelma, käyttöliittymä, muuttuja, funktio, metodi


---


Tämän opinnäytetyön alkuperä on tarkastettu Turnitin Originality Check -ohjelmalla.

Tässä insinöörityössä kehitetään verkkopohjainen alusta, jonka avulla hautapaikat on mahdollista etsiä interaktiivisen kartan ja muokattavien hakuparametrien avulla. Hakuparametreihin kuuluu hautauksen kannalta pakollisia asioita, kuten hautapaikan sijainti, hautaustapa, uskonto, huoltopalvelut ja hautauksen hinta. Lisäominaisuutena ohjelmassa on portaali, jonka avulla käyttäjät voivat täyttää hautausalan sähköisiä lomakkeita globaalisti. Universaalilla lomakkeella tarkoitetaan lomakkeita, jotka voidaan täyttää riippumatta siitä, missä päin maailmaa alustaa käytetään.

Ohjelman tavoitteena on digitalisoida hautauspalvelualaa, ja samalla sen on tarkoitus auttaa hautapaikan valitsemista koskevien vaikeiden päätösten tekemisessä. Lisäksi kehitetyn alustan ja sen työkalujen avulla helpotetaan hautaamiseen liittyvän prosessin eri osapuolten välistä yhteistyötä. Insinöörityössä tarkastellaan useita JavaScript esimerkkikoodeja. JavaScript esimerkkikoodit, jotka ovat osana suurempaa ohjelmoitua kokonaisuutta on listattu liitteissä 1, 2 ja 3 täydessä ohjelmansisäisessä kontekstissaan, jotta ne olisivat mahdollisimman helposti luettavissa.

## Sisällys

## Lyhenteet

NPM: *Node Package Manager*. Node.js-ympäristölle kehitetty JavaScript-pakettien paketinhallintajärjestelmä.

JSON: *JavaScript Object Notation*. Standardi JavaScript-kielen syntaksiin perustuva tekstipohjainen tiedostotyyppi.

CSS: *Cascading Style Sheets*. Tekstipohjainen tiedostotyyppi, joka on kehitetty määrittelemään ohjelmatiedostoille muotoiluohjeita.

MVP: *Minimum viable product*. Startup-piireissä syntynyt tuotekehityksen konsepti, jossa tuotetta kehitetään siihen saakka, kunnes se täyttää vain varhaisten asiakkaiden toiveet. Käytetään tyypillisesti markkinoille pääsemiseen.

# 1 Hautausmaaohjelman rakenne

## 1.1 Hautausmaaohjelman suunnittelufilosofia

Tässä insinöörityössä kehitetään verkkopohjainen alusta, jonka työkalujen avulla voidaan selata ja etsiä avoimia hautapaikkoja. Työkaluihin kuuluu interaktiivinen kartta, sekä hautauksen kannalta pakollisia hakuparametrejä, kuten hautapaikan sijainti, hautaustapa, uskonto, huoltopalvelut ja hautauksen hinta.

Hakuohjelma on rakennettu verkkoselaimella käytettäväksi. Sen tarkoituksena on, että sitä olisi mahdollista käyttää kaikilla yleisillä selaimilla ja käyttöjärjestelmillä. Verkkosovelluksen rakenteen hallittavuuden varmistamiseksi hakuohjelma on ohjelmoitu siten, että ohjelman sisäiset funktiot voivat käyttää eri valtioiden ja hautausmaiden kartta- ja asiadataa ilman funktioiden koodin erillistä muokkaamista.

Hakuohjelman funktiot on pyritty ohjelmoimaan modulaarisesti, eli funktiot on jaettu useisiin pienempiin komponentteihin. Modulaarisuuden tarkoituksena on, että aikaisempien ominaisuuksien laajentaminen ja uusien lisääminen olisi mahdollisimman yksinkertainen ja suoraviivainen prosessi.

## 1.2 Hautausmaaohjelman perusrakenne

Tässä insinöörityössä kehitettävä ohjelma on React Vite -verkko-ohjelma. React on käyttöliittymien kehittämiseen tarkoitettu JavaScript-kirjasto ja Vite on verkkotyökalujen kehittämiseen tarkoitettu paikallinen kehityspalvelin. Ohjelma käyttää NPM (*Node Package Manager*) JavaScript pakettien paketinhallintajärjestelmää sekä useita JavaScript-kirjastoja. Ohjelma koostuu lähes kokonaan Java- tai TypeScriptillä kirjoitetuista ohjelmatiedostoista.

Hakuohjelman karttatoiminnallisuudet käyttävät Leaflet JavaScript -kirjastoa sekä OpenStreetMap-rajapintaa. Kartan apuna on käytetty myös Leafletin

alakirjastoja, kuten React Leafletia ja Leaflet Marker Clusteria, jotka lisäävät ominaisuuksia kartan alustamiselle React-ohjelman tuotantoympäristössä.

Lisäksi kehitetyn ohjelman koodin luomisessa on käytetty apuna useita React-alakirjastoja. Muuttujien hallitsemiseen käytettiin React-kirjaston omia hookeja, kuten useEffect, useState ja useRef. React-kirjaston yhteydessä hookeilla tarkoitetaan erityisiä funktioita, joita käytetään esimerkiksi muuttujien tilan hallitsemisessa.

Verkko-ohjelman navigointi on toteutettu React Router Dom-kirjaston avulla, joka mahdollistaa tavallisten linkkien lisäksi arvojen ohjaamisen uusille sivuille sen useNavigation ja useLocation -hookeilla. React Router Dom-kirjaston hookit muun muassa mahdollistavat sen, että suodattimien arvot etenevät myös haun seuraaviin vaiheisiin.

Kehitetyn ohjelman käyttöliittymän muotoilussa käytetään tavallisen css-muotoilun lisäksi React Bootstrap-kirjastoa. React Bootstrap-kirjasto sisältää useita valmiiksi muotoiltuja elementtejä, kuten navigointipalkin ja esiasetettuja painikkeita. React Bootstrap-kirjasto lisää myös sivun rakenteen muotoiluun auttavia työkaluja kuten säiliöitä, sekä valmiiksi ohjelmoituja rivejä ja sarakkeita eri käyttöliittymän elementeille.

Lähes kaikki kehitetyn ohjelman käyttämä data säilytetään JSON-tiedostojen muodossa, mutta sijaintidataa sisältävät tiedostot säilytetään rakenteeltaan hieman erilaisessa GeoJSON-tiedostossa. GeoJSON-tiedostot mahdollistavat sijainti- ja asiatietojen säilyttämisen rakenteeltaan selkeässä muodossa.

## 1.3 Hautausmaaohjelman kehitystyökalut

Tässä insinöörityössä kehitettävä ohjelma ohjelmoitiin Visual Studio Code-ympäristössä, ja kehitysvaiheessa apuna käytettiin NPM-paketinhallintajärjestelmän kehitystyökaluja. Kaikki kirjastot ja apuohjelmat ladattiin Visual Studio Coden terminaalin avulla NPM-komentoja käyttäen. NPM-komennoilla alustetaan

ohjelman tiedostorakenne, sekä lisätään ohjelman kannalta olennaisia React-kirjastoja.

Hakuohjelma tarvitsee toimiakseen dataa, joka sisältää maantieteellisiä koordinaatteja, sekä koordinaatteja koskevaa asiatietoa. Ilman asiatietoja ohjelma näyttäisi vain pisteitä kartalla. Kaikki tällainen data, joka tarvitsee sekä karttaettä asiatietoa on valmistettu geojson.io:n verkkosivulla. Verkkosivulta saatavan datan GeoJSON-tiedosto on täydellinen formaatti ohjelman toiminnan kannalta, koska sama tiedosto pitää sisällään sekä elementin tarkan sijainnin kartalla että kaikkien erikseen määriteltyjen muuttujien arvot. Tämä tarkoittaa sitä, että ohjelma ymmärtää kaikkia geojson.io-verkkosivulla valmistettuja tiedostoja, kunhan esimerkiksi karttamerkeille annetut parametrit on nimetty yhdenmukaisesti.

# 2 Hautausmaaohjelman taustatietoja

## 2.1 Hautausmaaohjelman tilaaja ja tavoitteet

Tämän insinöörityön on tilannut Ondasera-niminen suomalainen nuori, kasvuhakuinen yritys eli startup, joka valmistaa ja suunnittelee It-työkaluja hautausalan toimijoille ja niiden sidosryhmille.

Hautausmaaohjelma pyritään lanseeraamaan Suomen ja Saksan markkinoille hautausalan digitalisaation edelläkävijänä. Ohjelmaa tullaan käyttämään MVP (*Minimum viable product*)-tuotteena, eli ominaisuuksiltaan ainoastaan kysynnän minimivaatimusten täyttävänä tuotteena, jota käytetään esimerkiksi markkinoille pääsemisessä sekä lisärahoituksen keräämisessä. Ohjelmaa tullaan mahdollisesti käyttämään myös alustana useampia ominaisuuksia kattavalle ohjelmalle.

Hautausala on digitalisoitunut hitaasti, jos sitä verrataan moniin muihin aloihin. Alalla käytetään edelleen vanhentuneita ja käytettävyyden näkökulmasta hankalia tiedonhakutyökaluja. Tämän seurauksena monet helposti digitalisoitavissa olevat asiat tehdään edelleen manuaalisesti ilman erillisiä työkaluja. Hautausalan digitalisaation edistäminen on ajankohtaista, ja Ondasera pyrkii saamaan

vahvan aseman kilpailussa toimimalla pioneerina uusille työkaluille ja ajatuksille.

# 3 Hautausmaaohjelman käyttöliittymä

## 3.1 Käyttöliittymän työkalut

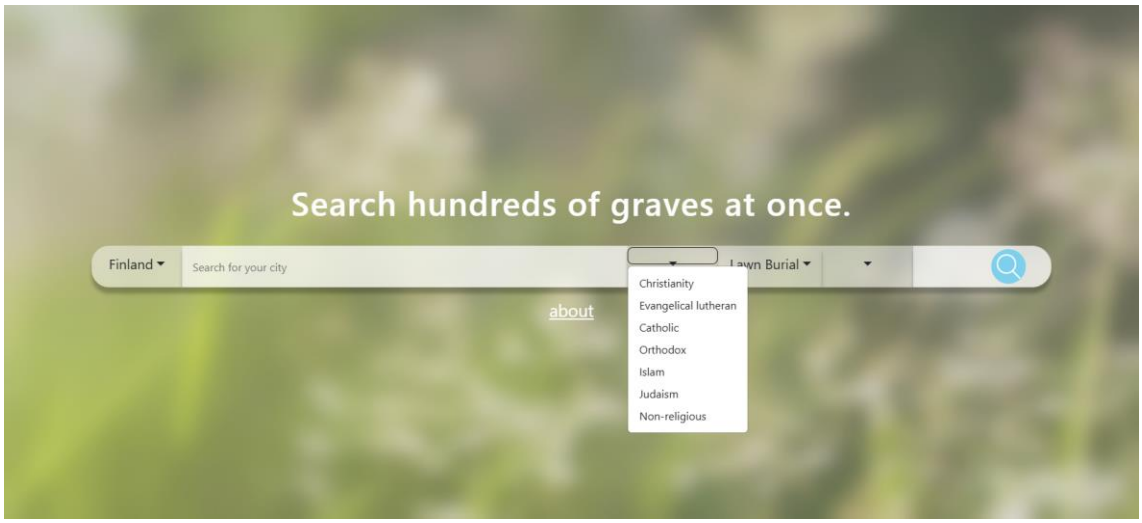
Luvussa 3 käsitellään ohjelman käyttöliittymää, ja esitellään sen työkalujen toiminnallisuutta. Tässä insinöörityössä kehitettävä ohjelma koostuu kahdesta keskeisestä työkalusta: hautakorttelin hakutyökalusta sekä universaalien lomakkeiden hakemiseen ja täyttämiseen tarkoitetusta työkalusta. Hautakorttelin hakutyökalu antaa käyttäjälle suoraviivaisen ja helposti lähestyttävän tavan etsiä hautapaikkoja. Universaalien lomakkeiden täyttötyökalu taas pyrkii yhdistämään eri palveluissa ja osoitteissa säilytettyjä lomakkeita yhteen keskitettyyn paikkaan, jossa hautausalan toimijat voivat löytää ja täyttää alan yleisimpiä lomakkeita samassa paikassa.

Molemmat työkalut on rakennettu saman ohjelmiston arkkitehtuuriin, mutta niiden toiminnallisuus ja käyttöliittymät poikkeavat toisistaan. Seuraavaksi käydään läpi työkalut ja niiden keskeiset toimintaperiaatteet yksi kerrallaan.

## 3.2 Hautakorttelin hakutyökalu

Tässä insinöörityössä kehitettävän hautausmaaohjelman ensimmäinen työkalu on hautakorttelin hakutyökalu. Hakutyökalu toimii samalla verkkosovelluksen etusivuna ja oletustyökaluna. Työkalun käyttöön on useita polkuja, mutta käyttäjälle esitetään ensisijaisena älykäs hakukenttä, johon käyttäjä voi syöttää valtion ja kaupungin lisäksi hautaustyypin, hautatyypin ja uskonnon, jotka valitaan hakukentän yhteydessä olevista monivalinta- lomakkeista. Ohjelma on tehty React-kirjastoa käyttäen, joten sitä voidaan käyttää kaikissa moderneissa selaimissa ja käyttöjärjestelmissä.

Kuvan 1 hakutyökalussa näkyy hakukenttä, jonka avulla valitaan haluttu valtio ja kaupunki. Lisäksi kuvassa 1 näkyy myös lisävalintoihin käytettävät pudostusvalikot. Valtiota ja kaupunkia lukuun ottamatta työkalun hakuparametrien syöttäminen on vapaaehtoista, mutta kaikki muokatut hakuparametrit täyttyvät automaattisesti haun seuraavissa vaiheissa. Tämä helpottaa sopivan hautapaikan etsimistä.




Kuva 1. Hautakorttelin hakutyökalun hakukenttä.

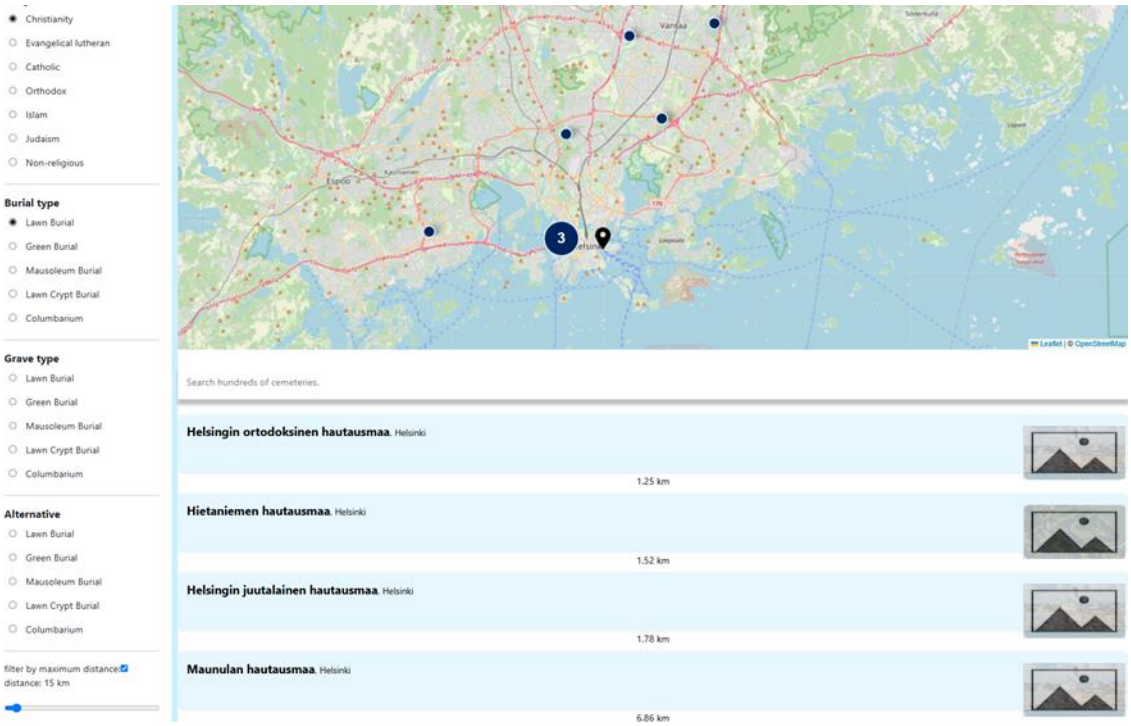
Hautakorttelin hakutyökalu avaa seuraavaksi käyttöliittymän seuraavan sivun, joka näyttää käyttäjän parametreihin soveltuvat hautausmaat. Hakuparametrit ovat edelleen muokattavissa ja käyttäjä voi lisätä myös uusia ehtoja, kuten meren läheisyyden tai maksimietäisyyden valitusta kaupungista. Uskonto ja hautaustyypit valitaan karttasivun vasemmalla puolella sijaitsevien monivalintalomakkeiden avulla. Hautausmaan maksimietäisyys käyttäjän sijainnista määritellään monivalintalomakkeiden alapuolella sijaitsevan liukusäätimen avulla.

Hakutyökalun käyttäjä voi sitten valita työkalun hakutulosten joukosta ehdotetun hautausmaan tai vaihtoehtoisesti etsiä sopivaa hautausmaata manuaalisesti kuvassa 2 näkyvästä sivun yläosaan upotetusta interaktiivisesta kartasta. Jos käyttäjällä on ennestään tiedossa jokin tietty hautausmaa, johon hän haluaa tutustua, sitä voi etsiä sivulta suoraan hautausmaan nimellä. Hautausmaanhaun

hakupalkki ehdottaa hakukenttään syötetyn tekstin kannalta sopivia hautausmaita etsinnän helpottamiseksi.

Jotta hakutyökalun karttaa olisi aina mahdollisimman helppo lukea ja käyttää, yhdistyvät toisiaan lähellä olevat hautausmaita esittävät merkit klusteriryhmäksi, joka myös näyttää kuinka monta jäsentä klusteriryhmään kuuluu. Klusteriryhmällä tarkoitetaan merkkiä kartalla, johon on yhdistetty useiden eri yksittäisten sijaintien merkit.

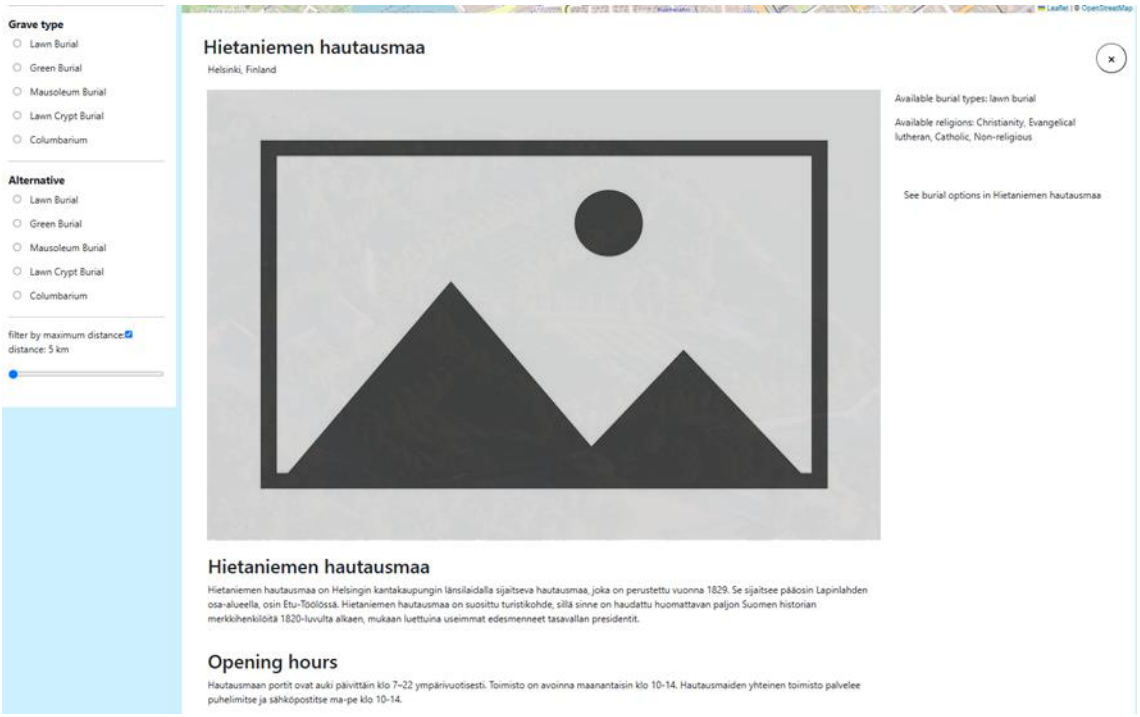
Kuvasta 2 näkee miten ”Hietaniemen hautausmaa”, ”Helsingin ortodoksinen hautausmaa” ja ”Helsingin juutalainen hautausmaa” muodostavat kolmen hautausmaan klusteriryhmän. Hakutyökalun kartalla klusteriryhmään viittaa numeron sisältävä symboli, jota painamalla kartta automaattisesti tarkentaa näkymän etäisyydelle, jolla kaikkia ryhmän jäseniä voidaan tarkkailla kartalla erikseen.




Kuva 2. Hakutyökalun kartta, hakuparametrit ja hakutulokset.

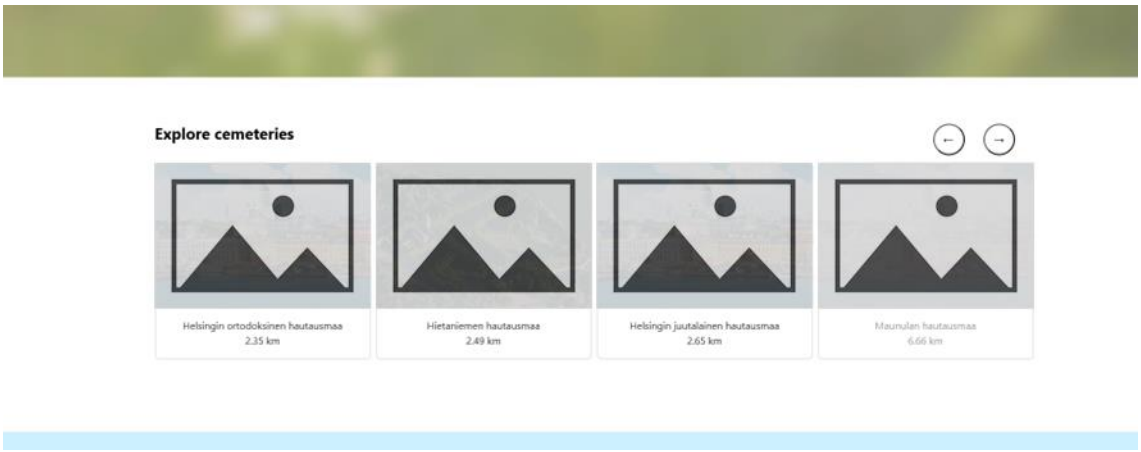
Jos käyttäjä valitsee hautausmaan joko kartalta tai ehdotusten joukosta, kartta tarkentaa automaattisesti valittuun kohteeseen ja sen alapuolelle aukeaa

ponnahdusikkunan kautta hautausmaan profiili. Kuvasta 3 näkee, miten profiiliin on listattu hautausmaan nimi, sen hautaustavat ja sinne hautaamiseen hyväksytyt uskonnot. Lisäksi profiiliin voi lisätä tarkemman kuvaustekstin sekä hautausmaan yleiset aukioloajat. Hautausmaan profiilisivulla on myös painike, jota painamalla hakutyökalun käyttäjä pääsee hautausmaan karttasivulle. Tältä sivulta käyttäjä voi etsiä hautapaikkaa.



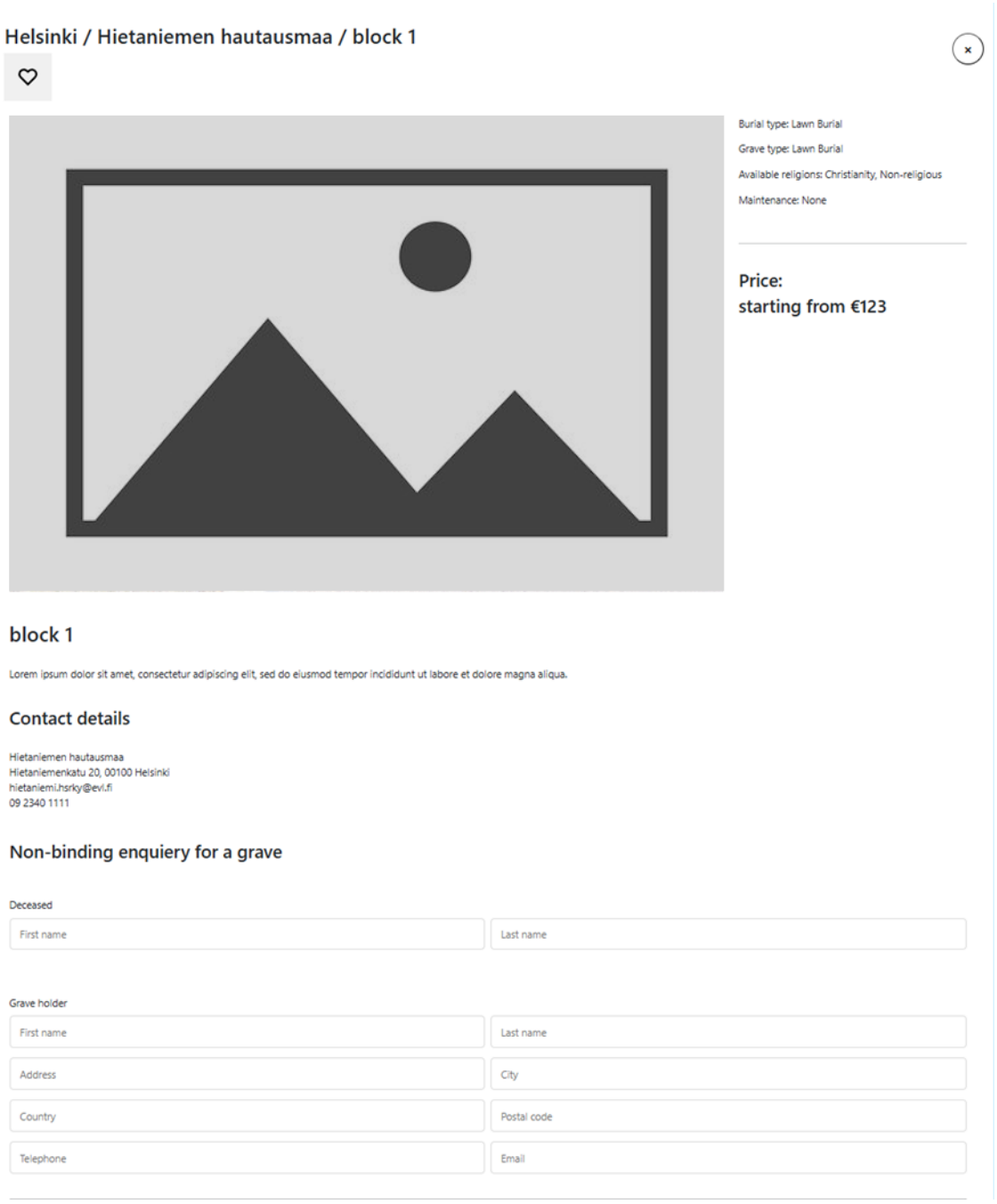

Kuva 3. Hautausmaan profiilisivu.

Hautausmaan hakemisen kartan tai hakuparametrien avulla voi myös ohittaa kokonaan, jos hakutyökalun käyttäjä valitsee jonkin ohjelman käyttöliittymän etusivulla listatuista hautausmaista (ks. Kuva 4). Hautausmaat on listattu käyttäjän maantieteellisestä sijainnista käsin niin, että etäisyydeltään lähin hautausmaa on listassa ensimmäisenä. Jos käyttäjä valitsee hautausmaan käyttöliittymän etusivulta, tapahtuma ohittaa hautausmaan ponnahdusikkuna-profiilisivun ja ohjaa käyttäjän suoraan hautausmaan karttasivulle. Jos käyttäjä on laittanut hakukenttään parametrejä, ne ovat voimassa haun seuraavissa vaiheissa, vaikka käyttäjä valitsisikin hautausmaan suoraan käyttöliittymän etusivulla listatuista hautausmaista.




Kuva 4. Etusivun hautausmaaehdotusten syöte.

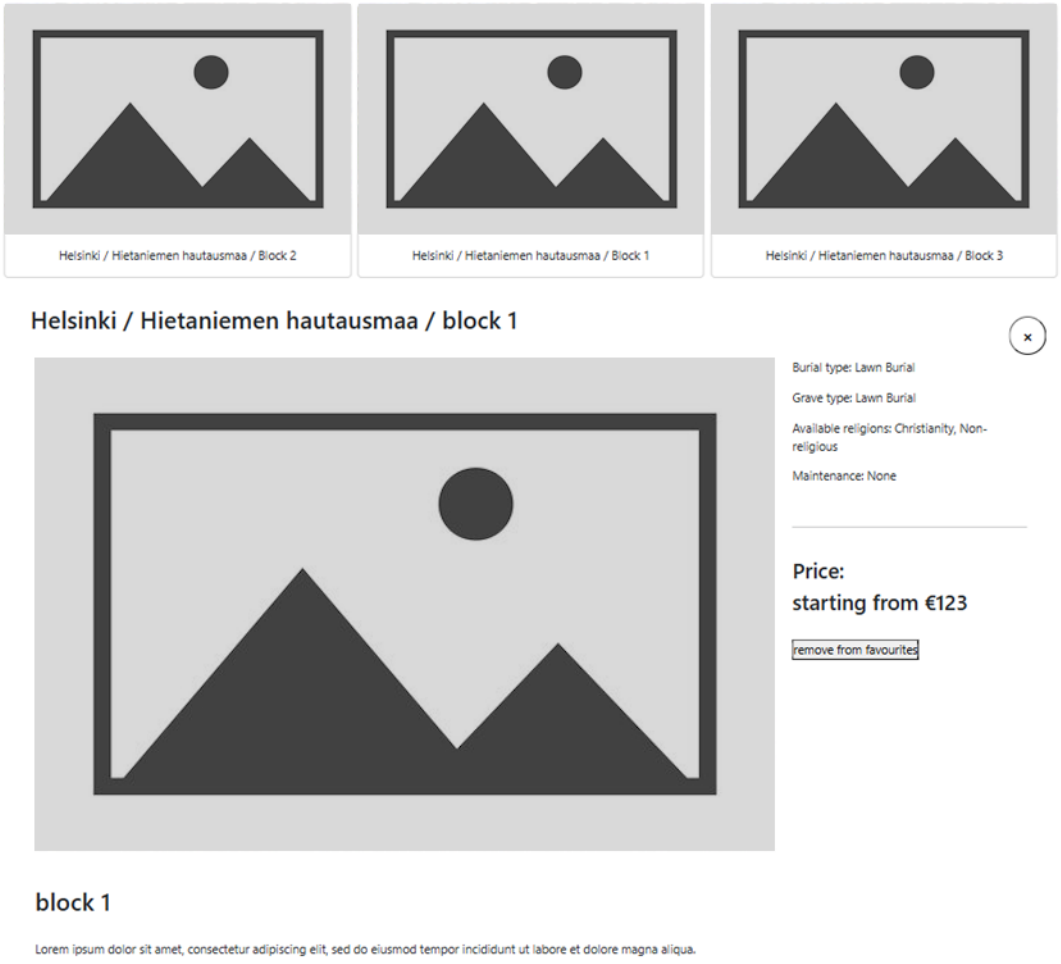
Hautakorttelihaun kolmas ja viimeinen vaihe tarkentaa hakutyökalun interaktiivisen kartan näkymän valittuun hautausmaahan, jossa käyttäjä voi selata hautausmaan kortteleita. Hakuparametrit, kuten uskonto ja hautaustyyppi, ovat edelleen muokattavissa karttasivun vasemmalla puolella sijaitsevissa monivalintalomakkeissa. Lisäksi hakutyökalun hakuvaihtoehtoihin lisätään uusi parametri haudan huoltovaihtoehdoille, joka voi olla joko olematon, vapaaehtoinen tai kaikenkattava. Kun käyttäjä valitsee korttelin listalta tai kartalta, aukeaa kuvan 5 esittämä ponnahdusikkuna, josta nähdään korttelin tarkemmat tiedot. Korttelin tarkempiin tietoihin kuuluu korttelin numero, kuva korttelista, hautausmahdollisuudet, sallitut uskonnot, lyhyt tiivistelmä ja hinta-arvio. Lisäksi sivulta löytyy yhteystiedot hautausmaan toimistoon sekä lomake, jonka täyttämällä käyttäjä voi tehdä korttelia koskevan kyselyn hautausmaalle suoraan sivulta käsin.




Kuva 5. Hautausmaakorttelin profiilisivu.

Hautausmaakorttelin profiilisivulla on myös sydänsymbolilla muotoiltu painike, jota painamalla profiili tallennetaan käyttäjän suosikkeihin. Käyttäjän tallentamia hautausmaa kortteleita voi selata suosikit-välilehdellä, joka listaa tallennetut hautausmaat siinä järjestyksessä, missä ne on lisätty suosikiksi.

Kuvassa 6 nähdään, miten hautausmaakorttelin profiili saadaan avattua myös suosikit-sivulta. Käyttäjä voi halutessaan poistaa kohteen suosikeista Hautausmaakorttelin profiilissa olevan painikkeen avulla.




Kuva 6. Korttelin profiili avattuna suosikit-välilehdellä.

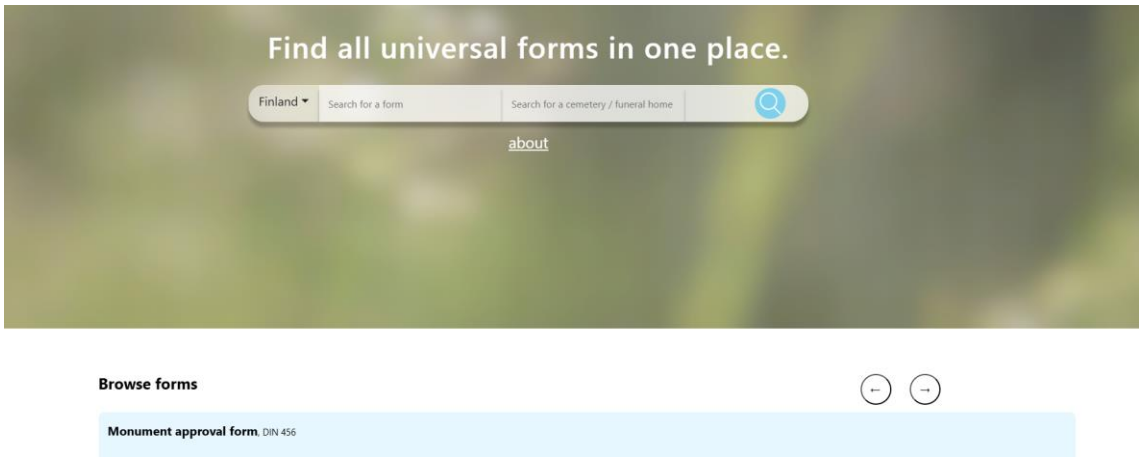
Hautakorttelihaun tarkoituksena on olla mahdollisimman yksinkertainen ja käyttäjäystävällinen palvelu ja tarjota silti monipuolinen ja tehokas hakutyökalu. Interaktiivinen kartta toimii erinomaisesti alla listattujen ehdotusten kanssa, mutta kumpaakin voi käyttää myös itsenäisesti.

## 3.3 Universaalien lomakkeiden täyttötyökalu

Tässä insinöörityössä kehitettävän ohjelman toisena työkaluna on universaalien hautausalan lomakkeiden portaali. Universaalilla lomakkeella tarkoitetaan lomakkeita, jotka voidaan täyttää riippumatta siitä, missä päin maailmaa ollaan. Käytännössä kyse on niistä hautausalan lomakkeista, joiden sisältö ei riipu alueellisesta lainsäädännöstä tai byrokratiasta. Lomakkeet koskevat siis ainoastaan hakemuksia, jotka eivät tarvitse alue- tai toimivaltakohtaista tietoa.

Tässä insinöörityöprosessissa on suunniteltu alusta, johon voidaan koota mahdollisimman paljon hautausalan lomakkeita, joiden avulla sijainnista riippumatta vastataan samoihin kysymyksiin. Palvelun tarkoituksena on helpottaa eri hautausalan toimijoiden, kuten kivityö- ja rakennusyritysten toimintaa niiden asioidessa hautausmaiden kanssa. Työkalusta löytyy esimerkiksi lomake, joka sisältää kaiken sen tiedon, mitä hautausmaa tarvitsee kivityöyritykseltä, ennen kuin he voivat aloittaa monumentin tai hautakiven rakentamisen.

Kuvassa 7 oleva lomaketyökalun ensimmäinen sivu muistuttaa hyvin paljon edellisen työkalun etusivua (ks. Kuva 1). Siellä on hakukenttä, johon syötetään sekä toivotun lomakkeen että hautausmaan nimet. Molemmat hakukentät ehdottavat käyttäjän hakuun sopivia vaihtoehtoja, joista valitsemalla kenttä täyttyy automaattisesti.




Kuva 7. Lomakkeiden täyttötyökalun etusivu.

Lisäksi yleisimmät lomakkeet on listattu hakukentän alapuolelle. Kun käyttäjä syöttää tarvittavat tiedot aukeaa seuraava ikkuna, jossa valittu lomake on jo valmiiksi täytettävissä. Hautausmaiden yhteistiedot ovat jo valmiiksi järjestelmässä, ja ohjelma täyttää lomakkeeseen valitun hautausmaan tiedot automaattisesti. Tässäkin ikkunassa käyttäjä voi vielä vaihtaa hautausmaata ja lomaketta. Käyttäjä voi lähettää täytetyn lomakkeen eteenpäin ohjelman sisällä, joka automaattisesti jakaa lomakkeen valitun hautausmaan kontaktin sähköpostiosoitteeseen.

# 4 Hautaohjelman hakufunktiot

## 4.1 Käyttäjän syötevalinta

Luvussa 4 esitellään hakuohjelman käyttämiä hakufunktioita, ja niiden rakennetta. Kun tässä insinöörityössä kehitettävä ohjelma haluaa käyttäjältä kirjoitetun syötteen, tapahtuu se syötevalinta-funktion avulla. Syötettä tarvitaan sanallisen hakemisen yhteydessä ja molemmat ohjelman työkalut käyttävät sitä. Ohjelman muuttujat käsitellään lähes poikkeuksetta React-kirjaston useState-hookin avulla. UseState on Reactiin sisäänrakennettu hook, joka helpottaa muuttujien arvojen hallitsemista. Esimerkkikoodista 1 näkee, miten useState tarvitsee kaksi parametriä, state-muuttujan ja sitä käsittelevän funktion.

```
const [example, setExample] = useState("");
```

Esimerkkikoodi 1. Esimerkki, jossa luodaan vakio, joka käyttää React-kirjaston useState-hookia.

Koska suuri osa ohjelman muuttujista on useStatea käyttäviä vakioita, täytyy tämä ottaa huomioon myös käyttäjän syötevalinnan kanssa. Funktio ottaa kutsuttaessa vastaan neljä parametriä. Parametreihin kuuluu useStaten muuttujaa käsittelevän funktio, state-muuttuja, tekstikentän paikkamerkin teksti ja hakukentän tyyli.

Hakukentän tyyli määrittää, mitkä hakukentän kulmista on pyöristettyjä. Tällä tavalla samaa hakukenttäfunktiota voidaan käyttää muiden hakufunktioiden

joukkoon upottamalla tai itsenäisenä hakukenttänä. Kaikki ohjelman tekstihakukentät käyttävät samaa funktiota, vaikka ne näyttävätkin erilaisilta ja niitä käytetäänkin hyvin erilaisissa ympäristöissä.

Esimerkkikoodista 2 näkee, mitä funktio tekee jokaisella annetulla parametrillä.

```
function InputSearch({ setInputValue, value, placeholder, style }) {

  return (
    <input
      type="text"
      onChange={(event) => {
        const inputValue = event.target.value;
        setInputValue(inputValue);
      }}
      value={value}
      placeholder={placeholder}
      className={style}
    />
  );
}
```

Esimerkkikoodi 2. JavaScript-funktio, joka on vastuussa käyttäjän syötteen käsittelystä. Funktio koostuu React-kirjaston input-komponentista, joka asettaa sille syötetyn state-muuttujan käyttäjän syötteen mukaiseksi.

## 4.2 Käyttäjän monivalinta

Kuten syötevalinta, monivalintakin käyttää useState-hookia muuttujien hallitsemisessa. UseState-hookin kahden parametrin lisäksi monivalinnan funktio ottaa vastaan listan monivalinnan vaihtoehdoista joko JSON-tiedostona tai JavaScriptin taulukkomuodossa. Monivalinnan funktio muodostaa vaihtoehdoista kartan JavaScript-kielen map-metodilla. Kuten esimerkkikoodissa 3 havainnollistetaan, kartan jäsenissä on vaihtoehdon valintanappi ja valinnan nimi. Lisäksi Jokaisessa syöttökentässä on erillinen funktio, joka huolehtii siitä, että vain yksi monivalinnan vaihtoehto voi olla valittuna samaan aikaan.

```
export default function MultipleChoice({title, options, setInputValue,
value}) {
  const [selected, setSelected] = useState(value);

  function onChange(i) {
    setSelected((prev) => (i === prev ? null : i));
    setInputValue(i)
  }

  return (
    <div className="multiHolder">
        <span className="H3">{title}</span>
         {options.map(item => (
        <div className="multiContent">
        <label key={item.value}>
          <span className="multiBtnHolder">
          <input
            type="radio"
            className="multiBtn"
            checked={item.value === selected}
            onChange={() => onChange(item.value)}
          />
          </span>
          <span className="body">{item.value}</span>
        </label>
        </div>
                  ))}
    <hr/>
    </div>
  );
}
```

Esimerkkikoodi 3. JavaScript-funktio, jonka avulla voidaan luoda monivalintakysymyksiä. Ohjelma tarvitsee toimiakseen muuttujan sekä listan vaihtoehdoista valituille arvoille. Funktio koostuu React-kirjaston input -komponentista, jolle on annettu tyypiksi radio (”valinta”), sekä funktiosta, joka pitää huolen, että vain yksi valinnoista voidaan tehdä kerrallaan.

## 4.3 Käyttäjän liukusäädinvalinta

Hautausmaaohjelman numeraalisten muuttujien, kuten etäisyyden määrittämisen yhteydessä, liukusäädin on useimmiten käyttäjäystävällisin ratkaisu. Kuten muutkin hakufunktiot, liukusäädin tarvitsee toimiakseen state-muuttujan ja sitä ohjaavan funktion. Liukusäädinfunktio sisältää React-kirjaston input -komponentin, jonka tyypiksi on asetettu range (”alue”), joka antaa arvonsa eteenpäin määritellylle muuttujalle. Koska liukusäädintä käytetään toistaiseksi ainoastaan etäisyyden määrittämisen yhteydessä, on sen alue säädetty staattiseksi välille 5–200. Jos funktion käyttökohteet laajenevat tulevaisuudessa, voidaan ääriarvot helposti vaihtaa funktion kutsumisen yhteydessä määriteltäviksi propseiksi.

```
function SliderSearch({ setInputValue, value, name }) {

  return (
    <>
    {name}: {value} km <br/>
    <input
      type="range"
      onChange={(event) => {
        const inputValue = event.target.value;
        setInputValue(inputValue);
      }}
      value={value}
      className="slider"
      max={200}
      min={5}
    />
    </>
  );
}
```

Esimerkkikoodi 4. JavaScript-funktio, jonka avulla voidaan luoda liukusäädinvalinta. Ohjelma tarvitsee toimiakseen state-muuttujan, sitä hallitsevan funktion sekä haluttaessa otsikon. Liukusäätimen alue on määritelty staattisesti välille 5–200.

## 4.4 Hakufunktiot käytännössä ohjelmassa

Kehitetyssä ohjelmassa hakutyökaluja käytetään lähinnä datan suodattamisen yhteydessä. Tyypillisesti hakutyökaluissa käytetään JavaScript-ohjelmointikielen filter-metodia. Metodin avulla voidaan tarkistaa, löytyykö jonkin hakufunktion muokkaaman muuttujan arvoa jonkin JSON-tiedoston sijainnista. Jos halutaan suodattaa ohjelmassa hautausmaadataa monivalinta-aliohjelmalla esimerkiksi valitun uskonnon perusteella, se tapahtuu esimerkkikoodi 5 mukaisesti.

```
props.geojson.features
     .filter((data) => {
          const titleReligion = data.properties.religion.toLowerCase();
          const inputReligion = cityValue.toLowerCase();

     return titleReligion.includes(inputReligion);})
```

Esimerkkikoodi 5. Esimerkki siitä, miten monivalinnan muokkaamaa muuttujaa voidaan käyttää datan suodattamisessa JavaScript-ohjelmointikielen filter-metodin avulla.

# 5 Hautausmaaohjelman modulaarisuus

## 5.1 Modulaarinen suunnittelufilosofia

Luvussa 5 käsitellään, miten ohjelman työkalujen toiminnallisuus koostuu komponenteista, ja komponentit koostuvat omista alafunktioistaan. Kaikki kehitetyn ohjelman osa-alueet ja työkalut noudattavat samanlaista modulaarista suunnittelufilosofiaa.

Tämä ennalta suunniteltu modulaarinen lähestymistapa kaikkeen ohjelmoimiseen mahdollistaa lähes kaikkien funktioiden uudelleenkäyttämistä eri tilanteissa. Se mahdollistaa ohjelman työkalujen ja ominaisuuksien laajentamisen ilman että sen skaala kasvaa.

## 5.2 React Props

Aliohjelmat ja funktiot tarvitsevat lähes poikkeuksetta erilaisia arvoja toimiakseen oikein. Jotta niitä voitaisiin uudelleen käyttää erilaisissa yhteyksissä, voidaan käyttää propseja kiinteästi ohjelmoitujen muuttujien sijaan. React-kirjaston yhteydessä propilla tarkoitetaan argumenttia, joka annetaan komponentille sen kutsumisen yhteydessä. Propsien avulla komponentti voi sivuuttaa joidenkin arvojen julistamisen funktion sisällä, kunhan ne määritellään sitä kutsuttaessa. Selkeyden vuoksi tässä työssä viitataan propseihin niiden englanninkielisen monikon taivutuksen mukaisesti (prop/props).

Esimerkkikoodissa 6 näkyy Suomen hautausmaita koskevan karttasivun koodi. Sivun koodi on hyvin lyhyt ja yksinkertainen: sen sijaan että jokaiselle valtiolle tehtäisiin oma ja monimutkainen karttasivu, voidaan käyttää samaa karttasivun funktiota, jonka sisällä kartta ja sen ominaisuudet on jo määritelty. Tämä on mahdollista, koska funktioon ei ole kovakoodattu karttadataa, vaan se ottaa datan vastaan GeoJSON-nimisen propin muodossa. Propin nimen mukaisesti kaikki karttafunktiot vastaanottavat dataa GeoJSON-tiedostomuodossa. Ohjelman toiminnan kannalta ei siis ole olennaista, minkä valtion tai alueen data on

kyseessä, kunhan se on sille sopivassa muodossa. Tämä takaa alustalle loputtoman laajentamisen mahdollisuuden, jossa vain isännöivän palvelimen suorituskyky on rajana.

```
export default function Finland() {

  return (
    <>
      <div className={styling.mapHolder}>
        <CityMap geojson={finland}/>
      </div>
    </>
  );
}
```

Esimerkkikoodi 6. Suomen hautausmaita koskevan karttasivun funktio. Funktio kutsuu karttafunktion ja syöttää sille GeoJSON-tiedostomuotoisen datasetin, joka pitää sisällään Suomen hautausmaat.

# 6 Hautausmaaohjelman lomakkeet

## 6.1 Ohjelman lomakkeiden modulaarisuus

Luvussa 6 käsitellään lomakkeiden täyttötyökalun ohjelmallista rakennetta. Insinöörityössä kehitetty universaalien lomakkeiden täyttötyökalu tarvitsee toimiakseen funktion lomakepohjalle. Lomakepohjan funktio, noudattaa samaa modulaarisuutta kuin monimutkaisemmatkin asiat. Lomakkeiden rakenne lähtee liikkeelle yksittäisestä lomakkeen kysymyskomponentista, joka ottaa vastaan yhden kysymyksen, jonka pitää olla muodossa ”string” eli teksti.

```
type FormQuestionProps = {
question: string
}
export function FormQuestion({ question }: FormQuestionProps) {
return (
     <>
     <div className={styling.inputBox}>
          <input
               type="text"
               className={styling.field}
               placeholder={question} required
          />
     </div>
</>
)
}
```

Esimerkkikoodi 7. TypeScript-komponentti, joka vastaanottaa yksittäisen kysymyksen, jonka se sijoittaa css-muotoiltuun syöttökenttään.

Lomakepohjan funktion hierarkiassa seuraavana on lomakekomponentti, joka kokoaa kysymysfunktioista lomakkeen sille annettujen kysymysten mukaisesti. Kuten esimerkkikoodissa 8 näytetään, lomakefunktio tarvitsee kaksi syötettä toimiakseen. Nämä kaksi syötettä ovat lomakkeen kysymyksiä koskevan väliotsikon (esimeriksi vainajan nimi), sekä listan kysymyksiä.

```
const Form = (props) => {
return (
<>
<span className="body">{props.formName}</span>
     <section className={styling.Contact}>
          <Row sm={1} lg={2} className="g-2">
           {props.questions.map(item => (
             <Col key={item.id}>
               <FormQuestion {...item} />
             </Col>
           ))}
         </Row><br/>
    </section>
</>
  )
}
```

Esimerkkikoodi 8. JavaScript-funktio, joka vastaanottaa väliotsikon sekä listan kysymyksiä. Funktio kokoaa sitten kysymykset väliotsikon alapuolelle kahden kysymyksen riveihin. Kysymyskentät muodostuvat, kun ohjelma tekee taulukon kysymyskomponenteista, joista jokainen saa arvokseen yhden kysymyksen kysymyslistalta.

Lomakepohjan seuraavassa vaiheessa lomakekomponenttien avulla voidaan muodostaa valmiita lomakkeita. Esimerkkikoodista 9 näkee, miten lomake on

rakennettu kahdesta lomakekomponentista. Molempiin komponentteihin on syötetty eri määrä toisistaan poikkeavia kysymyksiä. Komponenttien avulla voidaan muodostaa toisistaan poikkeavia lomakkeita.

```
const Form = (props) => {
export default function FormTest() {
     return (
     <>
     <h1>Esimerkkilomake</h1> <br/>
     <Form formName={"Haudan omistaja"} questions={[
          { "question": "etunimi"},
          { "question": "last name"},
          { "question": "address"},
          { "question": "city"},
          { "question": "postal code"},
          { "question": "telephone"},
          { "question": "email"}
     ]} />
     <hr />
     <Form formName={"Vainaja"} questions={[
          { "question": "first name"},
          { "question": "last name"},
          { "question": "date of birth"},
          { "question": "date of death"},
          { "question": "city"},
          { "question": "religion"}
     ]} />
     </>
);}
```

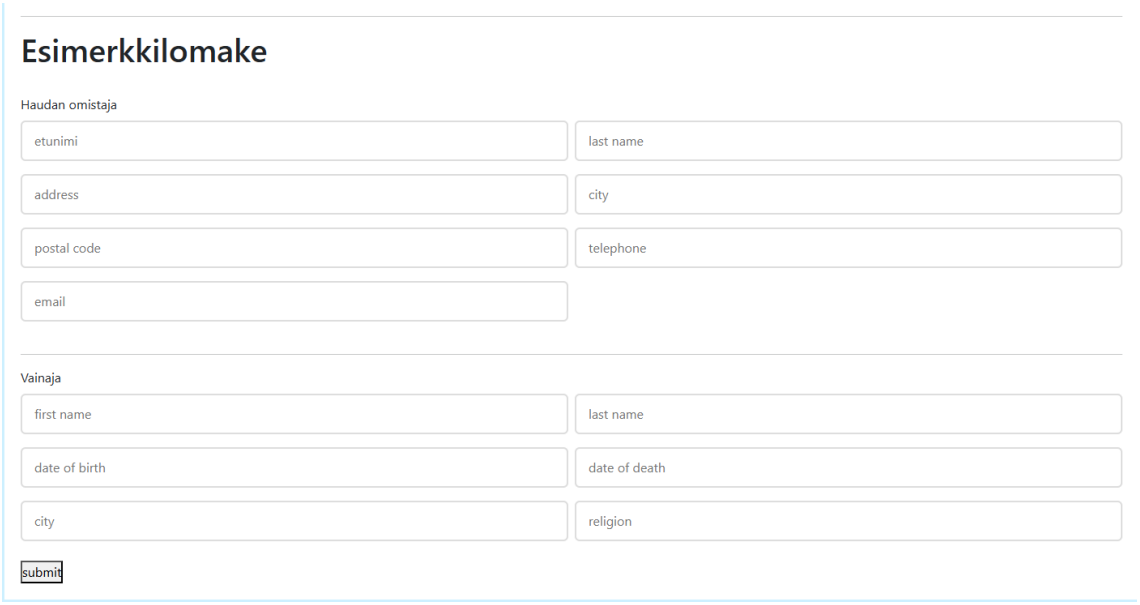
Esimerkkikoodi 9. JavaScript-komponentti, jossa on rakennettu käyttäjälle täytettävä esimerkkilomake, käyttäen lomakefunktioita.

## 6.2 Ohjelman lomakesivun rakenne

Lomakkeen rakentamisen jälkeen se pitää vielä saada näkyviin käyttäjälle. Tämä tapahtuu lomakkeiden pääohjelmassa. Täytettävien kysymysten lisäksi lomakesivu tarvitsee valitun hautausmaan tiedot.

Hautausmaatietojen vaihtaminen tapahtuu koodin sisällä samalla tavalla kuin lomakkeenkin. Pääohjelmassa on funktio, joka vaihtaa muuttujan arvon johonkin id-numeroon. Id-numero syötetään esitäytetyn lomakkeen aliohjelmaan, joka vaihtaa lomakkeen tiedot id-numeroon yhdistetyn hautausmaan mukaisiksi.

Kuvasta 8 näkee, miltä lomake näyttää, kun se näytetään sivulla.




Kuva 8 Esimerkki komponenteilla valmistetusta lomakkeesta

Ohjelman rakenteen ansiosta samoilla työkaluilla voi siis tehdä loputtoman määrän eri kysymyksiä sisältäviä lomakkeita eri aihealueista.

# 7 Hautausmaaohjelman datan käsittely

## 7.1 GeoJSON-tiedoston rakenne

Luvussa 7 käsitellään, miten dataa säilytetään GeoJSON-tiedostoissa ja miten tiedostojen dataa käytetään. Karttasivut vastaanottavat dataa GeoJSON-tiedostomuodossa. Siksi on tärkeä ymmärtää, miten ohjelma käsittelee GeoJSON-muotoista dataa. Liitteessä 2 esitellään, miten GeoJSON-tiedostojen tiedot jakautuvat erikseen sijainti- ja asiatietoihin. Asiatiedot jaetaan alaryhmään features.properties ja sijaintitieto alaryhmään features.geometries.

Kun ymmärretään, miten data on säilötty GeoJSON-tiedoston sisällä, dataa voidaan noutaa tiedostosta esimerkkikoodi 10 kartan mukaisesti.

```
{props.geojson.features
     .map(item => (
          data.properties.cemetery,
          data.properties.city,
          data.properties.imgUrl,
          data.geometry.coordinates[0],
          data.geometry.coordinates[0]
     ))}
}
```

Esimerkkikoodi 10. Esimerkki siitä, miten JavaScript-ohjelmointikielen map-metodilla voidaan luoda taulukko GeoJSON-tiedostosta, joka näyttää kunkin elementin dataa.

Jotta ohjelma osaa yhdistää asiatiedot oikeaan sijaintiin, pitää data järjestää sellaiseen muotoon, että sitä on mahdollisimman helppo lukea. JSON-tiedosto muistuttaa tavallista JavaScript-taulukkoa. Yksi tiedosto säilyttää monta jäsentä, joilla jokaisella on jokin määrä muuttujia, joiden arvo voidaan määritellä erikseen. Jos JSON-tiedostoilla on identtisesti nimetyt muuttujat, voidaan niiden arvoja renderöidä ja vertailla helposti. JavaScript-ohjelmointikieleen on sisäänrakennettu lukuisia metodeja, jotka auttavat datan hallitsemisessa ja esittämisessä. Seuraavaksi käydään läpi ohjelman kannalta keskeisimpiä metodeja.

## 7.2 Map-metodi

JSON-tiedostot voivat sisältää monen eri elementin dataa, minkä takia sen renderöiminen vaatii jonkin metodin. Lähes poikkeuksetta ohjelman renderöidessä dataa JSON-tiedostosta se tehdään JavaScript-ohjelmointikielen map-metodin avulla. Esimerkkikoodissa 11 näkee, miten metodia voidaan käyttää GeoJSON-tiedoston arvojen renderöimisessä. Se on täydellinen työkalu JSON- ja GeoJSON-tiedostomuodoissa olevien tiedostojen datan renderöimiseen.

```
{props.geojson.features
.map(item => (
                  <Col key={item.properties.cemetery}>
                        <CemeteryCard {...item.properties} />
                  </Col>
                ))}
```

Esimerkkikoodi 11. Esimerkki JavaScript-ohjelmointikielen map-metodin käytöstä, jossa JSON-tiedoston jäsenet renderöidään erikseen määritellyn komponentin avulla.

## 7.3 Filter-metodi

Ennen JSON-datan kartoittamista map-metodilla sitä voidaan myös suodattaa JavaScript-ohjelmointikielen filter-metodin avulla. Esimerkkikoodissa 12 suodatetaan karttaan merkittäviä kohteita GeoJSON-tiedoston jäsenten properties.city arvon ja käyttäjän syötteen perusteella määritetyn kaupungin arvon perusteella. Jos GeoJSON-tiedoston jäsen ei sisällä syötteen kaupungin arvoa, sitä ei renderöidä sivulle.

```
{props.geojson.features
.filter((data) =>
     {
     const titleCity = data.properties.city.toLowerCase();
     const inputCity = cityValue.toLowerCase();

     return titleCity.includes(inputCity;
     })
```

Esimerkkikoodi 12. Esimerkki JavaScript-ohjelmointikielen filter-metodin käytöstä. Julistetaan kaksi vakiota, joista toisen arvo on hakutyökalun muuttuja ja toinen on JSON-tiedoston jäsenen vastaavan parametrin arvo. Jos jäsenen parametrin arvo ei sisällä hakutyökalun määrittelemää muuttujaa, suodatetaan se pois.

Käyttäjän asettamien suodattimien ja syöttöjen lisäksi ohjelma pystyy suodattamaan dataa esimerkiksi sijainnin avulla. Käyttäjän sijainti on joko käyttäjän laitteen maantieteellinen sijainti tai valitun kaupungin keskikohta riippuen käyttäjän tekemistä valinnoista. Ohjelma laskee kahden sijainnin välisen etäisyyden esimerkkikoodin 13 mukaisesti.

```
{props.geojson.features
.filter((data) =>
     {
     const distance = (Math.acos(
          Math.sin(toRadius(data.geometry.coordinates[1])) *
          Math.sin(toRadius(_location.state.coordsX)) +
          Math.cos(toRadius(data.geometry.coordinates[1])) *
          Math.cos(toRadius(_location.state.coordsX)) *
          Math.cos(toRadius(_location.state.coordsY) –
          toRadius(data.geometry.coordinates[0])),
     ) * earthRadius) * 0.001;

     return distance <= distanceValue);
     })
```

Esimerkkikoodi 13. Esimerkki JavaScript-ohjelmointikielen filter-metodin käytöstä. Julistetaan vakio, jonka arvoksi lasketaan maantieteellisen etäisyyden kaavalla etäisyys JSON-tiedoston jäsenen koordinaattien ja käyttäjän sijainnin välillä. Jos etäisyys on suurempi kuin käyttäjän asettama maksimietäisyys, suodatetaan jäsen pois ennen renderöintiä.

Suodatin vertaa käyttäjän ja kohteen sijainnin välistä etäisyyttä käyttäjän valitsemaan maksimietäisyyteen, joka ohjelman sisällä määritetään muiden suodattimien yhteydessä sijaitsevalla liukusäätimellä.

## 7.4 Sort-metodi

Esitetyn datan järjestystä voidaan myös muuttaa tiedostossa olevasta järjestyksestä, ilman JSON-tiedoston muokkaamista JavaScript-ohjelmointikielen sort-metodin avulla. Sort-metodia voidaan käyttää numeroiden järjestämiseen suurimmasta pienempään, voidaan siis laskea GeoJSON-tiedoston sijaintien etäisyyttä käyttäjästä ja vertailla sitten jäsenten etäisyyksiä toistensa kanssa. Esimerkkikoodi 14 käyttää sort-metodia näyttääkseen käyttäjälle hakutuloksia järjestyksessä lähimmästä hautausmaasta kaukaisimpana olevaan hautausmaahan.

```
{props.geojson.features
.sort((a, b) =>
     {
     const distA = (Math.acos(
          Math.sin(toRadius(a.geometry.coordinates[1])) *
          Math.sin(toRadius(location.latitude)) +
          Math.cos(toRadius(a.geometry.coordinates[1])) *
          Math.cos(toRadius(location.latitude)) *
          Math.cos(toRadius(location.longitude) –
          toRadius(a.geometry.coordinates[0])),
     ) * earthRadius)

     const distB = (Math.acos(
          Math.sin(toRadius(b.geometry.coordinates[1])) *
          Math.sin(toRadius(location.latitude)) +
          Math.cos(toRadius(b.geometry.coordinates[1])) *
          Math.cos(toRadius(location.latitude)) *
          Math.cos(toRadius(location.longitude) –
          toRadius(b.geometry.coordinates[0])),
     ) * earthRadius)

     if (distA > distB) return 1;
     if (distA < distB) return -1;
     return 0;
     })
```

Esimerkkikoodi 14. Esimerkki JavaScript-ohjelmointikielen sort-metodista, joka järjestää JSON-tiedoston jäsenet etäisyyden perusteella lyhyimmästä pisimpään. Metodi laskee aina kahden jäsenen etäisyyden ja renderöi niistä alhaisemman numeron. Sen jälkeen ohjelma vertaa suurempaa numeroa seuraavaan jäseneen. Sama toistuu, kunnes kaikki jäsenet on renderöity sivulle.

# 8 Hakutyökalun interaktiivinen kartta

## 8.1 Leaflet-kartan alustus

Luvussa 8 käsitellään interaktiivisen kartan rakennetta ja sen kehityksen kannalta olennaisia kirjastoja. Tämän insinöörityöprosessin aikana tuotetun ohjelman interaktiivinen kartta on toteutettu Leaflet-kirjaston avulla. Kartan alustaminen on toteutettu Leaflet-kirjaston omaa pikaopasta (Leaflet quick start guide 2010–2025) seuraten. Suunnittelijan pitää aluksi määritellä korkeus ja leveys kartalle ohjelman css-tiedostossa. Sen jälkeen kartta voidaan määritellä esimerkkikoodin 15 mukaisesti.

Jotta leaflet voisi renderöidä kartan karttaohjelman käyttöliittymään, tarvitaan siihen tileLayer, joka sisältää karttapiirustuksen muodostamiseen tarvittavat karttalaatat.

Esimerkkikoodissa 15 nähdään, miten tileLayer määritetään OpenStreetMap kartaksi. Lisäksi karttaan pitää asettaa oletusnäkymä, joka kertoo mihin ja kuinka lähelle kartta kohdistaa, kun sivun lataa. Esimerkkikoodin 15 kartta on asetettu kohdistamaan edellisessä vaiheessa valitun kaupungin sijaintiin.

```
mapRef.current = leaflet.map("map").setView([_location.state.coordsX,
_location.state.coordsY], 13);
    leaflet.tileLayer("https://tile.open-
streetmap.org/{z}/{x}/{y}.png", {
      attribution:
        '&copy; <a href="http://www.openstreetmap.org/copyright">Open-
StreetMap</a>',
    }).addTo(mapRef.current);
```

Esimerkkikoodi 15. Esimerkki OpenStreetMap kartan alustamisesta Leaflet-kirjaston avulla.

## 8.2 Leaflet Marker Cluster

Hakutyökalun kartan alustamisen jälkeen halutaan kartalle lisätä ohjelman kannalta olennaiset merkit ja elementit. Jotta kartta pysyisi selkeästi luettavana, perustetaan ensiksi klusteriryhmä React Cluster Group -kirjaston avulla, jonne merkit voidaan sitten lisätä. Klusteriryhmän alustaminen tapahtuu esimerkkikoodin 16 mukaisesti. Klusteriryhmä määritetään vakiolle ”markers”, jotta jatkossa sen käyttäminen olisi yksinkertaisempaa. Koska ohjelma ei käytä kirjaston vakiomerkintöjä pitää klusterin ”class” eli luokka määritellä erikseen käyttämään mukautettua css-muotoilua.

Kartan klusteriryhmät määritetään myös näyttämään numerolla, kuinka monta jäsentä siihen kuuluu. Se tehdään JavaScript-ohjelmointikielen getChildCount-funktion avulla.

```
const markers = leaflet.markerClusterGroup({
      iconCreateFunction: function (cluster) {
        return leaflet.divIcon({
          html: '<div class="cluster-div">' + cluster.getChildCount()
+ '</div>'
        })
       }
    })
```

Esimerkkikoodi 16. Esimerkki siitä, miten klusteriryhmä määritetään React Cluster Group-kirjaston avulla.

## 8.3 GeoJSON-datan lisääminen kartalle

Tämä alaluku 8.3 käsittelee GeoJSON-datan lisäämistä karttalle. Kartan alustamisen ja klusteriryhmän määrittämisen järkeen, karttaohjelma lisää propsina syötetyn GeoJSON-tiedoston kartalle. Tämä saadaan aikaan lisäämällä uusi ”layer” eli kerros karttaan. Leaflet-kirjastoon on sisäänrakennettu leaflet.geoJSON-funktio, joka yksinkertaistaa datan lisäämistä kartalle huomattavasti.

Esimerkkikoodista 17 nähdään, kuinka sen sijaan, että GeoJSON-data lisättäisiin suoraan karttaan Leaflet-kirjaston metodilla addTo(mapRef.current), lisätään se alustettuun klusteriryhmään Leaflet-kirjaston metodilla addTo(markers).

```
layer = leaflet.geoJSON(props.geojson,
     {
          onEachFeature: function (feature, layer) {
     }
     })
     .addTo(markers);
     layer.on("click", markerOnClick);
```

Esimerkkikoodi 17. Geojson-tiedoston lisääminen klusteriryhmään. GeoJSON-tiedoston merkit määrätään myös suorittamaan funktio nimeltä markerOnClick, kun niitä painetaan.

Lisäksi GeoJSON-tiedoston merkeille määritellään funktio, joka suoritetaan sitä painettaessa. Funktiota käytetään silloin, kun halutaan käsitellä kartalla olevan merkin sisältämiä asiatietoja. Kun GeoJSON-data on lisätty onnistuneesti klusteriryhmään, se voidaan lisätä karttaan esimerkkikoodin 18 metodilla.

```
markers.addTo(mapRef.current)
```

Esimerkkikoodi 18. Klusteriryhmän sisältävän vakion ”markers” lisääminen Karttaohjelman karttaan.

# 9 Hautausmaaohjelman karttasivun funktiot

## 9.1 Merkkien asiatiedon näyttäminen ponnahdusikkunoissa

Luvussa 9 käsitellään miten ohjelma saa asiatiedon kerättyä kartan merkeistä ja miten se näytetään käyttöliittymässä. Ohjelman modulaarisen suunnittelufilosofian seurauksena kaikki ponnahdusikkunat ja hakutulosten infokorit käyttävät samaa arkkitehtuuria. Tämän vuoksi ikkunoiden esittämä tieto koostuu erillisistä useState-muuttujista, joiden arvoa muutetaan valitun kohteen mukaisesti.

Esimerkiksi hautausmaiden profiili määritellään esimerkkikoodin 19 mukaisesti. Muuttujat kuten cem, city ja country ovat kaikki arvoltaan tyhjiä. Kun jokin hautausmaa valitaan joko valikosta tai kartalta, päivitetään muuttujat kartan kohteen mukaisesti ennen kuin kohteen profiili-ikkuna avataan.

```
<h2>{cem}</h2>
          <div className={styling.box}>
            <span className="name-sm">{city}</span>, <span class-
Name="desc">{country} </span> <br/><br/>
            <Row>
              <Col xs={9}>
                <img src={img} className={styling.img}></img>
<br/><br/>
                <h2>{cem}</h2>
                <p>{desc}</p>
                <br/>
                <h2>{"Opening hours"}</h2>
                <p>{hours}</p>
              </Col>
              <Col>
                <p>Available burial types: {types}</p>
                <p>Available religions: {religions}</p>
                <br/><br/>
              </Col>
            </Row>
          </div>
```

Esimerkkikoodi 19. Esimerkki siitä, miten hautausmaan profiili määritellään käyttäen erillisiä muuttujia.

Jotta tyhjien muuttujien tiedot voidaan muuttaa asianmukaisiksi, pitää dataa pystyä käsittelemään. Se, miten sijainnin asiatieto käytännössä saadaan kohteesta, eroaa hieman riippuen siitä, valitaanko kohde valikosta vai suoraan kartalta. Seuraavaksi selitetään, miten sijainnin asiatiedon noutaminen kartalta ja valikosta poikkeavat toisistaan.

Koska jokaisen hakutyökalun kartalla olevan kohteen tiedot poikkeavat toisistaan, pitää tieto saada suoraan sen sijainnin näyttävästä merkistä kartalla. Kun GeoJSON-tiedoston sisältämät merkit lisättiin klusteriryhmään, merkeille asetettiin funktio, joka suoritetaan silloin, kun merkkiä painetaan.

Kartalla olevaa elementtiä painettaessa kutsuttava funktio on esimerkkikoodin 20 funktio. Funktio tarvitsee toimiakseen kartalta valitun kohteen asiatiedot. Sillä on monta tehtävää. Ensiksi se jakaa GeoJSON-tiedoston kaksi alaryhmää muuttujiin attributes ja coordinates, jonka jälkeen se vaihtaa jokaisen tyhjän muuttujan arvon kohteen arvon mukaiseksi. Sen jälkeen se avaa hautausmaan hakutyökalun profiilin ponnahdusikkunan ja asettaa kartan keskittämään kohteen sijainnin koordinaattien mukaisesti.

```
function markerOnClick(e)
{
	var attributes = e.layer.feature.properties;
	var coordinates = e.layer.feature.geometry;

	setCity(attributes.city)
	setCountry(attributes.country)
	setHours(attributes.hours)
	setCem(attributes.cemetery)
	setDesc(attributes.desc)
	setTypes(attributes.types)
	setReligions(attributes.religion)
	setImg(attributes.imgUrl)

	setCoordsX(coordinates.coordinates[1])
	setCoordsY(coordinates.coordinates[0])

	setPop(true);

	mapRef.current.setView([coordinates.coordinates[1], coordi-
nates.coordinates[0]], 15);
}
```

Esimerkkikoodi 20. JavaScript-funktio, joka kutsutaan, kun käyttäjä valitsee kohteen kartalta. Funktio asettaa kaikki profiilin muuttujat kohteen mukaisiksi ja avaa profiilin ponnahdusikkunan.

Toiminnan kannalta ei ole haittaa, vaikka käyttäjä painaisi kartalta jotain toista elementtiä jonkin toisen hautausmaan profiili-ikkunan ollessa edelleen avoinna, koska funktio päivittää profiilin tiedot uuden hautausmaan elementtien mukaisesti. Käyttöliittymän kartan lisäksi hautausmaita voi valita sen alapuolella olevasta hautausmaavalikosta.

Hautausmaavalikon syöte renderöidään, eli lisätään näkyviin käyttöliittymään luvussa 7, Hautausmaaohjelman datan käsittely, mainittujen metodien läpi kulkemisen jälkeen map-metodin avulla. Hakutulosten asettelu on määritelty erillisessä funktiossa ”CemeteryCard”, joka sijoittaa tiedot niiden oikeille paikoille kortin asettelun mukaisesti. Esimerkkikoodista 21 nähdään miten valikon ”kortit” renderöidään ohjelmaan.

```
.map(item => (
     <Col key={item.properties.cemetery}>
     <button onClick={() => {
          pageOnClick(
               item.properties.cemetery, item.properties.city,
               item.properties.country, item.properties.hours,
               item.properties.desc, item.properties.types,
               item.properties.religion, item.properties.imgUrl,
               item.geometry.coordinates
               );}}
               className={styling.cardBtn}>
                         <CemeteryCard {...item.properties} />
                       </button>
                   </Col>
                 ))}
```

Esimerkkikoodi 21. Esimerkki siitä, miten ohjelma renderöi GeoJSON-tiedoston hautausmaat valikkoon JavaScript-ohjelmointikielen map-metodilla.

Lisäksi esimerkkikoodissa 21 havainnollistetaan, että hautausmaavalikon syötteen hakukortit on koodattu painikkeeseen, joka kutsuu funktion ”pageOnClick”. Funktio toimii samalla tavalla kuin kartalta valitsemisen yhteydessä. Keskeinen ero on se, että map-metodin ansiosta jokaisen syötteen kortin painikkeen laukaisema funktio pitää jo sisällään oikeat tiedot. Esimerkkikoodista 22 nähdään, miten elementin sijasta funktio ”pageOnClick” ottaa kohteen arvot suoraan muuttujina vastaan. Molemmat funktiot vaikuttavat samojen muuttujien arvoihin, joten molemmat voivat käyttää samaa ponnahdusikkunaa tiedon renderöimiseen.

```
function pageOnClick(cem, city, country, hours, desc, types, reli-
gions, imgUrl, coords) {
    setCity(city)
    setCountry(country)
    setHours(hours)
    setCem(cem)

    setDesc(desc)
    setTypes(types)
    setReligions(religions)
    setImg(imgUrl)

    setCoordsX(coords[1])
    setCoordsY(coords[0])
    setPop(true);
    console.log(imgUrl)

    mapRef.current.setView([coords[1], coords[0]], 15);
  }
```

Esimerkkikoodi 22. Funktio, joka kutsutaan, kun käyttäjä valitsee kohteen kartan alapuolella olevasta valikosta. Funktio asettaa kaikki profiilin muuttujat kohteen mukaisiksi ja avaa profiilin ponnahdusikkunan.

Nämä kaksi samankaltaista, mutta toisistaan poikkeavaa funktiota mahdollistavat sen, että sekä kartalla että valikossa listatut hautausmaat voivat käyttää samaa dataa. Tämä helpottaa ohjelman käyttämän datan valmistamista ja muokkaamista. Lisäksi se varmistaa, ettei hakutulosten ja kartan välillä ilmene ristiriitaisuuksia. Tämä hakutulosten ja kartan ristiriidattomuus tekee ohjelmasta luotettavamman.

## 9.2 Käyttäjän suosikkien tallentaminen selaimen muistiin

Hakutulosten tallentaminen on hyvin keskeinen ja tärkeä ominaisuus jokaisessa hakukoneessa. Kehitetyn ohjelman rakenne ei tue tällä hetkellä erillistä käyttäjän rekisteröintiä, jonka vuoksi tallennettuja hautakortteleita ei voida säilöä käyttäjäprofiilin kaltaisessa ympäristössä.

React-kirjasto kuitenkin tarjoaa työkalut käyttäjän selaimen paikallisen muistin hallitsemiseen. Tämä mahdollistaa käyttöliittymän luomisen, missä käyttäjä voi tallentaa haluamiaan profiileita laitekohtaiseen säiliöön ja tarkastella niitä Suosikit-välilehdellä. Esimerkkikoodista 23 näkee, miten hautausmaa datan tallentaminen suosikeihin tapahtuu. Funktio lisää avatun profiilin arvot state-muuttujan

nimettyihin arvoihin, jonka jälkeen muuttuja tallennetaan selaimen paikalliseen muistiin React-kirjaston useEffect-koukun avulla.

```
function handleSubmit() {
    {
      console.log("added to favourites")
      setFavs((currentFavs) => {
        return [
          ...currentFavs,
          {
            id: crypto.randomUUID(), index: props.index, block: block,
bType: bType, maintenance: maintenance, imgUrl: img, desc: desc, reli-
gion: religion, price: price, completed: false
          },
        ]
      })
    }
  }

  useEffect(() => {
    localStorage.setItem("FAVOURITES_BLOCK",JSON.stringify(favs))
  }, [favs])
```

Esimerkkikoodi 23. JavaScript-funktio, joka lisää aktiivisen profiilin arvot erilliseen state-muuttujaan. Sen jälkeen muuttuja ladataan käyttäjän selaimen paikalliseen muistiin mukautettuun sijaintiin ”FAVOURITES_BLOCK”.

Tallennettuja hautauspaikkojen profiileja voidaan selata erillisellä Suosikit-välilehdellä. Tallennetuttujen profiilien data saadaan paikallisesta selaimen muistista esimerkkikoodin 24 mukaisesti antamalla state-muuttujan arvoksi sen paikallisen muistin sijainti, jonne suosikit tallennetiin (”FAVOURITES_BLOCK”).

```
 const [favs, setFavs] = useState(() => {
        const localValue = localStorage.getItem("FAVOURITES_BLOCK")
        if (localValue == null) return []
        return JSON.parse(localValue)
    })
```

Esimerkkikoodi 24. Esimerkki siitä, miten selaimen paikallisessa muistissa olevaa dataa voidaan asettaa state-muuttujan arvoksi.

Koska käyttäjä voi tallentaa useita profiileita suosikkeihinsa, suosikit voidaan renderöidä käyttämällä map-metodia.

# 10 Yhteenveto

Tässä insinöörityössä kehitettiin Hautauspalvelualan digitaalinen hautapaikan hakutyökalu. Se tehtiin Ondasera Oy:n toimeksiannosta. Insinöörityön tavoitteena oli luoda verkkopohjainen alusta, joka näyttää, miten hitaasti digitalisoituneen alan kehittämisestä voidaan hyötyä.

Ohjelma saavutti kaikki sille asetetut vaatimukset työkalujen toiminnallisuuden ja helppokäyttöisyyden kannalta. Se ylitti myös odotukset joillakin alueilla, muun muassa siinä, miten ohjelman karttafunktiot onnistuttiin saattamaan sellaiseen muotoon, missä kaikkea dataa voidaan säilyttää yhdessä tiedostossa. Onnistumisena voi pitää myös sitä, että palvelussa vältettiin pakollisen käyttäjätilin luominen ja rekisteröityminen. Ohjelmaan onnistuttiin silti kehittämään ominaisuus, jossa käyttäjä voi tallentaa hakutuloksiaan yhteen selkeästi löydettävään sijaintiin.

Erilaiset tuotantoympäristöt tarjoavat erilaisia mahdollisuuksia ja ominaisuuksia vastaavan työkalun kehittämisessä. Valittu lähestymistapa oli tämän insinöörityön tarpeiden kannalta monella tavalla tarkoituksiin sopiva valinta. Työkalujen ilmainen luonne mahdollisti kehittämisen ilman taloudellista painetta ja kiinteitä aikarajoja. Lisäksi JavaScript-ohjelmointikielen suosion vuoksi avustavan kirjallisuuden löytäminen oli helppoa, ja se on monipuolisesti yhteensovitettavaa eri tilanteissa. Ohjelman modulaarinen rakenne antaa sille paljon uudelleenkäyttöarvoa eli sitä voidaan hyödyntää muissakin konteksteissa kuin hautausmaapalvelualla.

Kehitetyn ohjelman tekeminen vaati ohjelmointia ja työkalujen käyttöä edellyttävää teknistä osaamista. Ohjelman kehittämisessä panostettiin käyttöliittymän käytettävyyteen. Tämän hautausmaa-hakuohjelman kehittämisessä onnistuttiin hyvin, koska siinä noudatettiin modernin vastaavanlaisen verkkotyökalun kehityksen periaatteita. Ohjelman modulaarisen suunnittelun avulla voidaan sen toimintoja laajentaa tulevaisuudessa uusilla työkaluilla tai vanhoja kehittämällä.

## Lähteet

Alejandro AO 2023. *Javascript Interactive Map with Leaflet EASY (with Marker Clusters & Popups). YouTube.* <https://www.youtube.com/watch?v=LxenV0YaX8M&list=LL&index=32> (8.4.2025)

Leaflet 2010–2025. *Leaflet Quick Start Guide.* <https://leafletjs.com/examples/quick-start/> (8.4.2025)

Mohit Kumar Toshniwal 2024. *Add Maps to a React.js application using Leaflet.js and OpenStreetMap. YouTube.* <https://www.youtube.com/watch?v=8qEkEkjHls&list=LL&index=57> (8.4.2025)

NPM 2014-2025. *About npm.* <https://www.npmjs.com/about> (8.4.2025)

React 2013–2025. *useEffect.* <https://react.dev/reference/react/useEffect> (8.4.2025)

React 2013–2025. *useRef.* <https://react.dev/reference/react/useRef> (8.4.2025)

React 2013-2025. *useState.* <https://react.dev/reference/react/useState> (8.4.2025)

React Bootstrap 2025. *Grid system.* <https://react-bootstrap.netlify.app/docs/layout/grid> (8.4.2025)

Sirén, Roni 2025. Hautausmaa-alan digitaalinen hakutyökalu, ohjelmakoodi. Tekijän henkilökohtainen tietokanta

## Karttafunktion koodi

```
export default function ExampleMap(props) {
    const navigate = useNavigate();
    const _location = useLocation();
    const [pop, setPop] = useState(false);
    const mapRef = useRef();
    const [block, setBlock] = useState("");
    const [bType, setbType] = useState("");
    const [maintenance, setMaintenance] = useState("");
    const [img, setImg] = useState("");
    const [desc, setDesc] = useState("");
    const [religion, setReligion] = useState("");
    const [price, setPrice] = useState("");
    const [coordsX, setCoordsX] = useState("");
    const [coordsY, setCoordsY] = useState("");
    const [religionValue, setReligionValue] = useState(_loca-
tion.state.Religion);
    const [typesValue, setTypesValue] = useState(_location.state.Buri-
alType);
    const [maintenanceValue, setMaintenanceValue] = useState("");
    const [isChecked, setIsChecked] = useState(false)
    const [resultsAmount, setResultsAmount] = useState(4);
    var layer28

    useEffect(() => {
        mapRef.current = leaflet.map("map").setView([_loca-
tion.state.coordsX, _location.state.coordsY], 17);
        leaflet.tileLayer("https://tile.open-
streetmap.org/{z}/{x}/{y}.png", {
            attribution:
                '&copy; <a href="http://www.openstreetmap.org/copy-
right">OpenStreetMap</a>',
        }).addTo(mapRef.current);
        const customIcon = leaflet.icon({
            iconUrl: "/imgs/mapIcon.png",
        })
        const markers = leaflet.markerClusterGroup({
            iconCreateFunction: function (cluster) {
                return leaflet.divIcon({
                    icon: customIcon,
                    html: '<div class="cluster-div">' + clus-
ter.getChildCount() + '</div>'
                })
            }
        })
        layer28 = leaflet.geoJSON(props.geojson, {
            onEachFeature: function () {
            }
        })
            .addTo(markers);
        layer28.on("click", markerOnClick);
        function markerOnClick(e) {
            var attributes = e.layer.feature.properties;
            var coordinates = e.layer.feature.geometry;
```

```
            setBlock(attributes.block)
            setbType(attributes.bType)
            setMaintenance(attributes.maintenance)
            setCoordsX(coordinates.coordinates[1])
            setCoordsY(coordinates.coordinates[0])
            setPop(true);
            mapRef.current.setView([coordinates.coordinates[1], coor-
dinates.coordinates[0]], 19);
        }
        markers.addTo(mapRef.current)
    }, []);
    function pageOnClick(block, type, maintenance, coords, img, desc,
religion, price) {
        setBlock(block)
        setbType(type)
        setMaintenance(maintenance)
        setImg(img)
        setDesc(desc)
        setReligion(religion)
        setPrice(price)
        setCoordsX(coords[1])
        setCoordsY(coords[0])
        setPop(true);
        mapRef.current.setView([coords[1], coords[0]], 19);
    }
    function loadMore() {
        if (resultsAmount < props.geojson.features.length) {
            setResultsAmount(resultsAmount + 4)
        }
    }
    function loadLess() {
        if (resultsAmount > 4) {
            setResultsAmount(resultsAmount - 4)
        }
    }

    return (
        <>
            <div className={styling.box}>
                <div className={styling.mapPage}>
                    <div className={styling.services_filters}>
                        <div className={styling.background}>
                            <br />
                            <MultipleChoice title="Religion" op-
tions={religions} setInputValue={setReligionValue} value={religion-
Value} />
                            <MultipleChoice title="Burial type" op-
tions={burialTypes} setInputValue={setTypesValue} value={typesValue}
/>
                            <MultipleChoice title="Maintenance" op-
tions={maintenanceTypes} setInputValue={setMaintenanceValue} />
                        </div>
                    </div>
                    <div className={styling.services_results}>
                        <div className={styling.background_results}>
                            <div id="map" ref={mapRef}></div>
                            <Popup trigger={pop}>
                                <div className="btnContainer">
```

```
                                    <button onClick={() => { set-
Pop(false) }} className="backBtn">×</button>
                                </div>
                                <h2>{cemDetails[props.index].city} /
{cemDetails[props.index].cemetery} / block {block}</h2>
                                <div className={styling.box}>
                                    <br />
                                    <Row>
                                        <Col xs={9}>
                                            <img src={img} class-
Name={styling.img}></img> <br /><br />
                                            <br />
                                            <h2>block {block}</h2>
                                            <br />
                                            <p>{desc}</p>
                                            <br />
                                            <FormPrefilled form-
Name={"Contact details"} json={cemDetails} index={props.index} /><br
/>
                                            <br />
                                        </Col>
                                        <Col>
                                            <p>Burial type:
{bType}</p>
                                            <p>Grave type: {bType}</p>
                                            <p>Available religions:
{religion}</p>
                                            <p>Maintenance: {mainte-
nance}</p>
                                            <br />
                                            <hr />
                                            <br />
                                            <h3>Price:</h3>
                                            <h3>{price}</h3>
                                            <br />
                                        </Col>
                                    </Row>
                                    <form>
                                        <GraveEnquiryForm />
                                        <button className="btnPage"
onClick={() => { navigate(cem, { replace: true, state: { coordsX:
```

```
coordsX, coordsY: coordsY } }) }}>Send an enquiery for a grave in
block {block}</button>
                                    </form>
                                </div>
                            </Popup>
                            <br />
                            <div className={styling.block}>
                                <span className="subHeadB">Explore
cemetery blocks in {cemDetails[props.index].cemetery}</span>
                                <div className="btnContainer">
                                    <div className={styling.forceL-
ine}>
                                        <div className="btnContain-
erR"><button onClick={() => { loadLess() }} className="back-
Btn">←</button>  </div>
                                        <div className="btnContain-
erR"><button onClick={() => { loadMore() }} className="back-
Btn">→</button>  </div>
                                    </div>
                                </div> <br />
                                <Row xs={4} sm={4} md={4} lg={4}
xl={4} xxl={4} className="g-2">
                                    {props.geojson.features
                                        .filter((data) => {
                                            const titleType =
data.properties.bType.toLowerCase();
                                            const inputType =
typesValue.toLowerCase();
                                            const titleReligion =
data.properties.religion.toLowerCase();
                                            const inputReligion =
religionValue.toLowerCase();
                                            const titleMaintenance =
data.properties.maintenance.toLowerCase();
                                            const inputMaintenance =
maintenanceValue.toLowerCase();

                                            return titleMainte-
nance.includes(inputMaintenance) && titleType.includes(inputType) &&
titleReligion.includes(inputReligion);
                                        })
                                        .slice(resultsAmount - 4, re-
sultsAmount)
                                        .map(item => (
                                            <Col>
                                                <div className="coun-
tryHolder">
                                                    <button class-
Name="cardBtn" onClick={() => {
                                                        pageOn-
Click(item.properties.block, item.properties.bType, item.proper-
ties.maintenance, item.geometry.coordinates, item.properties.imgUrl,
```

```
item.properties.desc, item.properties.religion, item.proper-
ties.price);
                                                        }} variant="white"
>
                                                            <Card class-
Name="custom-class-service">
                                                                <Card.Img
                                                                    vari-
ant="top"
                                                                    src={i
tem.properties.imgUrl}
                                                                    class-
Name="img"
                                                                />
                                                                <Card.Body
className="d-flex flex-column">
                                                                    <span
className="desc">Block {item.properties.block}</span>
                                                                </Card.Bod
y>
                                                            </Card>
                                                        </button>
                                                    </div>
                                                </Col>
                                            ))}
                                        </Row>
                                    </div>
                                </div>
                                <br /><br />
                            </div>
                        </div>
                    </div>
                </>
            )
}
```

## GeoJSON-tiedosto

{"type":"FeatureCollection","features":[{"type":"Feature","properties":{"cemetery":"Malmin hautausmaa","city":"Helsinki","country":"Finland", "hours":"7-22","desc":"lorum ipsum","types":"lawn burial, green burial, columbarium","religion":"Christianity, Evangelical lutheran, Catholic, Non-religious","imgUrl":"/imgs/malmi.PNG"},"geometry":{"coordinates":[25.027109310124132,60.23675797380116],"type":"Point"},"id":0},

{"type":"Feature","properties":{"cemetery":"Hietaniemen hautausmaa","city":"Helsinki","country":"Finland", "hours":"9-16","types":"lawn burial","religion":"Christianity, Evangelical lutheran, Catholic, Non-religious","imgUrl":"/imgs/hietaniemi.jpg"},"geometry":{"coordinates":[24.91445422110948,60.170814508083],"type":"Point"},"id":1},

{"type":"Feature","properties":{"cemetery":"Helsingin juutalainen hautausmaa","city":"Helsinki","country":"Finland", "hours":"","desc":"lorum ipsum","types":"lawn burial","religion":"Christianity, Evangelical lutheran, Catholic, Non-religious","imgUrl":"/imgs/helsinki.jpg"},"geometry":{"coordinates":[24.911687824679575,60.174126424910696],"type":"Point"},"id":2,

{"type":"Feature","properties":{"cemetery":"Helsingin ortodoksinen hautausmaa","city":"Helsinki","country":"Finland",
"hours":"","desc":"lorum ipsum","types":"lawn burial","religion":"Christianity, Evangelical lutheran, Catholic, Non-religious","imgUrl":"/imgs/helsinki.jpg"},"geometry":{"coordinates":[24.918824888969112,60.16574044594881],"type":"Point"},"id":3},

{"type":"Feature","properties":{"cemetery":"Maunulan hautausmaa","city":"Helsinki","country":"Finland", "hours":"","desc":"lorum ipsum","types":"lawn burial","religion":"Christianity, Evangelical lutheran, Catholic, Non-religious","imgUrl":"/imgs/helsinki.jpg"},"geometry":{"coordinates":[24.917323735025434,60.227970714006375],"type":"Point"},"id":4

## Suosikkien selaamisen funktion koodi

```
export default function FavouritesViewer() {
    const [pop, setPop] = useState(false);
    const [favs, setFavs] = useState(() => {
        const localValue = localStorage.getItem("FAVOURITES_BLOCK")
        if (localValue == null) return []
        return JSON.parse(localValue)
    })

    const [block, setBlock] = useState("");
    const [desc, setDesc] = useState("");
    const [img, setImg] = useState("");
    const [bType, setBtype] = useState("");
    const [religion, setReligion] = useState("");
    const [maintenance, setMaintenance] = useState("");
    const [price, setPrice] = useState("");
    const [index, setIndex] = useState(0);

    //store favourites in local storage
    useEffect(() => {
        localStorage.setItem("FAVOURITES_BLOCK", JSON.stringify(favs))
    }, [favs])

    function deleteFavourite(id) {
        setFavs(currentFavs => {
           return currentFavs.filter(fav => fav.id !== id) // poistetaan
ei haluttu id
        })
    }

    function pageOnClick(block, type, maintenance, img, desc, religion,
price, index) {
        setBlock(block)
        setBtype(type)
        setMaintenance(maintenance)
        setImg(img)
        setDesc(desc)
```

```
        setReligion(religion)
        setPrice(price)
        setIndex(parseInt(index))

        setPop(true);
    }

    return (
        <>
            {favs.length == 0 && "no added favourites."}
           <Row xs={1} sm={1} md={2} lg={3} xl={3} xxl={3} className="g-
2">
                  {favs
                    .map(item => (
                      <li key={item.id}>
                      <Col>
                        <div className="countryHolder">
                          <button className="cardBtn" onClick={() => {
                                   pageOnClick(item.block, item.bType,
item.maintenance, item.imgUrl, item.desc, item.religion, item.price,
item.index);
                          }} variant="white" >
                            <Card className="custom-class-service">

                              <Card.Img
                                variant="top"
                                src={item.imgUrl}
                                className="img"
                              />

                             <Card.Body className="d-flex flex-column">
                               <span className="desc"> {cemDetails[par-
seInt(item.index)].city} / {cemDetails[parseInt(item.index)].cemetery}
/ Block {item.block}</span>
                              </Card.Body>
                            </Card>
                          </button>
                        </div>
                      </Col>
```

```
                    </li>
                ))}
            </Row>
        <Popup trigger={pop}>
            <div className="btnContainer">
                <button onClick={() => { setPop(false) }} class-
Name="backBtn">×</button>
            </div>
            <h2>{cemDetails[index].city} / {cemDetails[index].ceme-
tery} / block {block}</h2>
            <div className={styling.box}>
                <br />
                <Row>
                    <Col xs={9}>
                      <img src={img} className={styling.img}></img>
<br /><br />
                        <br />
                        <h2>block {block}</h2>
                        <br />
                        <p>{desc}</p>
                        <br />
                        <FormPrefilled formName={"Contact details"}
json={cemDetails} index={index} /><br />
                        <br />
                    </Col>
                    <Col>
                        <p>Burial type: {bType}</p>
                        <p>Grave type: {bType}</p>
                        <p>Available religions: {religion}</p>
                        <p>Maintenance: {maintenance}</p>
                        <br />
                        <hr />
                        <br />
                        <h3>Price:</h3>
                        <h3>{price}</h3>
                        <br />
                            <button onClick={() => { deleteFavour-
ite(item.id) }} variant="danger">remove from favourites</button>
                    </Col>
                </Row>
```

```
                    <form>
                        <GraveEnquiryForm />
                        <button className="btnPage" onClick={() => { ""
}}>Send an enquiery for a grave in block {block}</button>
                    </form>
                </div>
            </Popup>
        </>
    )
}
```