Tanssitapahtumien hakupalvelimen rakentaminen Next.JS-teknologialla

Jere Manni

Haaga-Helia ammattikorkeakoulu
Tradenomi (Tietojenkäsittelyn koulutusohjelma)
Opinnäytetyö
2023

Tekijä(t)
Manni Jere

Tutkinto
Tradenomi

Opinnäytetyön nimi
Tanssitapahtumien hakupalvelimen rakentaminen Next.JS-teknologialla

Sivu- ja liitesivumäärä
57

Suomessa seuratanssitapahtumien julkaiseminen ja markkinointi tapahtuu pääsääntöisesti sosiaalisessa mediassa, eikä kunnollista keskitettyä hakupalvelua ole tarjolla. Lisäksi eri yhteisöillä on omia ryhmiä sekä kanavia, minkä takia tanssijoiden on oltava aktiivisia sosiaalisessa mediassa, jos haluaa harrastaa aktiivisesti tanssia.

Tässä työssä tehtiin selainpohjainen web-sovellus tanssitapahtumien julkaisemiseen ja niiden hakemiseen. Pyrkimyksenä oli tuottaa helppokäyttöinen palvelu niin tapahtumien järjestäjille sekä tapahtumia etsiville käyttäjille. Työssä onnistuttiin tekemään tietojenhallintasivu, josta tapahtumanjärjestäjä voi lisätä, muokata sekä poistaa tietoja. Sen lisäksi tehtiin loppukäyttäjälle hakusivu, josta tapahtumia voi etsiä eri parametreilla ja hakutulokset näytetään listattuna sekä kartalla.

Työn aikana opittiin käyttämään useita moderneja ohjelmointiteknologioita, kirjastoja sekä palveluita. Ohjelmisto rakennettiin Next.JS-viitekehyksen ympärille, josta käytettiin uusimpia päivityksiä (versio 13.4+) ja niiden mukana tullutta App-router/directory-mallia. Ohjelmointikielenä ja pääkirjastoina käytettiin Reactia, Typescriptiä sekä Tailwindia. Sovellusta ylläpidettiin Vercelin alustalla, jonka palveluiden kautta voitiin pyörittää PostgreSQL tietokantaa sekä applikaation testiserveriä. Kirjautuminen ja henkilötietojen ylläpito hoidettiin käyttäen kolmannen osapuolen (Clerk) palveluita.

Työ tehtiin yhteistyössä Tanssikoulu Antti Törmäsen kanssa, jonka asiantuntemus auttoi ohjaamaan projektin suunnittelua ja toteutusta.

Asiasanat
Tanssi, hakupalvelin, Next.js, SQL, React, Vercel

Sisällys

1 Johdanto

Tanssi on yksi maailman suosituimpia liikunnan muotoja, mutta usein se liitetään vain ryhmäliikuntatunteihin, kilpatanssiin tai ihan vain yöelämän tanssilattioihin. Näiden lisäksi on olemassa tanssin ja sosiaaliseen ympäristöön perustuva seuratanssipiiri, mikä kattaa laajan joukon erilaisia paritanssilajeja, joidenka parissa järjestetään tanssitapahtumia kuten lavatansseja, tanssibileitä sekä tanssikursseja. Seuratanssipiireissä tanssijat ovat valmiita matkustamaan pitkiä matkoja nauttiakseen intohimostaan, ja erilaiset tapahtumat sekä leirit vetävät puoleensa kävijöitä ulkomaita myöden.

Tanssiyhteisöt ovat kuitenkin hajautuneet, ja tapahtumien tiedonetsintä voi olla hankalaa. Facebookin käyttö tanssitapahtumien julkaisemiseen ja mainostamiseen on järjestäjille paras vaihtoehto, ja sillä tavoittaa suurimman osan tanssiharrastajista. Se myös tarkoittaa sitä, että tanssiharrastajien on lähes pakko käyttää Facebookia tietääkseen missä ja mitä tapahtuu. Nuoremman sukupolven vähentynyt käyttö Facebookissa rajoittaa tiettyjen tapahtumien näkyvyyttä ja ylipäänsä nostaa nuorten kynnystä päästä lajin pariin. Instagram sekä TikTok ovat toisaalta suosittuja nuorten keskuudessa ja tanssi onkin yksi suosituimpia sisällönaiheita, mutta nämä palvelut eivät tarjoa tehokasta keinoa tapahtumien etsimiseen, vaikka tapahtumien mainostaminen on siirtynyt näihin palvelualustoihin.

Lavatansseille on olemassa hakusivusto nimeltä tanssi.net (tanssi.net, 2023), mutta sen teknologia, ulkonäkö ja käytettävyys ovat vanhanaikaista. Sivustolla ei ole mobiilitukea, eikä tietoja pääse syöttämisen jälkeen muokkaamaan. Lisäksi se on täynnä mainoksia, mikä on ärsyttävää käyttäjien mielestä. Se on silti käytössä, sillä parempaa vaihtoehtoa ei ole.

Tämän työn tavoitteena on luoda helppokäyttöinen ja keskitetty alusta tanssitapahtumien sekä tanssikurssien julkaisemiseen ja etsimiseen. Pyrkimyksenä on yhdistää tanssijat ja tapahtumajärjestäjät nykyaikaisen teknologian avulla.

Työn tekijällä on yli 8 vuoden kokemus seuratanssimaailmasta tanssin harrastajana sekä osittain myös tanssin opettajana. Aihe on erityisen merkityksellinen, sillä tanssista on tullut erittäin tärkeä osa kirjoittajan elämää. Tästä syystä halu kehittää tanssimaailmaa oman osaamisen kautta on tekijälle erittäin motivoivaa. Hakupalvelun kautta pyritään tuomaan tanssia lähemmäs niitä, jotka eivät tanssia ole vielä löytäneet, sekä helpottamaan nykyisiä tanssiharrastajia löytämään itselleen sopivia tapahtumia.

Työ tehtiin yhteistyössä Tanssikoulu Antti Törmäsen kanssa (Törmänen, 2023). Antti on kokenut seuratanssiopettaja ja tietää hyvin, miten ala toimii. Kuten moni muukin opettaja, niin myös Antti

opettaa omassa tanssikoulussaan sekä toimii vierailevana opettajana muissa tanssikoulussa eri kaupungeissa ja sen lisäksi lukuisissa tapahtumissa koko Suomen alueella ympäri vuoden.

Opinnäytetyön tuloksena tehty selainpohjainen sovellus ja kaikki siihen liittyvät immateriaalioikeudet jäävät työn tekijälle eli Jere Mannille. Yhteistyö Tanssikoulu Antti Törmäsen kanssa on tarkoitus auttaa rakentamaan hyvä palvelualusta tanssiyhteisöille sekä tanssikoulun tarpeisiin, eikä suoranaisesti sovellusta heidän omakseen. Vastaavanlainen sovellus voisi helposti maksaa konsulttityönä tehtynä useita kymmeniä tuhansia euroja. Tämä työ palvelee kaikkia osapuolia ja rahalliset riskit jäävät hyvin pieniksi.

1.1 Tanssista

Tanssimaailmassa eri tanssilajit ja -tyylit kertovat monimuotoisuudesta ja tanssin rikkaudesta. Improvisoitu tanssi, kuten spontaani liikeilmaisu, on vastakohta koreografiselle tanssille, jossa jokainen askel ja liike on ennalta suunniteltu ja harjoiteltu. Koreografi luo tällaisen tanssin, jossa tanssijat seuraavat tarkkaa suunnitelmaa esityksissään.

Kilpatanssissa parit kilpailevat teknisessä osaamisessa, musikaalisuudessa ja esitystaidossa. Tämä voi olla sekä improvisoitua että koreografista. Ryhmäliikunta ja ryhmätanssi ovat myös tärkeitä osia tanssin kentässä, joissa suuremmat ryhmät tanssivat yhdessä ohjaajan johdolla tai suorittavat koreografiaa, kuten rivitanssissa tai kansantanssissa. Yksilötanssi korostaa henkilökohtaista ilmaisua ja tanssitekniikkaa, antaen tanssijalle tilaa esittää omia tanssikuvioitaan.

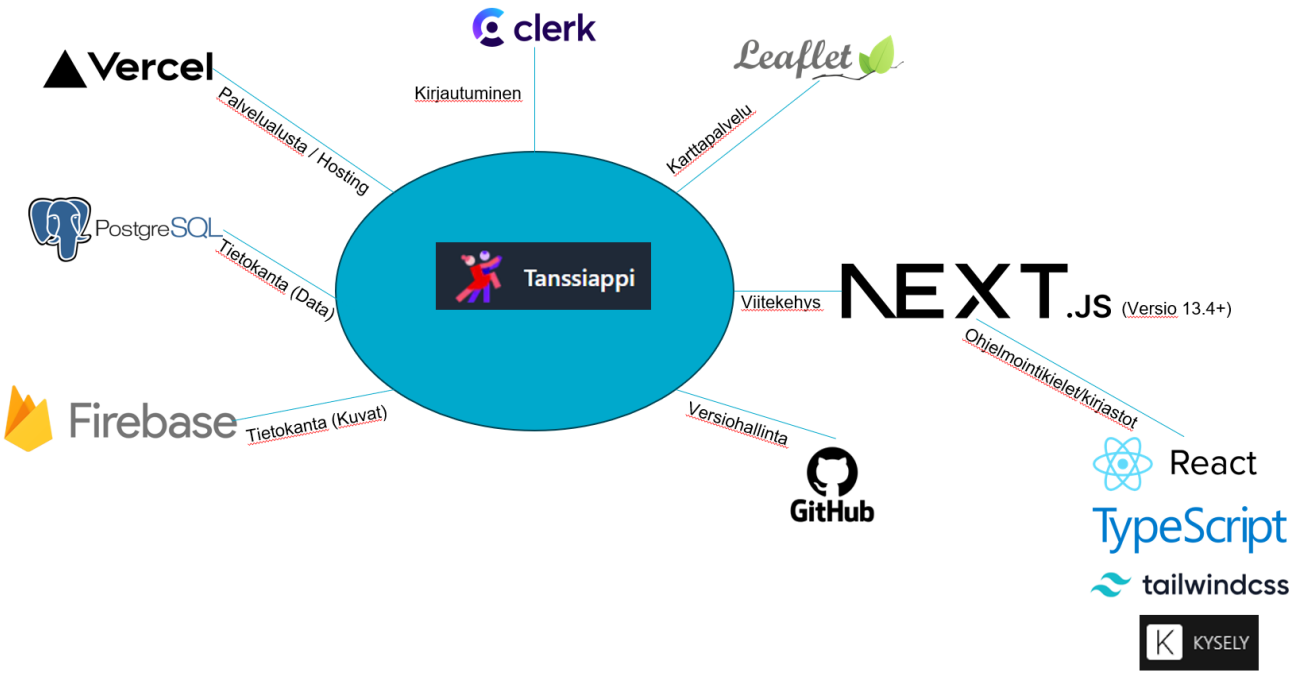
Lavatanssit ja seuratanssibileet taas on improvisoitua paritanssia, jossa kaksi tanssijaa liikkuu synkronoidusti yhdessä musiikin tahtiin eri lajeja, kuten esimerkiksi perinteisiä valssia tai tangoa, vauhdikkaampia swinglajeja kuten fuskua ja buggia, latinorytmisiä lajeja kuten salsaa tai bachataa, tai jopa kansantansseja kuten jenkkaa ja polkkaa. Seuratanssi ja tanssibileet heijastavat tanssin sosiaalista puolta, missä ihmiset kokoontuvat nauttimaan musiikista ja tanssista. Tapahtumat voivat vaihdella paritanssikursseista ja tanssibileistä tanssileireihin, jossa tanssi on sekä opetuksen että hauskanpidon keskiössä. Tanssikursseilla, osallistujat oppivat uusia tanssiliikkeitä ja -koreografioita ammattitaitoisen ohjaajan johdolla.

Näiden eri tanssityylien ja -muotojen ymmärtäminen on olennaista tanssin maailmassa, missä jokaisella lajilla on oma roolinsa ja merkityksensä. Ehkä kaikkein tärkein aspekti tanssissa on se, että se yhdistää ihmisiä iästä, sukupuolesta, rodusta, ulkonäöstä ja kielestä riippumatta. Lavatansseja järjestetään Suomessa päivittäin useita ja viikonloppuisin jopa yli 30 tapahtumaa päivässä

(tanssi.net, 2023). Helsingissä järjestetään useita tanssibileitä joka viikko ja erityisesti latinotanssin harrastajat ovat erittäin aktiivisia.

2 Käytetyt alustat ja teknologiat

Tässä opinnäytetyössä on hyödynnetty useita nykyaikaisia teknologioita ja alustoja, joiden yhdistelmä on mahdollistanut modernin web-sovelluksen kehittämisen. Projektin kehittämisen ytimessä on ollut Next.JS, joka on Reactiin perustuva viitekehys, tarjoten dynaamisen ja tehokkaan tavan rakentaa sovellusta. Lisää Next.JSstä kappaleessa 3. Käyttöliittymän luomiseen on käytetty Reactia, mikä on suosittu JavaScript-kirjasto dynaamisten ja interaktiivisten käyttöliittymien kehittämiseen. Tietojen tallentamiseen ja käsittelyyn valittiin PostgreSQL, joka on monipuolinen relaatiotietokannan hallintajärjestelmä. Kuvien tallentamiseen käytettiin Googlen Firebase-alusta tietokantapalvelua. Ulkoasun suunnittelussa hyödynnettiin Tailwind CSS:ää, joka on käytännöllinen työkalu visuaalisesti miellyttävien verkkosivujen ja sovellusten rakentamiseen. Karttapalveluna käytettiin Leaflet-kirjastoa. Kirjautumiseen ja henkilötietojen tallentamiseen käytettiin kolmannen osapuolen sovellusintegraatiota (Clerk). Sovelluksen kehitys, esikatselu ja julkaisu tapahtuivat Vercelin pilvipohjaisessa ympäristössä, joka tarjoaa tehokkaan integraation Reactin ja Next.JS:n kanssa. Lisäksi projektin versionhallinta ja yhteistyö toteutettiin GitHubin avulla, mikä toimi keskeisenä koodin säilytyspaikkana ja yhteistyöalustana. Muina apuvälineinä toimi mm. Visual Studio Code, joka on ilmainen ohjelmointiympäristö sekä ChatGPT yleiseen avustukseen.



Kuva 1. Ohjelmiston kehityksessä käytetyt teknologiat sekä niiden käyttötarkoitukset

2.1 Vercel

Vercel (Vercel, 2023) on moderni pilvipohjainen alusta, joka on suunniteltu verkkosovellusten kehittämisen, esikatselun ja julkaisun nopeuttamiseen ja yksinkertaistamiseen. Erityisesti yhteensopiva Reactin ja Next.js:n kaltaisten teknologioiden kanssa, Vercel on suosittu kehittäjien keskuudessa monipuolisuutensa ja helppokäyttöisyytensä ansiosta.

Tässä opinnäytetyössä Vercel palveli web-sovelluksen kehitysympäristönä, tarjoten kattavia ominaisuuksia. Vercelin integrointi suosittuihin versionhallintajärjestelmiin, kuten GitHubiin, mahdollistaa automatisoidun rakennus- ja julkaisuprosessin. Jokainen koodipäivitys voi käynnistää automaattisen sovelluksen rakennuksen ja julkaisun Vercelissä, tehden jatkuvasta integraatiosta ja toimituksesta sujuvaa. Vercel tarjoaa myös reaaliaikaista palautetta julkaisuprosessin aikana, auttaen kehittäjiä tunnistamaan ja korjaamaan mahdolliset ongelmat nopeasti. Tämä sisältää esimerkiksi ohjelmointivirheiden tunnistamisen ja suorituskykyyn liittyvät vihjeet.

Turvallisuuden osalta Vercel tarjoaa automaattisen SSL-varmenteen kaikille projekteille, varmistaen, että isännöidyt sivustot ja sovellukset ovat suojattuja HTTPS:n kautta. Lisäksi Vercel tukee serverittömiä funktioita, jotka mahdollistavat dynaamisten ja skaalautuvien sovellusten luomisen ilman perinteisen palvelininfrastruktuurin ylläpitoa.

Ympäristömuuttujien tuki mahdollistaa eri konfiguraatioiden helpon mukauttamisen kehitys-, testaus- ja tuotantoympäristöissä. Vercelin infrastruktuuri on suunniteltu automaattisesti skaalautuvaksi, mikä tarkoittaa, että sovelluksen liikenteen kasvaessa resurssit mukautuvat tarpeen mukaan.

Lisäksi Vercel käyttää optimoituja suorituskykypalvelimia ja CDN-verkkoa (Content Delivery Network), varmistaakseen, että sovelluksen sisältö toimitetaan käyttäjille mahdollisimman nopeasti.

Vercel Postgres on serveritön SQL-tietokantapalvelu, on integroitu alustaan ja on ihanteellinen työkalu esimerkiksi asiakasprofiilien käsittelyyn ja käyttäjien tuottaman sisällön tallentamiseen. Tämä palvelu on tarjolla beta-versiona ja kattaa sekä ilmaisen että maksullisen version tarpeet.

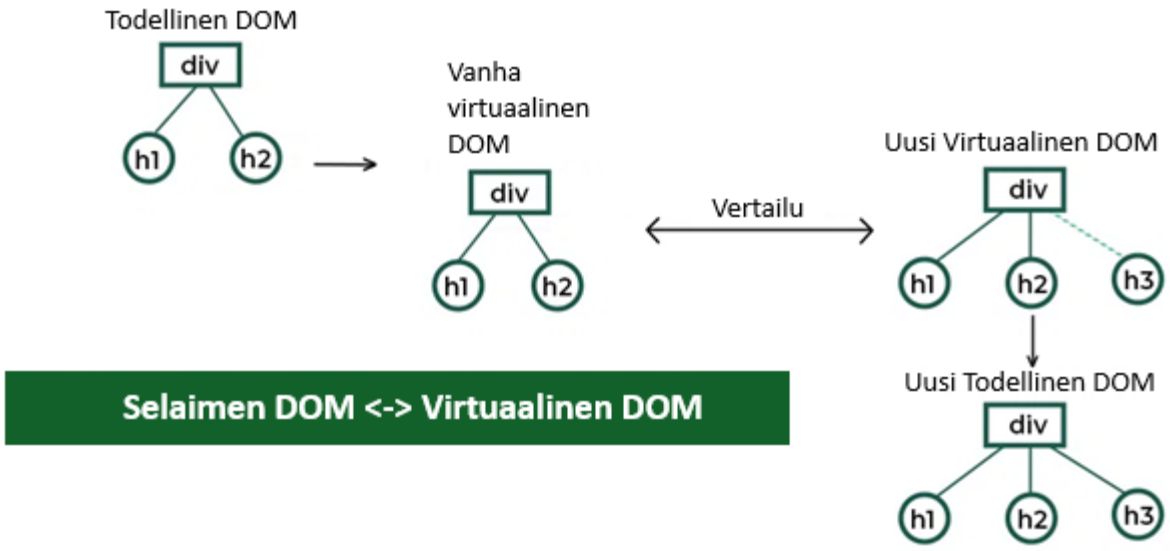
Vercelin kolme erilaista tilityyppiä – Hobby, Pro ja Enterprise – tarjoavat joustavuutta projektille, riippuen sen koosta ja vaatimuksista. Hobbypaketti on ilmainen ja sopii pienemmille projekteille, sisältäen kaikki perusominaisuudet web-sovelluksen luomiseen ja ylläpitämiseen.

2.2 React

React on komponenttipohjainen JavaScript-kirjasto, jota käytetään modulaaristen käyttöliittymien luomiseen ja kannustaa luomaan uudelleenkäytettäviä UI-komponentteja, jotka käyttävät dynaamista dataa ja voivat reagoida datan muutokseen komponenttitasolla. Reactin on kehittänyt Facebook, ja se geegsforgeegs.org-sivuston (geeksforgeeks, 2023a) mukaan suosituin front-end-kirjasto web-sovellusten kehittämisessä. Sen suosio perustuu sen tehokkuuteen, uudelleenkäytettävyyteen ja laajaan ekosysteemiin. React on erityisen suosittu sovellusten rakentamisessa, joissa käyttöliittymä vaatii dynaamisuutta ja jatkuvia päivityksiä.

React käyttää niin sanottua virtuaalista DOM:ia (Mozilla, 2023b), eli Document Object Model ohjelmointirajapintaa, optimoimaan käyttöliittymän päivitykset ja vertailee sitä nykyiseen DOM:iin. Sen avulla React voi tehokkaasti päivittää vain ne osat käyttöliittymästä, jotka ovat muuttuneet, sen sijaan että päivittäisi koko DOM-puun (Kuva 2). Se kuvaa verkkosivun sisällön, rakenteen ja tyylit hierarkkisena puurakenteena. Jokainen elementti, kuten otsikot, linkit ja tekstit, on esitetty solmuina tässä puussa.

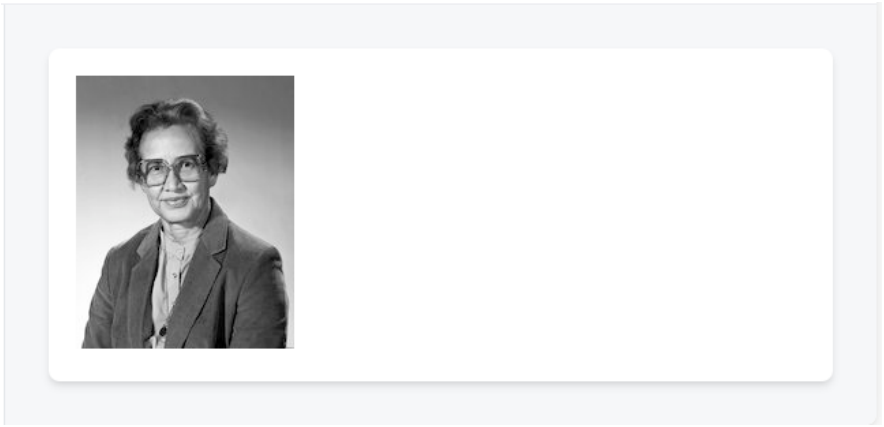
React-sovellus koostuu erillisistä komponenteista, jotka ovat itsenäisiä ja uudelleenkäytettäviä yksiköitä. Komponentit mahdollistavat sovelluksen jakamisen pienempiin osiin, mikä helpottaa kehitystä ja ylläpitoa. React noudattaa yksisuuntaista tiedon virtausta, mikä tarkoittaa, että data liikkuu aina vanhemmalta komponentilta lapsikomponenteille. Tämä tekee sovelluksen tilanhallinnasta ennakoitavaa ja helpommin hallittavaa.



Kuva 2. Selaimen ja vanhan virtuaalisen DOM:in vertailu uuteen virtuaaliseen DOM:iin (mukaillen geegsforgeeks.org, 2023)

Komponentit ovat Reactin ydin ja niiden avulla kehittäjä voi rakentaa pienistä ja yksinkertaisista paloista suuremman kokonaisuuden. Komponentit ovat siis ohjelmiston rakennuspalasia, joita voidaan uusiokäyttää useammassa paikassa. Hahmotetaan käyttöliittymän rakennetta tekemällä yksinkertainen komponentti, jonka ainoa elementti on kuva.

```
// Example of simple Profile component

export default function Profile() {
    return (
      <img
        src="https://i.imgur.com/MK3eW3Am.jpg"
        alt="Katherine Johnson"
      />
    )
  }
```



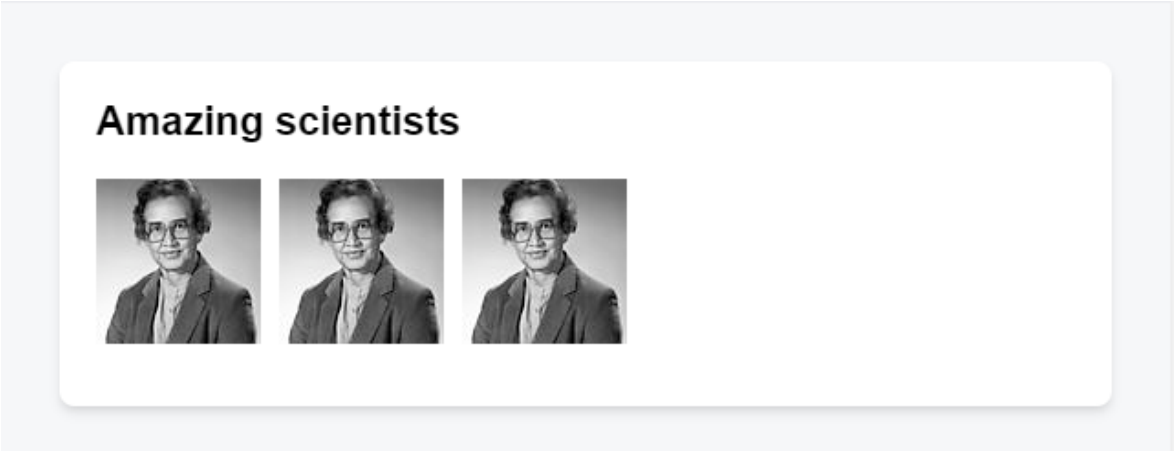
Kuva 3. Esimerkki profiilikomponentista html-sivustossa (React, Learn: Your first component, 2023)

Hahmotellaan vielä, miten samaista Profile-komponenttia voidaan käyttää monesti yhden html-elementin sisällä.

```
// Usage of Profile component
export default function Gallery() {
    return (
        <section>
            <h1>Amazing scientists</h1>
            <Profile />
            <Profile />
            <Profile />
        </section>
    );
}
```

Kun tämä React-koodi päätyy selaimelle luettavaksi, niin käytännössä selain avaa jokaisen Profile-komponentin omaksi html-koodiksi.

```
// What the browser sees

<section>
  <h1>Amazing scientists</h1>
  <img src="https://i.imgur.com/MK3eW3As.jpg" alt="Katherine Johnson" />
  <img src="https://i.imgur.com/MK3eW3As.jpg" alt="Katherine Johnson" />
  <img src="https://i.imgur.com/MK3eW3As.jpg" alt="Katherine Johnson" />
</section>
```



Kuva 4. HTML-sivu, jossa on otsikko sekä 3 erillistä Profile-komponenttia (React, 2023)

Reactin sisäänrakennetut koukut (Hooks) mahdollistavat tilan hallinnan ja sivuvaikutusten käytön funktionaalisissa komponenteissa ilman, että tarvitsee kirjoittaa luokkakomponentteja. Ennen koukkujen tuloa funktionaaliset komponentit olivat pääasiassa tilattomia ja puhtaita, mutta koukkujen avulla niihin voitiin lisätä monimutkaisempaa logiikkaa ja tilanhallintaa.

Yleisimmin käytettyjä koukkuja ovat *useState* tilan hallintaan, *useEffect* sivuvaikutusten, kuten datan hakemisen tai DOM-manipulaatioiden toteuttamiseen sekä useContext kontekstien käyttöön. Reactin yhteisö on myös kehittänyt monia mukautettuja koukkuja erilaisiin käyttötarkoituksiin, ja kehittäjät voivat myös luoda omia mukautettuja koukkujaan tarpeidensa mukaan.

Koukuilla on tavoitteena tehdä React-sovellusten kirjoittaminen selkeämmäksi, helpommin ymmärrettäväksi ja vähemmän virheherkäksi. Ne mahdollistavat logiikan jakamisen komponenttien välillä ilman, että tarvitsee muuttaa komponenttirakennetta tai luottaa monimutkaisempiin kuvioihin, kuten korkeamman järjestyksen komponentteihin.

useState on yksi Reactin peruskoukuista, joka mahdollistaa tilan käytön funktionaalisissa komponenteissa. Kun *useState*-koukkua kutsutaan, se palauttaa parin: nykyisen tilan arvon ja funktion, jolla tilaa voidaan päivittää. Tämän rakenteen avulla komponentti voi reagoida dynaamisesti käyttäjän toimintoihin tai muihin muutoksiin sovelluksessa. Esimerkiksi, jos haluat luoda laskurin, voit käyttää *useState*-koukkua seuraavasti:

```
// example of simple state

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

Tässä count on tilamuuttuja, joka alustetaan arvolla 0, ja setCount on funktio, jota voidaan käyttää päivittämään countin arvoa. Joka kerta kun setCountia kutsutaan, komponentti suorittaa uudelleenrenderöinnin päivitetyllä tilalla. *useState* tekee tilan hallinnasta Reactissa yksinkertaista ja intuitiivista, ja se on keskeinen osa monien React-sovellusten toiminnallisuutta.

useContext on toinen Reactin tarjoama keskeinen koukku, joka mahdollistaa kontekstin käytön funktionaalisissa komponenteissa. Konteksti on mekanismi Reactissa, joka mahdollistaa tietojen siirtämisen komponenttien puuhierarkiassa ilman, että tarvitsee välittää rekvisiittoja (*props*) manuaalisesti jokaisen tasolle.

Kontekstin yleinen käyttötapaus on esimerkiksi teeman asetusten, käyttäjätietojen tai muiden globaalien tietojen jakaminen monille komponenteille sovelluksessa. Se tarjoaa erityisen hyödyllisen ratkaisun, kun tietyt arvot tai funktiot on oltava saatavilla monissa hierarkian osissa.

Kun käytetään *useContext*-koukkua, ensimmäinen askel on luoda konteksti *createContext*-funktiolla, jonka jälkeen voit käyttää Context.Provider -komponenttia arvon antamiseen alikomponenteille ja lopulta YourComponent sekä kaikki sen alikomponentit voivat nyt käyttää *useContext*-hooksia kontekstin arvon hankkimiseen:

```
// example of simple context
const ThemeContext = React.createContext('light');

// Wrapping a component with a context
<ThemeContext.Provider value="dark">
    <YourComponent />
</ThemeContext.Provider>

// Using the context in YourComponent
const theme = useContext(ThemeContext);  // arvo on "dark"
```

useEffect on koukku, joka mahdollistaa sivuvaikutusten, kuten datan hakemisen, tilauksien manuaalisen muutoksen tai manuaalisen DOM-muokkauksen, suorittamisen funktionaalisissa komponenteissa. Se ottaa vastaan kaksi parametria: ’setup’ ja ’dependecies’, joista ensimmäinen on pakollinen. Setup-parametri suorittaan efektin logiikan, eli se on käytännössä jokin funktio, mikä tekee jotain. Setup-funktiosi voi myös valinnaisesti palauttaa siivousfunktion.

Kun komponenttisi lisätään DOM:iin, React suorittaa setup-funktiosi. Jokaisen uudelleenrenderöinnin jälkeen, kun riippuvuudet ovat muuttuneet, React suorittaa ensin siivousfunktion (jos olet antanut sen) vanhoilla arvoilla ja sitten setup-funktiosi uusilla arvoilla. Kun komponenttisi poistetaan DOM:ista, React suorittaa siivousfunktion.

Riippuvuuksien (*dependencies*) lista niistä parametreista, joiden muutoksen seurauksena setup-funktio ajetaan. React vertaa jokaista riippuvuutta sen edelliseen arvoon käyttäen Object.is-vertailua. Jos tämä argumentti jätetään pois, setup-funktio suoritetaan jokaisen komponentin uudelleenrenderöinnin jälkeen. Jos argumentiksi annetaan tyhjä lista, niin setup-funktio ajetaan vain yhdesti, kun komponentti liitetään DOM:iin (ensimmäinen renderöinti).

Käytettäessä *useEffect*-koukkua voit määrittää funktion, joka suoritetaan tietyissä tilanteissa:

```
// Example of simple useEffect (runs on every render)

useEffect(() => {
    // setup-function
    console.log('Component rendered or added to DOM.');

    // cleanup-function
    return () => {
        console.log('Component removed from DOM.');
    };
}); // no dependecies given


// Example of conditional useEffect (runs when dependencies change)

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

useEffect(() => {
    document.title = `Clicks: ${count}`;
}, [count]);  // [count] = dependecies
```

```
// Example of one-time useEffect (runs only on the first render)

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

useEffect(() => {
    document.title = ` Clicks: ${count}`;
}, []);  // [] = empty dependecies list
```

2.3 Typescript

TypeScript on avoimen lähdekoodin ohjelmointikieli, joka laajentaa JavaScriptiä tyypityksellä. Se tarjoaa ohjelmoijille mahdollisuuden määrittää tietotyyppejä muuttujille, funktioille ja olioille. TypeScriptin tavoitteena on tehdä JavaScriptistä vakaampi, helpommin ylläpidettävä ja virheistä vähemmän altis ohjelmointikieli. TypeScriptin tärkein piirre on sen kyky määrittää ja tarkistaa tietotyyppejä koodissa ennen suorituksen aikana. Tämä auttaa löytämään ja välttämään monia yleisiä ohjelmointivirheitä jo kehitysvaiheessa. (TypeScript, 2023)

TypeScript noudattaa ECMAScript-standardia (Wikipedia, 2023a), mikä tarkoittaa, että se on yhteensopiva monien JavaScript-ympäristöjen ja -kirjastojen kanssa. TypeScriptiä voi käyttää yhdessä monien muiden ohjelmistokehysten ja -kirjastojen kanssa, kuten React, Angular ja Node.js.

TypeScript on suosittu valinta etenkin suurissa ja monimutkaisissa projekteissa, joissa virheiden välttäminen ja koodin ylläpidettävyys ovat tärkeitä. Se tarjoaa monia etuja, mutta vaatii myös hieman opettelua verrattuna pelkkään JavaScriptiin.

Näytetään esimerkki TypeScriptin perusideasta. Ensimmäiseksi muodostetaan vakiomuuttuja “user”, joka saa kaksi ominaisuutta ”name” ja ”id”. Sen jälkeen muodostetaan sama vakiomuuttuja käyttäen TypeScript standardia. Lopuksi vielä väärin määritetty vakiomuuttuja, mikä aiheuttaa virheen:

```
// define const user (JavaScript)
const user = {
    name: "Mikko",
    id: 0,
};


// define interface User (Type)
interface User {
    name: string;
    id: number;
}
```

```
// define const user as User (TypeScript)
const user: User = {
    name: "Mikko ",
    id: 0,
};

// define const user as User, with wrong property name -> error
const user: User = {
    username: "Mikko ", // username is not defined in type User
    id: 0,
};

// Error
/* Type '{ username: string; id: number; }' is not assignable to type 'User'.
Object literal may only specify known properties, and 'username' does not exist in
type 'User'. */
```

2.4 Tailwind

CSS eli *Cascading Style Sheets* on tyylien kuvauskieli (Mozilla, 2023c), jota käytetään verkkosivujen ja sovellusten ulkoasun ja visuaalisen tyylin määrittämiseen. Tailwind on avoimen lähdekoodin CSS-viitekehys ja työkalusarja (Tailwind, 2023), joka on suunniteltu helpottamaan ja nopeuttamaan verkkosivujen ja sovellusten käyttöliittymän suunnittelua ja rakentamista. Se eroaa monista perinteisistä CSS-kehyksistä, kuten Bootstrapista (Bootstrap, 2023) tai Foundationista (Foundation, 2023), sillä se keskittyy tarjoamaan joukon matalan tason tyyliluokkia, jotka voidaan yhdistää luomaan haluttu käyttöliittymä ilman valmiita komponentteja.

Tailwind CSS perustuu "utility-first" lähestymistapaan, mikä tarkoittaa, että se tarjoaa pieniä ja erittäin määriteltyjä CSS-luokkia, joita voit käyttää suoraan HTML:ssä (w3schools, 2023b). Tämä lähestymistapa mahdollistaa nopean suunnittelun ja tyylin määrittämisen. Tailwind tarjoaa valmiita CSS-luokkia kaikkiin yleisesti tarvittaviin tyylittelyihin, kuten marginaaleihin, reunuksiin, fontteihin, taustoihin, väreihin ja muihin. Se sisältää myös tyyliluokkia, jotka mahdollistavat mobiiliystävällisen suunnittelun eri näyttökokoille ja laitteille. Alla esimerkki tailwindin käytöstä:

```
// className attributes define css styles
<div className="p-2 flex justify-center items-center">
    <p className="font-bold tracking-tight text-gray-900">
        Hae tanssitapahtumia
    </p>
</div>
```

```
// Examples:
// p-2 = padding: 2px
// flex = display: flex
// justify-center = justify-content: center
// items-center = align-items: center
```

Tailwind CSS on mukautettavissa omiin tarpeisiin. Siihen voidaan määrittää omia tyyliluokkia ja teemoja projektien tarpeisiin. Tailwindin avulla voi helposti yhdistää tyyliluokkia ja luoda monimutkaisia käyttöliittymiä ilman tarvetta kirjoittaa paljon omaa CSS-koodia. Koska Tailwind tarjoaa valmiit tyyliluokat, voidaan säästää aikaa kirjoittamalla vähemmän CSS-koodia ja keskittymällä suoraan suunnitteluun ja rakentamiseen.

Työssä käytettiin myös Tailwindiin perustuvia avoimia komponenttikirjastoja nimeltä Flowbite (Flowbite, 2023) sekä Flowbite React (Flowbite-React, 2023). Nämä tarjoavat valmiita komponentteja kuten kortteja, modaleita, lomakkeita, bannereita yms. vastaavia yleisiä käyttökelpoisia elementtejä. Alla esimerkki flowbite-reacting korttikomponentista (Card) ja miten sitä voidaan käyttää suoraan kirjastosta (import)

```
// Example of Flowbite-react Card component
import { Card } from 'flowbite-react';

export default function DefaultCard() {
  return (
    <Card
      className="max-w-sm"
      href="#"
    >
      <div>{Card content}</div>
    </Card>
  )
}
```

2.5 SQL (Postgres)

SQL (*Structured Query Language*) on tietokantojen kyselykieli, jota käytetään tietojen hallintaan, muokkaamiseen, tallentamiseen ja hakemiseen relaatiotietokannoissa. SQL mahdollistaa tietokantojen tehokkaan käytön ja hallinnan, ja se tarjoaa standardoidun tavan kommunikoida tietokantojen kanssa. (geeksforgeeks, 2023b)

SQL-kieli koostuu erilaisista komentotyypeistä, joita käytetään tietokantojen rakenteen määrittämiseen (CREATE), tiedon lisäämiseen (INSERT), päivittämiseen (UPDATE) ja hakemiseen (SELECT), tietokantojen kyselyiden suorittamiseen (SELECT), taulujen poistamiseen (DROP) ja paljon muuhun.

Esimerkki SQL kyselystä: `SELECT * FROM Customers;`

Taulukko 1. Malliesimerkki SQL kyselyn tuloksista. (w3school.com, 2023a)

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

PostgreSQL (lyhennetty Postgres) on yksi suosituimmista avoimen lähdekoodin relaatiotietokannoista, jotka käyttävät SQL-kyselykieltä. Se on suunniteltu tarjoamaan kehittyneitä ominaisuuksia ja suorituskykyä monipuolisiin tietokantatarpeisiin. PostgresSQL tukee monia SQL-standardiin sisältyviä ominaisuuksia sekä lisäksi omia laajennuksiaan. PostgreSQL on laajalti käytetty tietokanta monenlaisissa sovelluksissa, kuten verkkosovelluksissa, mobiilisovelluksissa, suurissa yritystietokannoissa ja paljon muissa. Sen avoimen lähdekoodin luonne ja vahva ominaisuusvalikoima ovat tehneet siitä suositun vaihtoehdon monille tietokantatarpeille. (postgresql.org, 2023)

2.6 Kysely (ORM)

Kysely on ORM-ohjelmointitekniikka (*Object-Relational Mapping)*, joka mahdollistaa relaatiotietokantojen ja objektisuuntautuneen ohjelmakoodin välisen vuorovaikutuksen muuntamalla tietokantataulujen tiedot ohjelmointikielen objekteiksi ja päinvastoin (Kysely, 2023). Se mahdollistaa tietokannan taulujen ja ohjelmointikielten olioiden välisen kartoituksen (*mapping*), jolloin tietokantaoperaatioita voidaan suorittaa käyttäen objekteja eikä perinteisiä SQL-kyselyitä. ORM käyttö on tietoturvallisempaa ja vähentää virheiden määrää, kuin suoran SQL kyselyn käyttö. (Wikipedia, 2023c)

2.7 Visual Studio Code

Visual Studio Code (VS Code) on ilmainen ja avoimen lähdekoodin tekstieditori ja integroitu kehitysympäristö (IDE), joka on kehitetty Microsoftin toimesta. Se on suunniteltu erityisesti ohjelmistokehittäjille ja tarjoaa monia ominaisuuksia, jotka tekevät koodin kirjoittamisesta, muokkaamisesta ja hallinnasta helpompaa ja tehokkaampaa. (Microsoft, 2023)

2.8 Github

GitHub on web-pohjainen alusta (github, 2023), joka tarjoaa versionhallinnan ja yhteistyöominaisuudet ohjelmistokehittäjille. Se on erityisen suosittu paikka, jossa kehittäjät voivat tallentaa, hallinnoida ja jakaa ohjelmistoprojektejaan sekä työskennellä yhdessä muiden kehittäjien kanssa.

GitHub käyttää Git-versionhallintaa (Git, 2023), joka mahdollistaa projektin historian ja muutosten seuraamisen ajan myötä. Git mahdollistaa eri versioiden luomisen, muutosten seuraamisen ja haaroittumisen. Se tarjoaa mahdollisuuden luoda sekä julkisia että yksityisiä varastoja (*repository*), joissa voit tallentaa koodiprojektejasi.

GitHub mahdollistaa yhteistyön eri kehittäjien välillä. Voit kutsua muita käyttäjiä projektiin, tarkastella toistenne muutoksia, kommentoida koodia ja tehdä yhteistyötä. GitHubissa voi seurata ja raportoida vikoja, ominaisuuspyyntöjä ja muita projektin kehittämiseen liittyviä asioita. GitHub-integraatioiden avulla voit liittää CI/CD-palveluita, jotka mahdollistavat automaattisen testauksen, koodin tarkastuksen ja jatkuvan julkaisun.

GitHub on tullut merkittäväksi ohjelmistoyhteisöjen ja -yritysten keskuudessa, ja se tarjoaa tehokkaan tavan ylläpitää, kehittää ja jakaa ohjelmistoprojekteja avoimesti tai yksityisesti.

2.9 ChatGPT

ChatGPT on OpenAI:n kehittämä GPT (Generative Pre-trained Transformer) -arkkitehtuuriin perustuva tekoäly (OpenAI, 2023), joka on suunniteltu erityisesti chat-sovelluksia ja keskustelupohjaista vuorovaikutusta varten. Se pystyy tuottamaan luonnollisen kielen tekstejä ja vastaamaan käyttäjän kysymyksiin tai aloitteisiin keskustelumuodossa. Koulutusdatansa ansiosta ChatGPT voi vastata monenlaisiin kysymyksiin ja tuottaa informaatiota monista eri aiheista. Sen tietopohja kattaa tiedot siihen mennessä, kun malli viimeksi koulutettiin, mutta se ei ole tietoinen tapahtumista koulutuksen jälkeen. Kirjoitushetkellä ChatGPT käyttää vuoden 2021 ja sitä aiempia tietoja. Vaikka ChatGPT on optimoitu keskusteluun, se perustuu yleiseen GPT-arkkitehtuuriin, mikä tarkoittaa, että se voi tuottaa tekstiä monissa eri formaateissa, kuten runoja, tarinoita, tiivistelmiä ja jopa koodikieltä.

ChatGPT:tä käytettiin työn aikana erilaisissa tilanteissa auttamaan tekijää ongelmatilanteissa, antamaan kehitysehdotuksia, muodostamaan testidataa sekä koodia. Esimerkki ChatGPT:n käytöstä ohjelmoinnissa (Huom. Vastausta on lyhennetty niin, että vain olennainen osa on liitetty tähän):

Kysely: *Tee yksinkertainen esimerkki React korttikomponentista käyttäen tailwind css kirjastoa*
Vastaus*: Luo Kortti-komponentti:*
Tiedosto: CardComponent.js

```
import React from 'react';

const CardComponent = ({ title, description, imageUrl }) => {
  return (
    <div className="bg-white rounded-lg shadow-md p-4 max-w-sm">
      {imageUrl && <img src={imageUrl} alt="Card thumbnail" className="w-full h-48
object-cover mb-4 rounded-t-lg"/>}
      <h2 className="text-xl font-bold mb-2">{title}</h2>
      <p className="text-gray-600 mb-4">{description}</p>
    </div>
  );
};

export default CardComponent;
```

3 Next.JS (versio 13.4.+)

Next.js on React-pohjainen viitekehys, joka on suunniteltu rakentamaan kattavia full-stack web-sovelluksia. Se yhdistää React-komponentit käyttöliittymien luomiseen samalla tarjoten lisäominaisuuksia ja optimointeja. Next.js konfiguroi automaattisesti Reactille tarvittavat työkalut, kuten niputuksen ja kääntämisen. Tämä vapauttaa kehittäjän huolehtimasta monimutkaisista asetuksista ja antaa keskittyä itse sovelluksen rakentamiseen. Riippumatta siitä, onko kehittäjä freelancer vai osa suurempaa tiimiä, Next.js tarjoaa tehokkaat työkalut dynaamisten ja nopeiden React-sovellusten luomiseen. Lisäksi sen hybridirenderöintikyky mahdollistaa sekä staattisen että palvelinpohjaisen sisällön toimittamisen, ja intuitiivinen tiedostopohjainen reititys tekee sivujen lisäämisestä vaivatonta. (Next.JS, 2023c)

Kirjoitushetkellä Next.JS versio on 13.4.19. Syksyllä 2022 julkaistiin versio 13.0.0, missä App reitittimen beta-versio julkaistiin ja toukokuussa 2023 julkaistiin versio 13.4, minkä myötä stabiili App reititin julkaistiin virallisesti. Tämä työ aloitettiin huhtikuussa 2023, joten siirryin suoraan uusista uusimpaan Next.JS -teknologiaan.

3.1 Pages router vs App router

Next.JS tarjoaa kaksi erilaista sisäänrakennettua menetelmää tai järjestelmää, jotka vaikuttavat huomattavasti sen toimintaan ja suorituskykyyn: App sekä Pages reititin (*router*).

App reititin on kehittyneempi menetelmä (Next.JS versio 13+), joka tarjoaa enemmän ominaisuuksia kuin Pages reititin, ja se on räätälöitävissä kehittäjän tarpeiden mukaan. App reitittimessä jokainen sivu ja komponentti on lähtökohtaisesti serveripuolen komponentti, ellei kehittäjä sitä toisin määrittele koodissaan.

Pages reititin on helppokäyttöinen ja suorituskykyinen Next.JS:n menetelmä. Se käyttää pääasiallisesti staattisia tiedostoja sivujen käsittelyyn, mikä tarkoittaa, että sivut ladataan palvelimelta ennakkoon. Tämä parantaa suorituskykyä ja vähentää HTTP-pyyntöjen määrää. Pages reititin on Next.JS:n vanhempi käytäntö (oletuksena ennen Next.JS versiota 13), ja se on edelleen tuettu tapa, mutta uudemmissa versiossa App reititin on suositeltu menetelmä.

Joissain tilanteissa on helpompi puhua reitittimen sijasta hakemistosta (*pages/app directory*), sillä Pages reititin toimii vain ”pages”-nimisen kansion alla, ja App reititin vain ”app”-nimisen kansion alla. On myös mahdollista käyttää molempia rinnakkain, jolloin osassa projektia (osa sivuista) käytetään pages reitittimen ominaisuuksia ja toisaalla taas app reitittimen ominaisuuksia. Tämä on erityisesti mahdollista silloin, kun vanhasta järjestelmää halutaan päivittää kohti uutta pieni pala kerrallaan. Projektin struktuurista lisää seuraavassa kappaleessa (3.2).

Tähän työhön on valittu uudempi App reititin, minkä vuoksi käsittelen sen toimintoja tarkemmin seuraavissa kappaleissa.

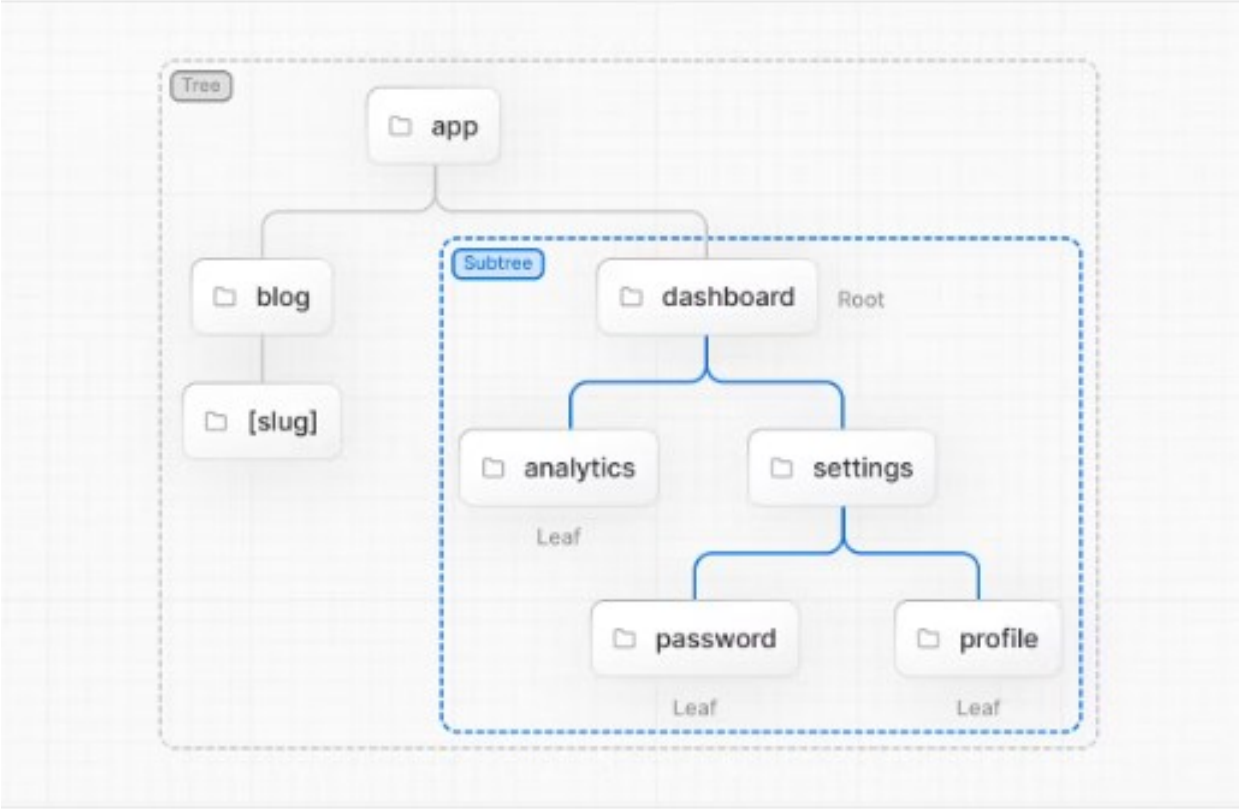
3.2 Projektin struktuuri ja reititys (app reititin)

Tässä osiossa käydään läpi, miten Next.JS-projektin struktuuri sekä reititys rakentuu käytettäessä app reititintä. Osion tueksi

Taulukko 2 on annettu tärkeimmät termit ja niiden merkitykset.

Taulukko 2. Next.JS-projektin terminologiaa

Nimi suomeksi	Nimi englanniksi	Merkitys
Puu	Tree	Hierarkkinen rakenne. Esimerkiksi komponenttipuu vanhempien ja lasten komponenteilla, kansiorakenne jne.
Alipuu	Subtree	Osa puuta, joka alkaa uudesta juuresta (ensimmäinen) ja päättyy lehtiin (viimeinen).
Solmu	Node	Yksittäinen elementti tai piste hierarkkisessa rakenteessa, kuten puussa tai verkossa.
Juuri	Root	Ensimmäinen solmu puussa tai alipuussa, kuten juuriasettelu (root layout).
Lehti	Leaf	Solmu alipuussa, joilla ei ole lapsia, kuten URL-polun viimeinen segmentti.
URL-segmentti	Segment	Osa URL-polusta, joka on rajattu kauttaviivoilla.
URL-polku	Path	Osa URL-osoitteesta, joka tulee domainin jälkeen (koostuu segmenteistä).

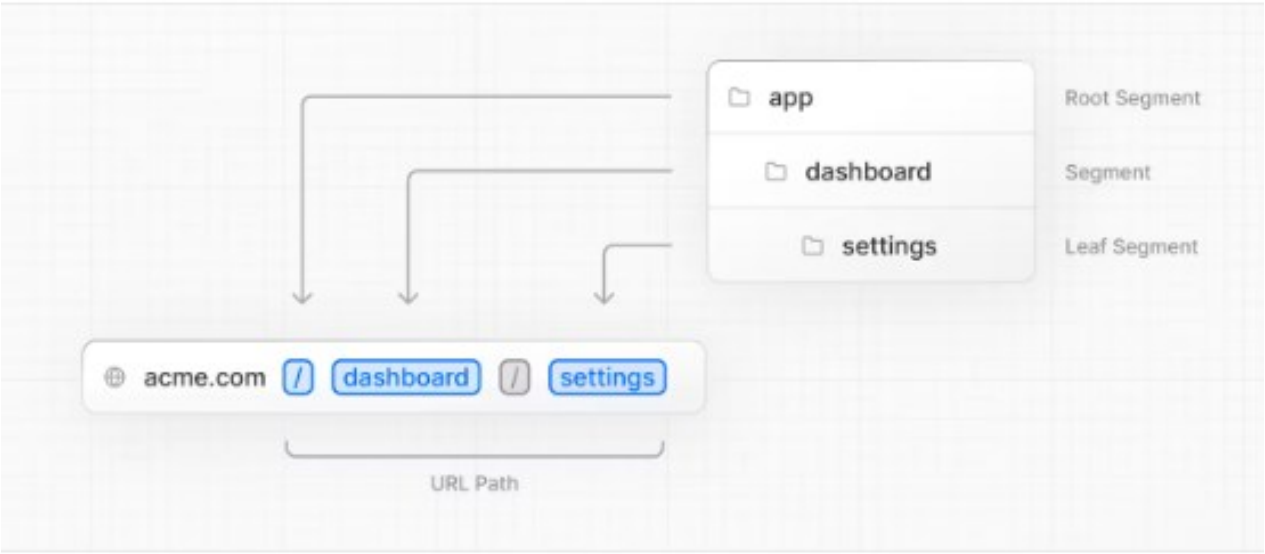


Kuva 5. Kuvakaappaus Next.JS dokumentaatiosta, missä esiintyy esimerkki hierarkkisesta rakenteesta (Next.JS, 2023f)

Kuva 6. Kuvakaappaus Next.JS dokumentaatiosta, missä esiintyy esimerkki URL rakenteesta (Next.JS, 2023f)

3.2.1 Reitittäminen sekä tiedostorakenteet

Next.JS käyttää hyväkseen sisäänrakennettua tiedostopolkuhierarkiaa, minkä avulla reititys tapahtuu ilman tarvetta konfiguroida reititintä erikseen. Jokainen kansio on liitoksissa suoraan reitittimeen siten, että kansio muodostaa aina segmentin url-polkuun (Kuva 7).



Kuva 7. Kuvakaappaus Next.JS dokumentaatiosta, missä esiintyy esimerkki kansiohierarkian ja polun suhteesta toisiinsa (Next.JS, 2023f)

URL-polku ei kuitenkaan toimi ilman varsinaista tiedostoa, mikä muodostaa sivun. App reitittimen sisäänrakennettu menetelmä käyttää hyväksi erilaisia spesiaalitiedostoja, joiden käyttäytyminen on sisäänrakennettuna App reitittimen toimintoihin. Esimerkiksi ”page”-niminen spesiaalitiedosto muodostaa automaattisesti sivun pääkomponentin, kun se sijaitsee minkä tahansa kansion sisällä (Kuva 8). URL ei toimi ilman tätä tiedostoa tai jos se on toisella nimellä.

```tsx
// app/page.tsx

export default function Page() {
  return <h1>Hello, Next.js!</h1>
}
```

Toinen pääkomponentti on ”layout”-niminen tiedosto, joka muodostaa layoutin eli pohja-asettelun sivukomponentille. Juuressa olevalla asettelulla määritetään käytännössä aina html-rakenne React komponenttien pohjalle sekä muita koko sivustolle yhteisiä komponentteja kuten esimerkiksi navigointikomponentti.

```tsx
// app/layout.tsx

export default function RootLayout({
  children,
}: {
  children: React.ReactNode
}) {
  return (
    <html lang="en">
      // <NavBar />
      <body>{children}</body>
      // children will be replaced by Page() component from app/page.tsx
    </html>
  )
}
```

Kuva 8. Kuvakaappaus Next.JS dokumentaatiosta, missä page.tsx tiedosto app-kansion (juuri) sisällä muodostaa sivun pääkomponentin, jolloin url-osoite toimii juuripolussa ”/”. layout.tsx tiedosto luo pohja-asettelun page.tsx sisällä olevalle pääkomponentille (Next.JS, 2023b)

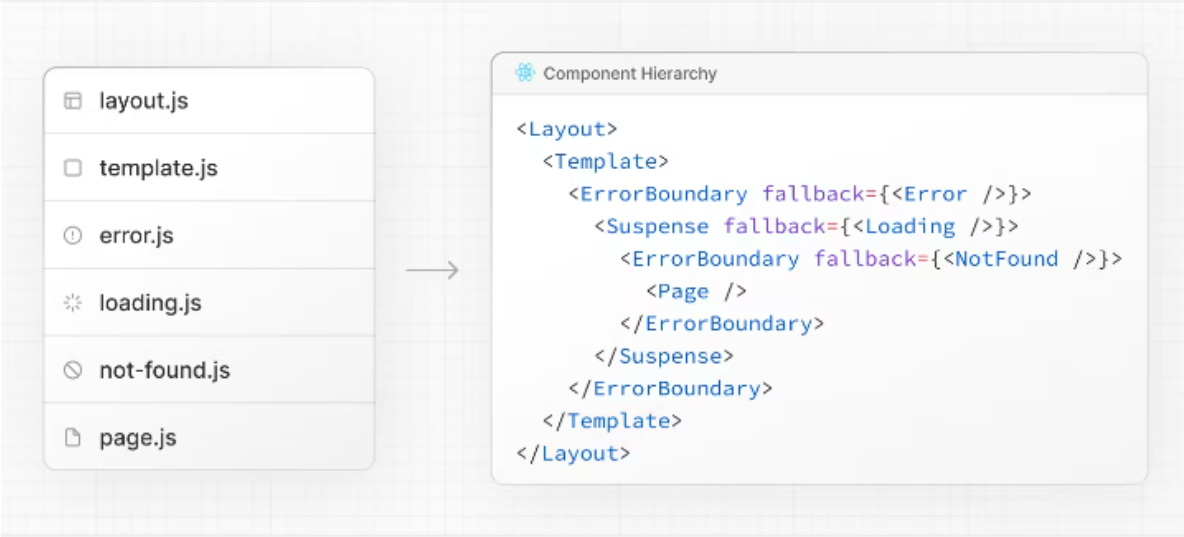
Page ja layout ovat pääkomponentit, joita on pakko käyttää, mutta App kansion sisällä voi käyttää myös muita spesiaalitiedostoja (Taulukko 3). Loading-sivu on kätevä väliaikainen sivu/komponentti, jota näytetään käyttäjälle, kun jotain sivua/komponenttia haetaan api-kutsuilla. Error-sivua voidaan

käyttää silloin, kun sivulla tai komponentissa tapahtuu jokin virhe. Loading- ja error-sivuja voidaan käyttää myös komponenttitasolla, eli jos yksittäinen komponentti on loading/error -tilassa, niin sen tilalla näytetään jotain sen tilasta käyttäjälle siten, että muut sivun komponentit toimivat normaalisti. Route-tiedosto muodostaa API-reitin siihen kansioon, missä se sijaitsee samalla tavalla, kuin page muodostaa julkisen sivun siihen kansioon missä se sijaitsee.

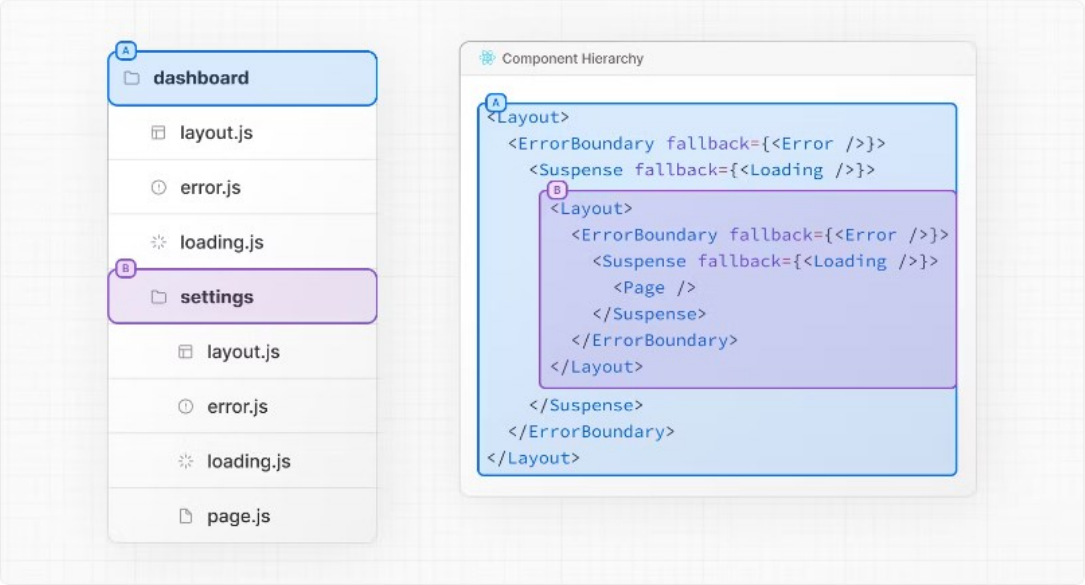
Taulukko 3. App kansion sisällä käytössä olevat spesiaalitiedostot.

Nimi	Käyttötarkoitus
layout	Jaettu pohja-asettelu segmentille ja sen lapsille
page	Reitin yksilöllinen käyttöliittymä ja tekee reitit julkisesti saataville
loading	Latauskäyttöliittymä segmentille ja sen lapsille
not-found	Ei löytynyt -käyttöliittymä segmentille ja sen lapsille
error	Virhekäyttöliittymä segmentille ja sen lapsille
global-error	Globaali virhekäyttöliittymä
route	Palvelinpuolen API-päätepiste
template	Erikoistunut uudelleenrenderoituva Layout-käyttöliittymä
default	Varakäyttöliittymä rinnakkaisille reiteille

Näitä spesiaalitiedostoja voi ja kuuluukin käyttää rinnakkain saman kansion sisällä, jolloin ne toimivat hierarkkisesti. Kuva 9 näyttää miten komponenttihierarkia toimii app kansion sisällä. Sisäkkäiset kansiot (reitit) muodostavat myös hierarkkisen komponenttirakenteen (Kuva 10).

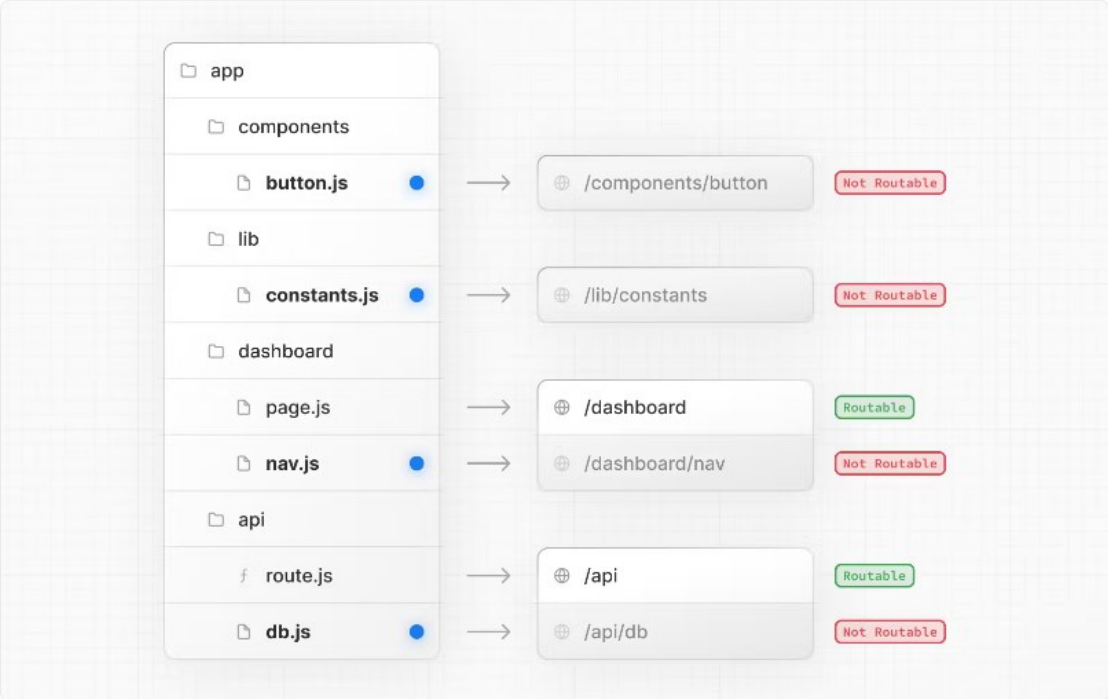


Kuva 9. Kuvakaappaus Next.JS dokumentaatiosta, missä esiintyy komponenttihierarkkia (Next.JS, 2023f)



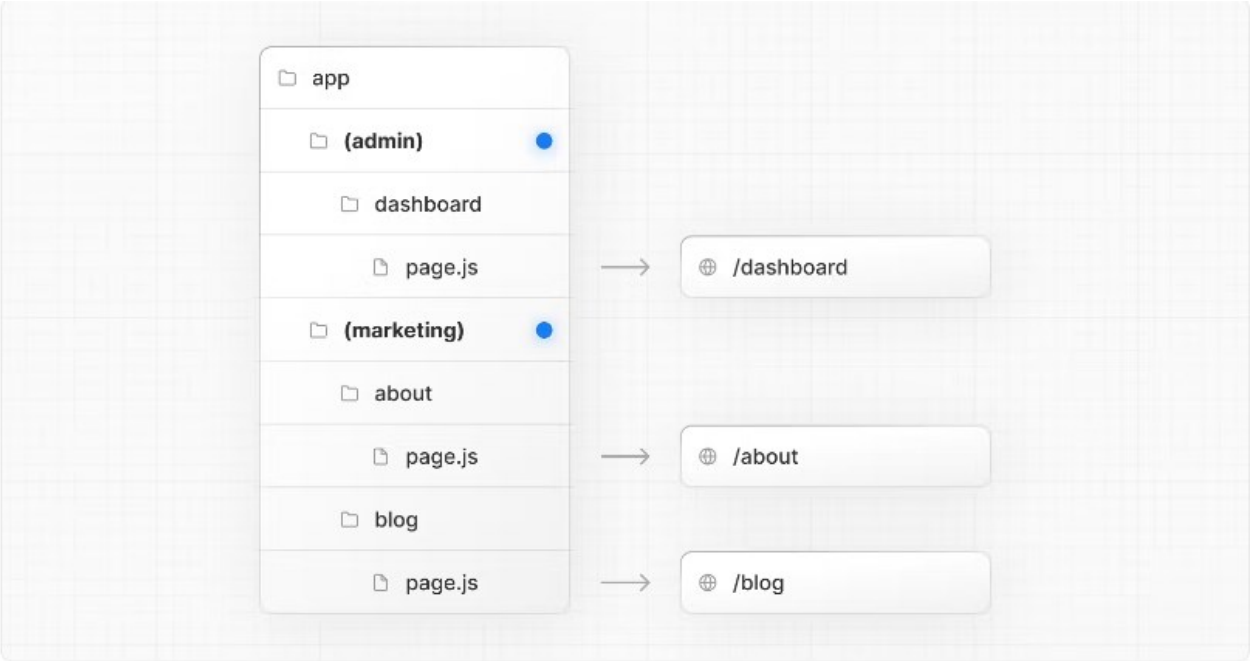
Kuva 10. Sisäkkäisten reittien komponenttihierarkia (Next.JS, 2023f)

Muitakin perustiedostoja voidaan sijoittaa samaan kansioon, eivätkä ne häiritsi reititystä tai hierarkiaa. Kuten aiemmin todettiin, niin ainoastaan tiedostot page.tsx tai route.tsx voi muodostaa julkisen reitin. Tästä syystä komponentteja voidaan sijoittaa sekä suoraan samoihin kansioihin, missä sivun pääkomponenttia pidetään, tai erikseen omaan komponenttikansioon, mihin ei ole muodostettu reittiä (Kuva 11).



Kuva 11. Kuvakaappaus Next.JS dokumentaatiosta, missä esiintyy komponenttien yhteiskäyttö (*colocation*) samojen kansioiden alla ja miten ne vaikuttavat reititykseen. Vain tiedosto page.tsx muodostaa toimivan url-reitin. (Next.JS, 2023f)

Yksi kätevä projektinhallintaan liittyvä tapa on tehdä ryhmittelyä (*route grouping*), missä sulkeiden sisälle kirjatut kansiot eivät muodosta url-segmenttiä. Tällöin voidaan mm. nimetä tiettyjen reittiryhmien kansiot selkeyttämään projektinhallintaa.



Kuva 12. Kuvakaappaus Next.JS dokumentaatiosta, missä esiintyy reittien ryhmittely (Next.JS, 2023d)

Dynaamiset reitit ovat myös osa Next.JS sisäänrakennettua reitityssysteemiä. Alla olevasta kuvasta voi nähdä miten sivun url parametreihin pääsee käsiksi ja miten parametreja muodostetaan käyttämällä hakasulkeita reittipoluissa kansion nimen ympärillä (Kuva 13).

Emme käy läpi jokaista reititysmallia tässä kohtaa, mutta mainittakoon vielä, että näiden lisäksi on olemassa muita kehittyneempiä reititysmalleja kuten rinnakkaiset reitit (*parallel routes*), jotka mahdollistavat useamman sivun näyttämisen samassa näkymässä sekä reittien ohjaukset (*interceptin routes*), jotka mahdollistavat reitin ohjaamisen ja sen näyttämisen toisen reitin kontekstissa.

app/blog/[slug]/page.tsx TypeScript

```tsx
export default function Page({ params }: { params: { slug: string } }) {
  return <div>My Post: {params.slug}</div>
}
```

Route	Example URL	`params`
`app/blog/[slug]/page.js`	`/blog/a`	`{ slug: 'a' }`
`app/blog/[slug]/page.js`	`/blog/b`	`{ slug: 'b' }`
`app/blog/[slug]/page.js`	`/blog/c`	`{ slug: 'c' }`

Route	Example URL	`params`
`app/shop/[...slug]/page.js`	`/shop/a`	`{ slug: ['a'] }`
`app/shop/[...slug]/page.js`	`/shop/a/b`	`{ slug: ['a', 'b'] }`
`app/shop/[...slug]/page.js`	`/shop/a/b/c`	`{ slug: ['a', 'b', 'c'] }`

Kuva 13. Kuvakaappaus Next.JS dokumentaatiosta, missä esiintyy esimerkki dynaamisesta reitityksestä. Yksittäinen [slug] vastaa aina yhtä parametria, mutta jos käytetään […slug] tapaa, niin se vastaa listaa parametreista. Toisin sanoen […slug] on X määrä erillisiä [slug] parametreja. (Next.JS, 2023a)

3.3 Renderöinti

Yksi Next.JS tärkeistä ominaisuuksista on kyetä renderöimään sivuja serverillä (*backend*) ja käyttäjän (*frontend*) puolella. Lähtökohtaisesti jokainen sivu on serverikomponentti, mutta sivun voi määritellä olemaan myös client-komponentti, jos se vaatii sellaisia toimintoja, joita ei voida tehdä serverin puolella.

3.3.1 Serverikomponentit

Reactin serverikomponentit mahdollistavat käyttöliittymän luomisen, joka voidaan renderöidä ja mahdollisesti välimuistittaa palvelimella. Next.js:ssä renderöinti on jaettu reitin segmenteittäin, jotta

voitaisiin käyttää osittaista renderöintiä ja tiedon suoratoistoa. Next.js tarjoaa kolme erilaista serveripohjaista renderöintistrategiaa: Staattinen ja dynaaminen renderöinti sekä striimaus.

Staattinen renderöinti on oletusarvona toimiva strategia renderöi sivut jo siinä vaiheessa, kun ohjelma käynnistetään/rakennetaan (*build*) serverillä. Tämä on mahdollista, koska sivut ovat staattisia eivätkä muutu. Renderöity sivu tallennetaan kerran ja sen jälkeen sitä voidaan käyttää jokaiselle erilliselle käyttäjälle, kun he tarvitsevat sitä. Tällöin käyttäjän ei tarvitse käyttää resursseja renderöintiin.

Dynaaminen renderöinnin avulla reitit renderöidään siinä vaiheessa, kun sitä kutsutaan. Dynaamista renderöintiä käytetään silloin, kun sivun tietoja ei ole tallessa välimuistissa tai jos niin sanottuja dynaamisia funktioita on käytössä kyseisellä sivulla

Striimauksen avulla reitti voidaan jakaa pienempiin paloihin, jotka latautuvat käyttäjälle siinä järjestyksessä, kun ne ovat valmiina. Tämä mahdollistaa sen, että käyttäjä näkee nopeasti osan sivusta, kuten esimerkiksi layoutin ja varsinainen sisältö saapuu hetkeä myöhemmin.

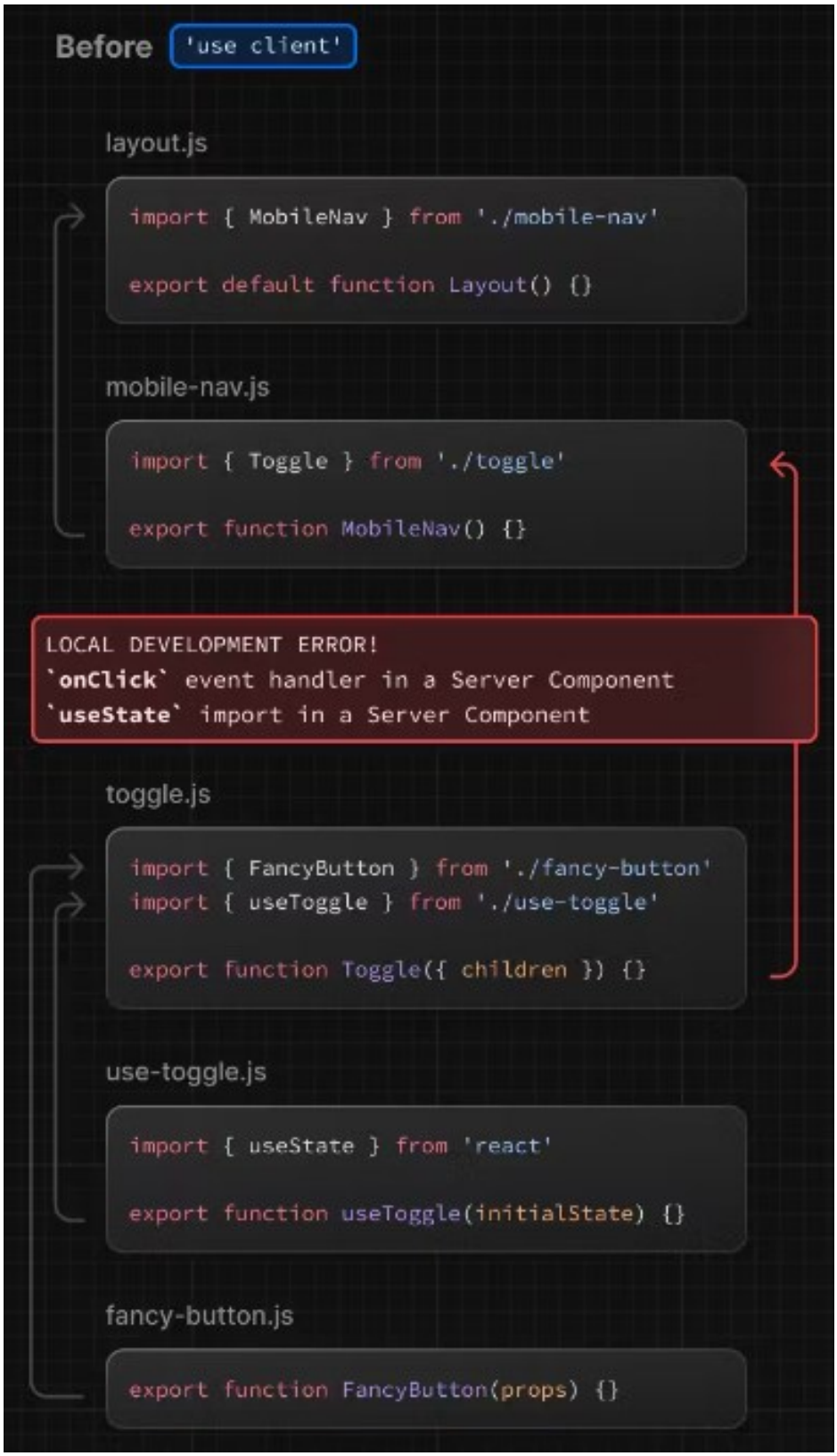
Serveripohjainen renderöinti tuo mukanaan mm. seuraavia etuja. Serverikomponentit mahdollistavat tiedonhaun siirtämisen palvelimelle, mikä voi nopeuttaa prosessia ja vähentää asiakkaan tekemien pyyntöjen määrää. Herkkä tieto ja logiikka, kuten API-avaimet, voidaan pitää palvelimella ilman riskiä niiden paljastumisesta asiakkaalle. Palvelimella renderöitäessä tulosta voidaan välimuistittaa (*cache*) ja käyttää uudelleen myöhemmissä pyynnöissä. Suuret riippuvuudet voidaan pitää palvelimella, mikä vähentää asiakkaalle lähetettävän JavaScriptin määrää. Renderöity HTML voidaan indeksoida hakukoneissa ja käyttää sosiaalisen median esikatseluissa.

3.3.2 Käyttäjäkomponentit (Client)

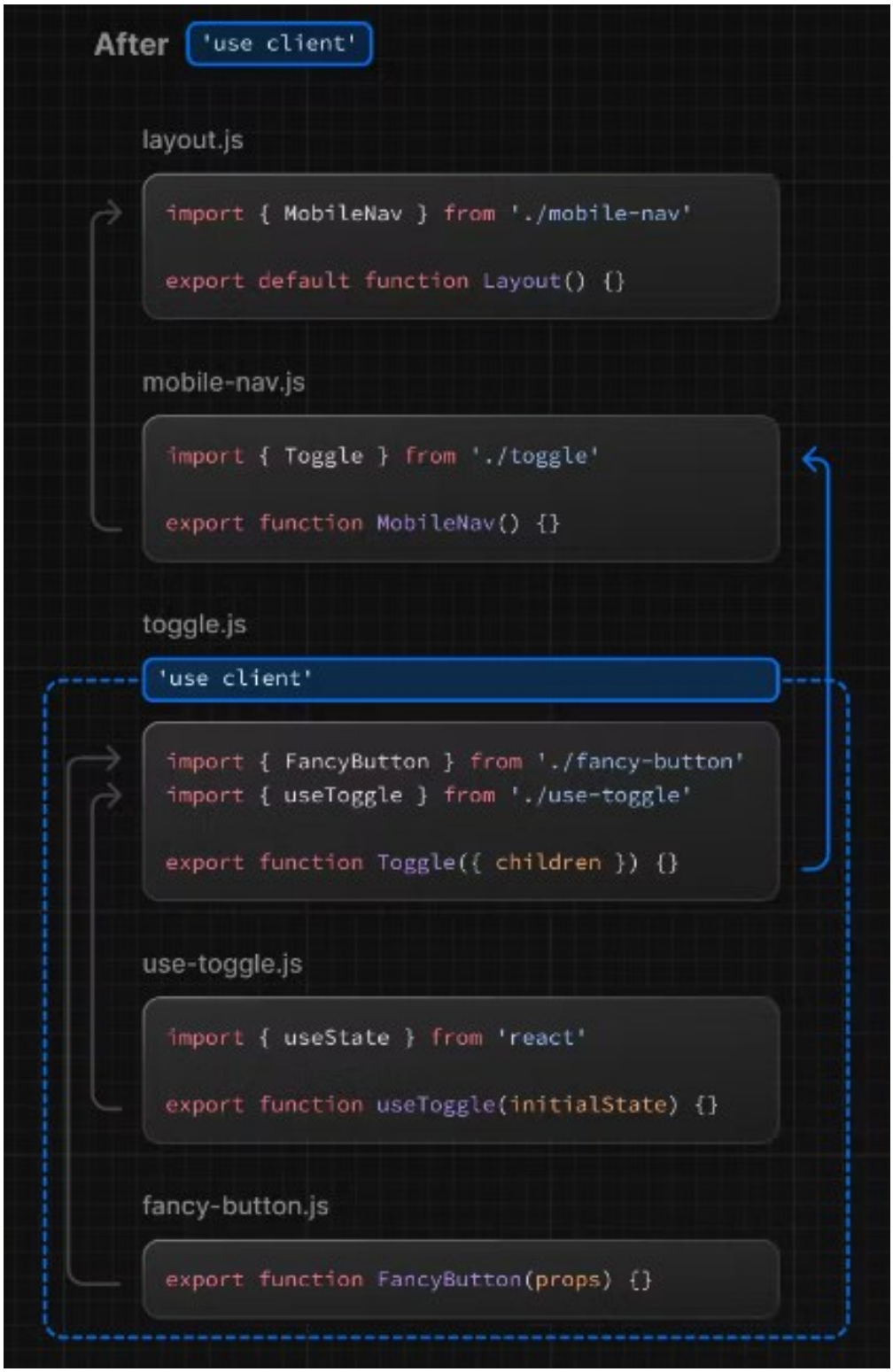
Käyttäjäkomponentit mahdollistavat interaktiivisen käyttöliittymän tekemisen, joita voidaan renderöidä käyttäjän kautta heti aktivointihetkellä. Käytännössä tällä tarkoitetaan perinteisen React-sivuston toimintaa, jossa käytetään erilaisia kytkimiä ja tilahallintaa. Esimerkiksi kaikki nappulat ovat reaktiivisia ja toteuttavat yleensä jonkin toiminnallisuuden, ja sen aiheuttama efekti näkyy käyttäjälle heti. Tällainen voi olla yksinkertainen laskuri, missä jokainen painallus kasvattaa tilahallinnassa olevaa arvoa yhdellä.

Käyttäjäkomponenttien käyttö app-reiteissä tapahtuu siten, että komponentin koodiosan alkuun laitetaan yksinkertaisesti teksti ’use client’. Tämän avulla Next.JS ymmärtää tehdä tästä käyttäjäkomponentin eikä yritä renderöidä tätä palvelimella.

```tsx
'use client'

import { useState } from 'react'

export default function Counter() {
  const [count, setCount] = useState(0)

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  )
}
```



Kuva 14. Kuvakaappaus Next.JS dokumentaatiosta, missä esiintyy käyttäjäkomponentin määritys ilman 'use client' -tekstin käyttöä. Komponentit, missä on käytetty Reactin käyttäjäpuolen toimintoja, eivät toimi ilman ’use client’ määritystä. (Next.JS. 2023g)



Kuva 15. Kuvakaappaus Next.JS dokumentaatiosta, missä esiintyy käyttäjäkomponentin määritys käyttäen 'use client' -tekstiä. Komponentit, missä on käytetty Reactin käyttäjäpuolen toimintoja, ovat nyt toimintakuntoisia, koska ne on määritetty ”client”-komponenteiksi. (Next.JS. 2023g)

3.4 API ja tiedonhakupyynnöt (Fetch)

Usein kahden ohjelman rajapintana toimii API (*Application Programming Interface*) eli sovellusohjelmointirajapinta, jonka avulla ohjelmat voivat keskustella toistensa kanssa. Usein ohjelmistoissa on frontend (*client*), joka keskustelee backendin (*server*) kanssa. Tämän kautta voidaan tehdä erilaisia toimintoja kuten tietokantakyselyjä.

Web-kehityksessä Fetch API on työkalu, joka mahdollistaa resurssien, kuten tiedostojen, kuvien ja datan, hakemisen verkon kautta. Fetch käyttää niin sanottujen pyyntöjen (*Request*) ja vastauksien (*Response*) määrittämiä objekteja tiedon hallinnassa. Fetch() metodi lähettää pyynnön, jonka ainoa vaadittu parametri on polku eli käytännössä jokin url-osoite mihin pyyntö lähetetään. Metodi palauttaa lupauksen (*Promise*), joka ratkeaa myöhemmin vastaukseksi, heti kun palvelin vastaa antaen tarvittavan header-datan. (Mozilla, 2023a)

Fetch-metodilla voidaan siis pyytää tietoa jostain toiselta palvelimelta. Kun vastaus on saatu, voidaan tätä käyttää sovelluksessa. Vastaus voi myös olla jokin virheviesti, jolloin tähän voidaan koodin kautta reagoida. Esimerkki suorasta fetchauksesta serverikomponentin kautta:

```
async function getData() {
  const res = await fetch('https://api.example.com/...')
  // The return value is *not* serialized
  // You can return Date, Map, Set, etc.

  if (!res.ok) {
    // This will activate the closest `error.js` Error Boundary
    throw new Error('Failed to fetch data')
  }

  return res.json()
}

export default async function Page() {
  const data = await getData()

  return <main></main>
}
```

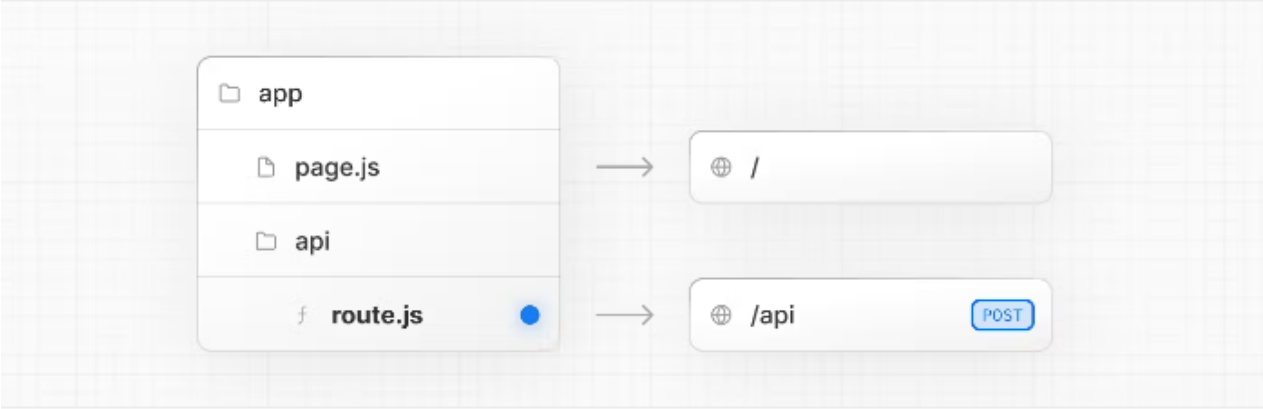
App reitittimen sisällä on neljä eri tapaa suorittaa hakea tietoa muista palvelimista:

1. Suoraan serveriltä käyttäen fetch() -metodia
2. Suoraan serveriltä käyttäen kolmannen osapuolen kirjastoja
3. Käyttäjän puolelta käyttäen reittikäsittelijää
4. Käyttäjän puolelta käyttäen kolmannen osapuolen kirjastoja

Periaatteessa käyttäjän puolelta voidaan hakea myös suoraan fetch()-metodilla, mutta tähän liittyy tietoturvariskejä, joten on parempi käyttää reittikäsittelijää eli Next.JS:n sisäistä API-järjestelmää. Palvelimen puolella voidaan käyttää fetch()-metodia suoraan, koska käyttäjä ei pääse käsiksi palvelimella oleviin tietoihin.

3.4.1 Reittikäsittelijä (Route handler) ja sisäinen API

Edellisessä osassa mainitsin Route-tiedostosta eli reittikäsittelijästä (*route handler*), jolla voidaan muodostaa api-endpoint haluttuun reittiin. Tätä url-osoitetta kutsuessa tapahtuu route-tiedostossa määritetyt toiminnot (Kuva 16). Vaikka api-reitit ovat samassa tiedostohierarkiassa, ne ovat silti toiminnassa vain palvelimella. Route ja page -tiedostot eivät voi olla samassa kansiossa, sillä niiden välille syntyy konflikti.



Kuva 16. Kuvakaappaus Next.JS dokumentaatiosta, missä esiintyy tyypillinen API-reitti. Tässä route.js tekee url-osoitteesta julkisen api-endpointin. (Next.JS, 2023e)

Reittikäsittelijä tukee seuraavia http-metodeja: GET, POST, PUT, PATCH, DELETE, HEAD ja OPTIONS. Yhteen reittikäsittelijään voi sisällyttää oman funktionsa jokaiselle metodille, jolloin sitä funktiota käytetään, mitä pyynnön metodissa halutaan. Next.JS käyttää omaa NextRequest ja NextResponse -objekteja, jotka ovat laajennettuja natiiveista Request ja Response -objekteista.

Reittikäsittelijä tukee myös dynaamista mallia samalla tavalla, miten Kuva 13 esimerkissä oli käsitelty normaalien sivujen tapauksessa.

Seuraavassa vielä esimerkki reittikäsittelijästä:

```
// app/api/route.ts

import { NextResponse } from 'next/server'

// GET method
export async function GET() {
  // API handling
  const res = await fetch('https://data.mongodb-api.com/...', {
    headers: {
      'Content-Type': 'application/json',
      'API-Key': process.env.DATA_API_KEY,
    },
  })
  const data = await res.json()

  return NextResponse.json({ data })
}

// POST method
export async function POST() {
  // API handling
  const res = await fetch('https://data.mongodb-api.com/...', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'API-Key': process.env.DATA_API_KEY,
    },
    body: JSON.stringify({ time: new Date().toISOString() }),
  })
  const data = await res.json()

  return NextResponse.json(data)
}

// PUT method
export async function PUT() {
}
```

3.5 Middleware

Middleware mahdollistaa koodin suorittamisen ennen kuin pyyntö on valmis (pyyntö voi olla esimerkiksi url-osoitteen nouto eli sivun avaaminen). Sen jälkeen, riippuen saapuvasta pyynnöstä, voit muokata vastausta kirjoittamalla uudelleen, ohjaamalla uudelleen, muokkaamalla pyynnön tai vastauksen otsikoita tai vastaamalla suoraan. Tämän avulla voidaan suorittaa esimerkiksi erilaisia validaatioita, kuten tarkastaa onko käyttäjä kirjautunut sisään ja tämän perusteella joko päästää tai olla päästämättä tietyille sivuille.

4 Toiminnallisen työn kuvaus

Tässä osassa kuvataan työn eri vaiheita ja työn toteutusta. Loppuosassa nähdään kuvia valmistuneesta applikaatiosta.

4.1 Määrittelyvaihe

Työn määrittely tehtiin kolmessa osassa. Ensimmäisessä osassa oma kokemukseni tanssimaailmasta ja tieto siitä, mitä tanssiharrastajan tarvitsevat, oli hyvin tiedossa, ja sen perusteella osasin määritellä suurin piirtein mitä olen sovelluksellani ajamassa takaa. Pääasialliset toiminnallisuudet olivat tietojen hakuun perustuvat perustoiminnot, joidenka avulla käyttäjä pystyy hakemaan tapahtumia eri parametrien pohjalta ja mielestäni tärkeimpänä se, että tapahtumat näkyvät kartalla sekä listana.

Verrokkisivustona käytin määrittelyvaiheessa (sekä myöhemmin) timma.fi -sivustoa (Timma.fi, 2023), joka on erilaisten kosmetiikka- ja hoitoalan palveluiden hakusivusto, mistä voi mm. hakea parturi-, hieronta-, tai vaikka kynsienhoitopalveluita. Sivustolla on perusperiaatteena vastaava palvelu, missä käyttäjä voi etsiä eri parametreilla sopivaa palvelua eri paikkakunnilta.

Toisessa osassa haastattelin Antti Törmästä, jolta sain tärkeää tietoa siitä, miten tapahtumajärjestäjät tällä hetkellä syöttävät tietoa erilaisiin palveluihin sekä siitä, mitä tapahtumajärjestäjät kaipaavat. Haastattelussa kävi ilmi muutama pääasiallinen toiminnallisuus, jotka olivat helppokäyttöisyys, nopea tiedon syöttö sekä erityisesti tietojen muokkaamismahdollisuudet. Nykyiset järjestelmät, joita tapahtumajärjestäjät käyttävät ovat lähinnä staattisia sivustoja, joihin tiedot lähetetään ja palveluntuottaja lisää ne sivustoilleen. Tästä syystä koin ehdottoman tärkeäksi rakentaa käyttöliittymän, josta tietoa pääsee syöttämään, muokkaamaan ja poistamaan.

Kolmas vaihe oli hankalin. Siinä piti miettiä mitkä kaikki asiat ja toiminnallisuudet halutaan tässä vaiheessa sovellukseen, ja että kuinka paljon työmäärällisesti pystyn tekemään ja mikä on järkevää opinnäytetyön puitteissa. Kuten aiemmin jo totesin, seuratanssipiirit ovat suuremmat kuin vain lavatanssit ja siihen liittyvät tanssikurssit. Päätin kuitenkin lähteä tekemään lavatansseihin liittyvää sovellusta, koska se oli sopivan laaja, mutta selkeästi rajattu yhteisö. Lavatanssit ovat selkeämpi tietorakenteeltaan, sillä siihen liittyy pääsääntöisesti vain paikka, aika, esiintyjät ja hinta. Tanssikurssit ovat hieman mutkikkaampia ja vaativat hieman enemmän miettimistä, sillä siihen liittyy enemmän detaljeja. Halusin nämä kaksi molemmat, sillä ne toimivat käsi kädessä keskenään. Lisäksi Tanssikoulu Antti Törmäsen pääsääntöisen tulolähde on nimenomaan tanssikurssit, joten oli sopivaa molempien osapuolien kannalta implementoida tanssikurssit.

4.2 Suunnittelu ja toteutus

Tanssiapplikaation suunnittelu ja toteutus kulkeutuivat projektin aikana käsi kädessä. Alustavassa suunnitelmassa mietittiin pääpiirteittäin sivuston rakenne ja perustoiminnallisuudet. Projektia ei suunniteltu tarkasti alkuvaiheessa, vaan sen annettiin elää sitä myötä, kun koodia puskettiin eteenpäin. Tällä tekniikalla haluttiin vauhdittaa alkuun pääsemistä, sekä pitämään suunnitelma joustavana.

Ennen työn aloittamista olin päättänyt, että työ tehdään Next.JS teknologialla ja koodaus tapahtuu Typescript/React-kombinaatiolla. Tämä yhdistelmä on erittäin kovassa suosiossa tällä hetkellä ja halusin ehdottomasti opetella nämä modernin web-ohjelmoinnin tekniikat. Todellisuudessa oma taitotasoni Next.JS:n suhteen oli käytännössä olematon, joten opeteltavaa oli paljon. Onneksi React ja typescript olivat kuitenkin jotenkin tuttuja, mutta eivät nekään ihan selkärangasta tulleet. Tämä oli myös yksi syy sille, miksi en halunnut määritellä tai suunnitella applikaatiota liian tarkasti, sillä en tiennyt tarkalleen mihin pystyn (tai mihin next.js pystyy). Applikaatio muokkautui jatkuvasti sitä myötä, kun opin lisää ja aloin ymmärtämään miten esimerkiksi Next.JS alustana toimii. Kaiken lisäksi Next.JS:stä oli julkaistu täysin uusi iso päivitys (13.4+), jonka myötä se tarjosi kaksi rinnakkaista tapaa rakentaa applikaatiota (pages vs. app directory), jotka ovat rakenteeltaan erilaisia ja muuttaa suuresti, miten sen sisäisiä tekniikoita implementoidaan. Valitsin näistä uudemman (app directory), koska halusin oppia jotain täysin uutta (osaksi sen takia että sen avulla voisi saada paremman aseman esimerkiksi työnhaussa). Tämä johti siihen, että tietoa oli rajoitetusti tarjolla ja haasteet kasvoivat, sillä esimerkkejä tai foorumipalstojen vinkkejä ei vielä ollut näin uudelle päivitykselle, mikä tarkoitti myös sitä, että ChatGPT ei esimerkiksi pystynyt auttamaan lähes ollenkaan Next.JS:n viimeisen päivityksen kanssa.

4.2.1 Projektin käynnistäminen

Ensimmäisenä tehtiin uusi kansio projektille omalla koneella ja käynnistettiin Visual Studio Code. Avattiin uusi konsoli ja navigoitiin projektikansioon. Konsolissa ajettiin seuraava koodipätkä uuden Next.JS -projektin luomiseksi:

```
npx create-next-app@latest
```

Seuraavaksi avataan github selaimesta ja luodaan omalle tililleni uusi privaatti varasto. Kopioidaan varaston URL ja käytetään sitä projektin yhdistämiseen kyseiseen varastoon:

```
git init
git add .
git commit -m ”initial commit”
```

```
git remote add origin https://github.com/your-username/your-repository.git
git push -u origin master
```

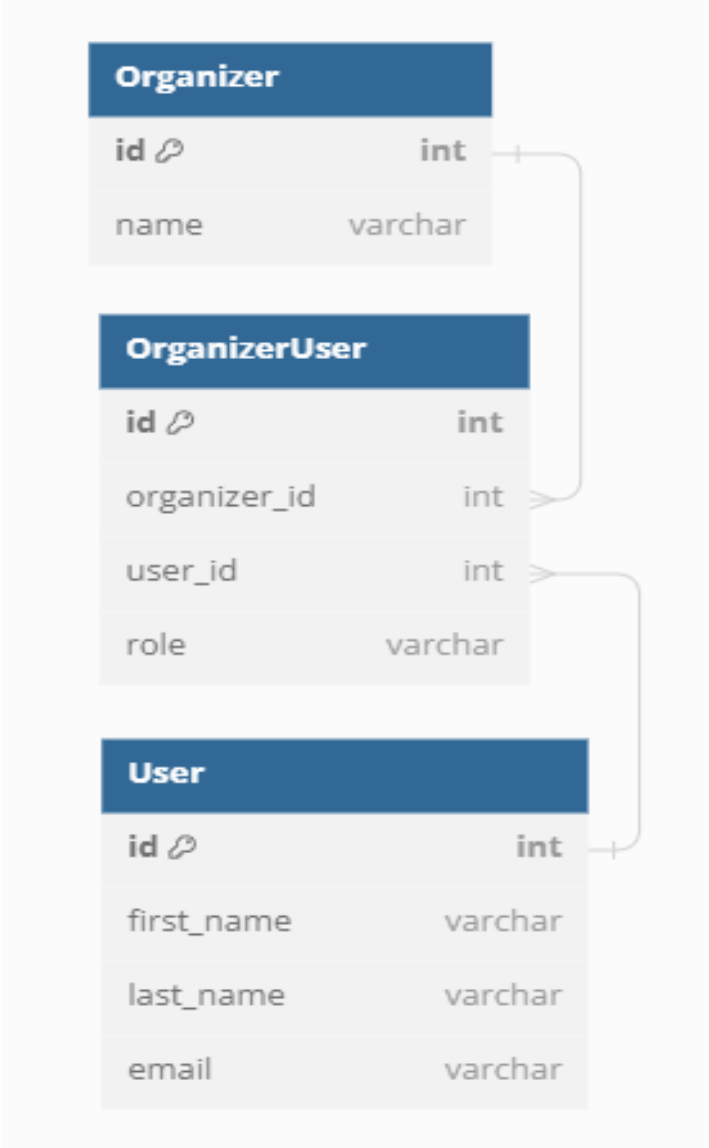
Käynnistetään uusi projekti Vercelin alustalla ja linkitetään se githubin. Aina kun varastoon pusketaan koodia, niin projekti automaattisesti tekee uuden ajon (*deploy*) eli ajaa uuden koodin ja käynnistää serverin päivitetyllä versiolla (main ja development). Samalla Vercelin automaatio tarkastaa koodikannan virheiltä ja tekee korjausehdotuksia pienten varoitusten kohdalla, sekä peruuttaa koko ajon, jos siinä havaitaan vakavia puutteita.

4.2.2 Tietokanta (PostgresSQL) ja relaatiot

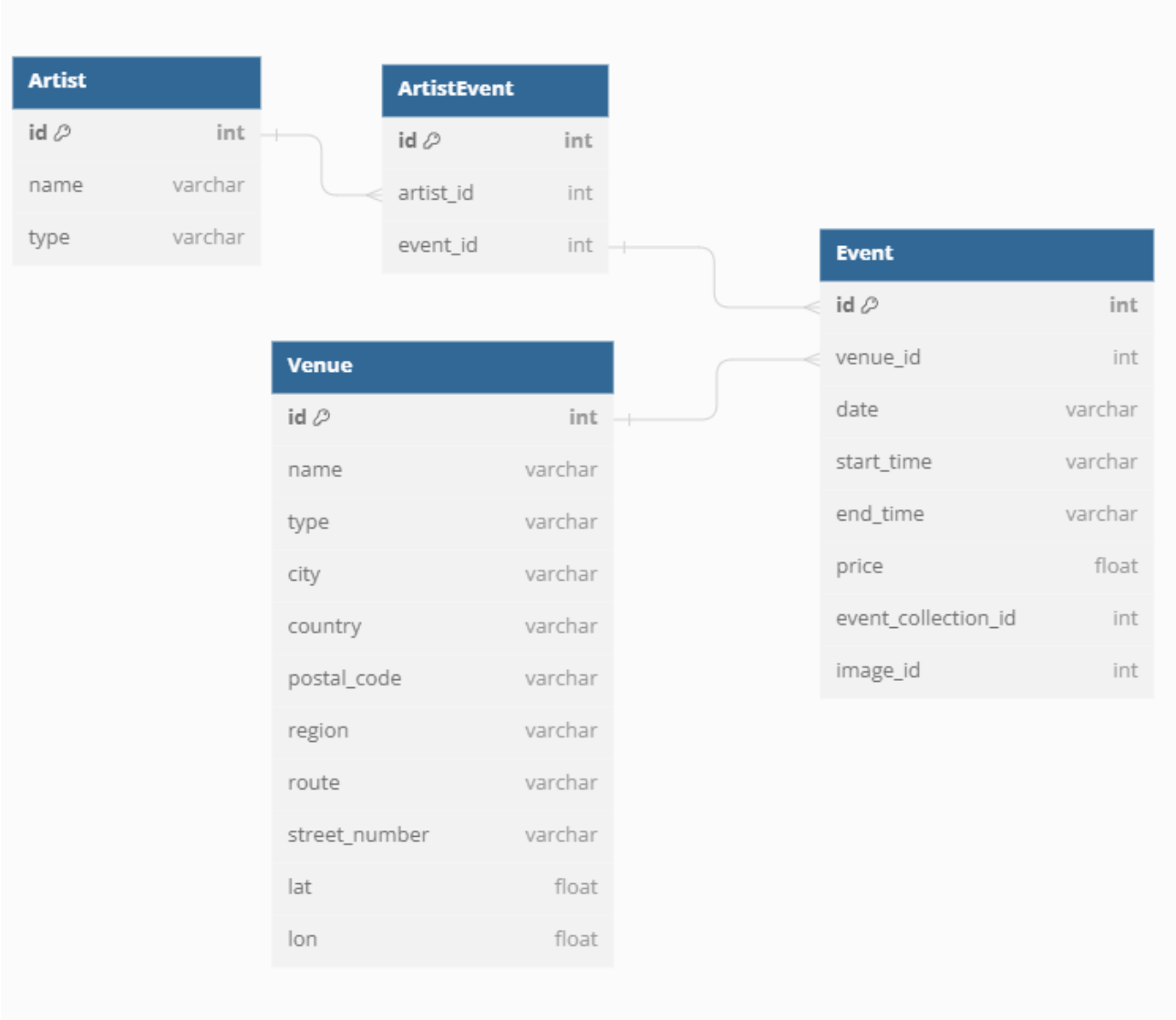
Työn alussa mietittiin tietokannan relaatioita tekemällä relaatiotaulu, joka havainnollistaa miten tieto tallennetaan kantaan ja miten ne ovat liitoksissa toisiinsa. Ensimmäisenä kokeilin testausmielessä käyttäjän (user) ja käyttäjään liittyvien tapahtumajärjestäjien lisäämistä tietokantaan (Kuva 17). Käyttäjällä on perinteisesti nimi sekä sähkö posti ja sillä voi olla useampi tapahtumajärjestäjä (organizer), sillä sama henkilö voi olla esimerkiksi opettajana useammassa koulussa tai kuulua johonkin tanssiryhmään. Näiden välillä on niin sanottu monen suhde moneen, eli yhdellä käyttäjällä voi olla useita järjestäjiä, ja samalla yhdellä järjestäjällä voi olla useita käyttäjiä. Tätä varten luotiin välitaulu (OrganizerUser).

Seuraavana mietittiin mitä tapahtumaan liittyy ja miten näitä kannattaa erotella omiin tauluihinsa. Tapahtumilla on muutamia selkeitä parametreja kuten aika, paikka, esiintyjät sekä hinta. Ajan päätin erotella kolmeen osaan: päivä, alkamisaika, loppumisaika. Esiintyjät ovat liitoksissa monen suhde moneen, joten pitää tehdä erillinen taulu esiintyjille sekä välitaulu tapahtuman välille. Paikka on myös eroteltava omaan tauluun, mutta yhdellä tapahtumalla voi olla vain yksi paikka, joten sen suhde on yhden suhde useampaan.

Kun lähdettiin tarkemmin miettimään miten järjestäjä lisää tapahtumia ja minkälaisia eroja lavatanssien, tanssibileiden ja tanssikurssien välillä, huomattiin muutama ero ja ongelma, jotka johtivat seuraaviin ratkaisuihin.



Kuva 17. Käyttäjän ja tapahtumajärjestäjän väliset relaatiot

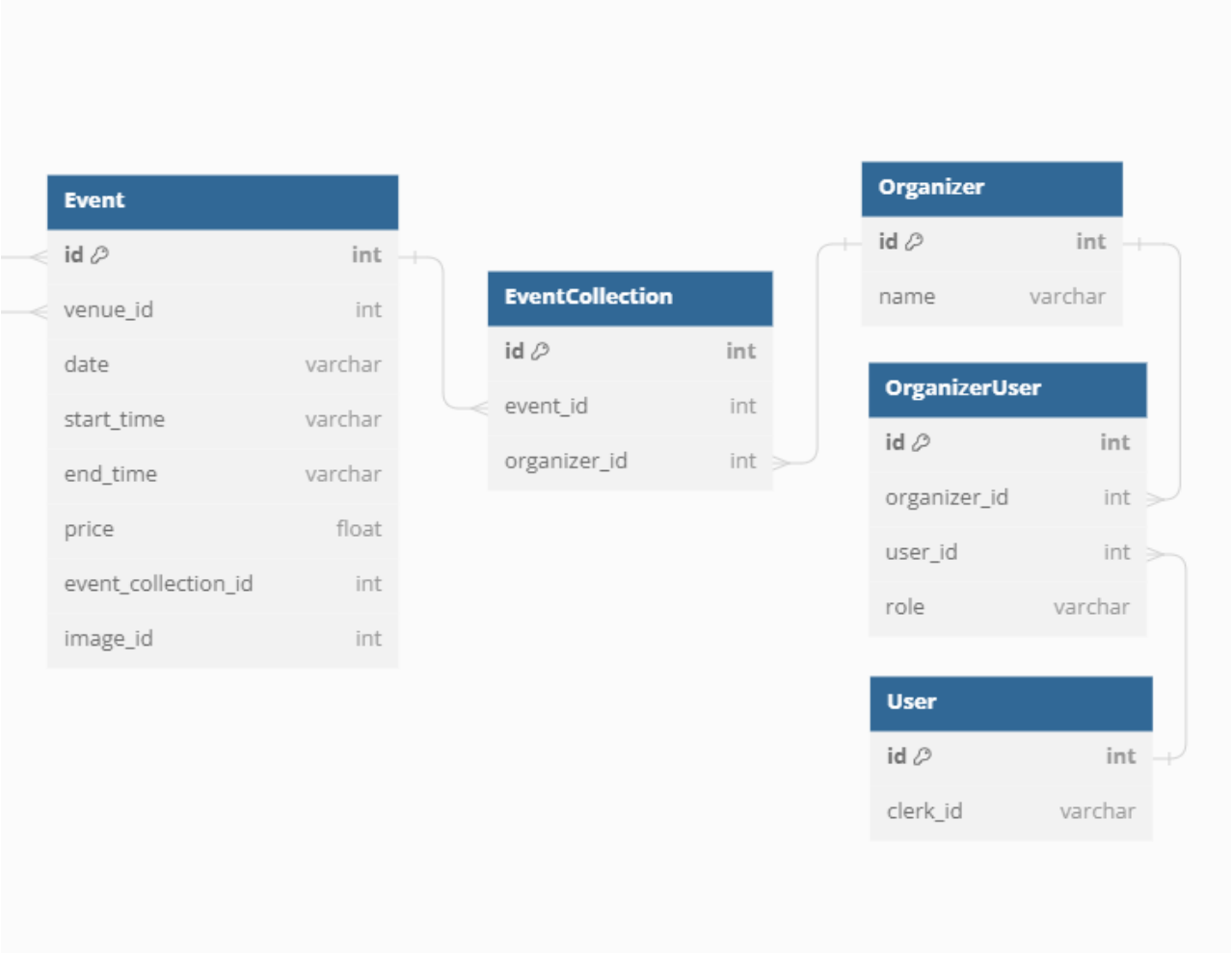


Kuva 18. Tapahtuma ja siihen liittyvät relaatiot

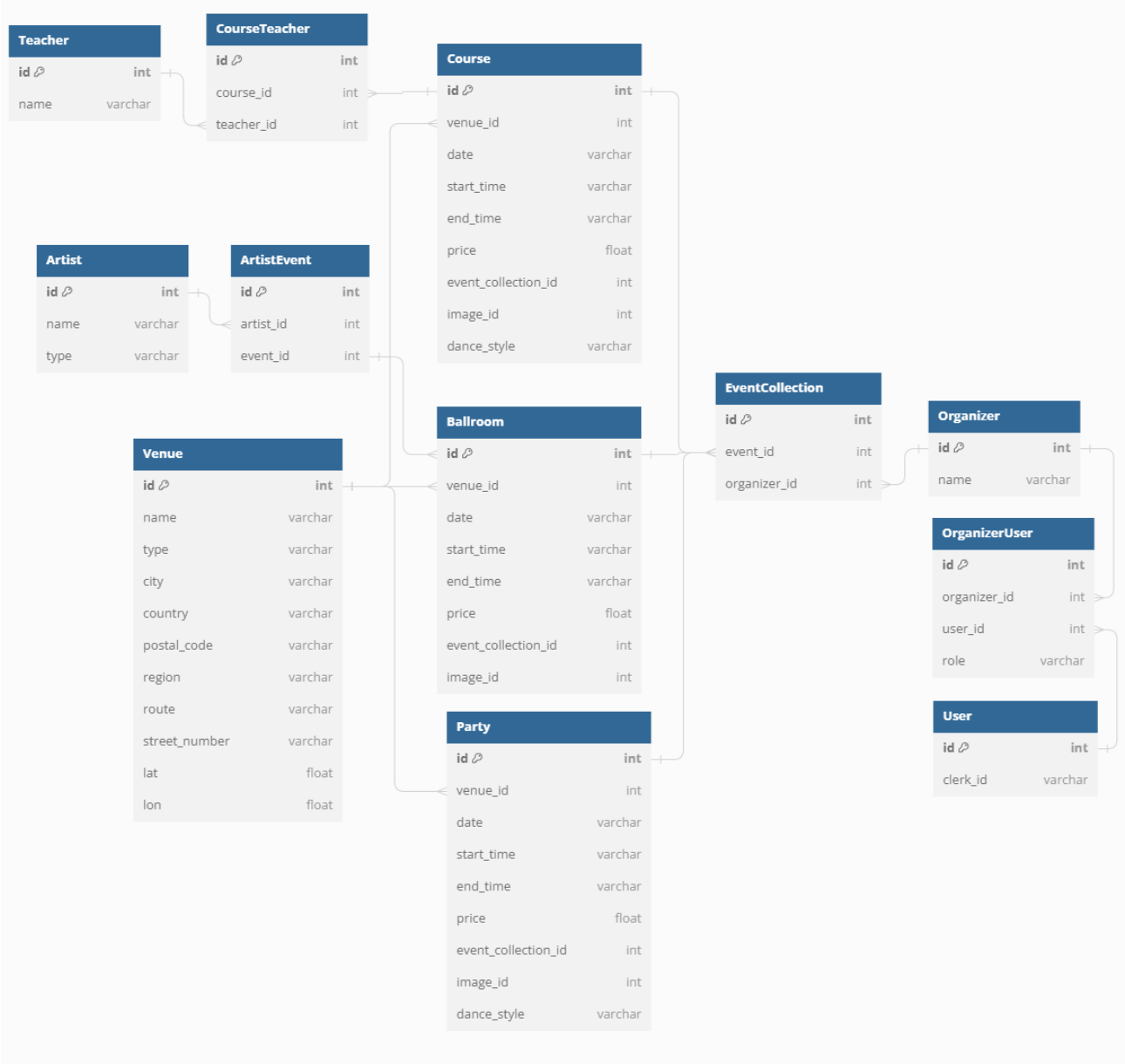
Tapahtumakokoelmat (EventCollection): Järjestäjän kannalta on helpompi jaotella tapahtumat kokoelmiin hallittavuuden helpottamiseksi. Näin voi nimetä esimerkiksi ”kevään 2023” tapahtumat yhden tapahtumakokoelman alle, sillä ne liittyvät toisiinsa. Sen lisäksi voi olla, että samalla järjestäjällä on vaikka intensiivinen viikonlopputapahtuma, jossa on useita tanssikursseja muutaman päivän sisällä, jotka liittyvät toisiinsa, mutta ei mitenkään ”kevään 2023” tapahtumiin. (Kuva 18)

Tapahtumatyypit: Toiseksi koin tärkeäksi erotella nämä kolme tapahtumatyyppiä. Tässä kohtaa jouduin punnitsemaan muutaman vaihtoehdon välillä: Yksi taulu, jossa on kaikkien eri tyyppien parametrit ja joita erotellaan esimerkiksi ”tapahtumatyyppi”-sarakkeella eri kategorioihin vai erotella kaikki kolme kategoriaa eri tauluihinsa. Päädyin lopulta laittamaan nämä omiin tauluihin selkeyttämään rakennetta. (Kuva 20)

Työn aikana olen miettinyt kuitenkin sitä, että lavatanssit ja tanssibileet voisivat olla yhdessä kategoriassa nimeltä ”tanssit”, sillä näillä on hyvin pitkälti samoja ominaisuuksia. Tanssikurssit voisivat olla nimellä ”kurssit”, mikä selkeyttäisi rakennetta siihen suuntaan, että olisi vain kaksi kategoriaa tanssit/kurssit ja näiden sisällä määriteltäisiin niiden tyyppi.



Kuva 19. Tapahtumakokoelmat (*EventCollection*) kokoaa useita tapahtumia nippuun, joita järjestäjän on helpompi hallita



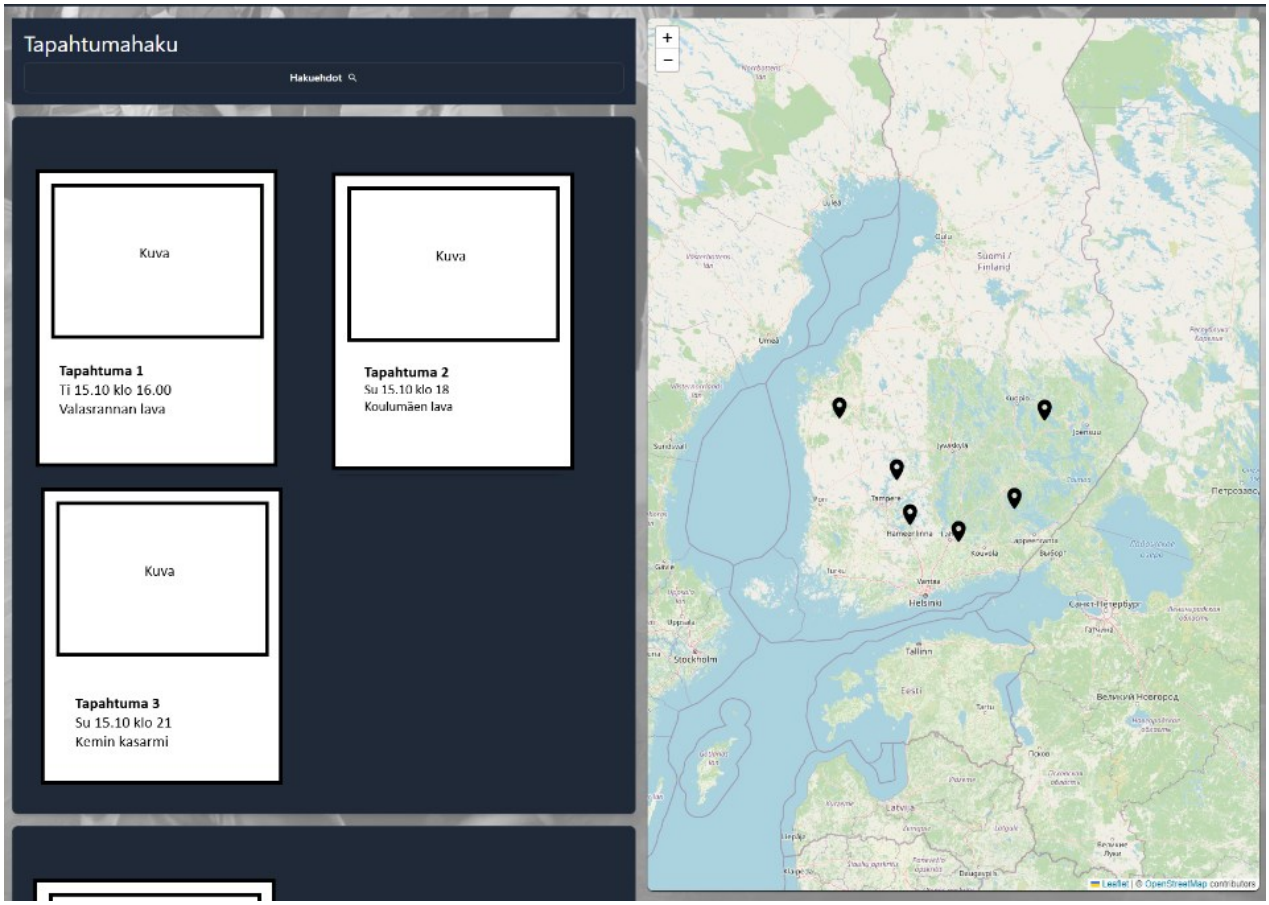
Kuva 20. Lopullinen hahmotelma tietorakenteesta ja tietokannan relaatioista työn alussa

4.2.3 Hakusivun hahmotelmat (*front-end*)

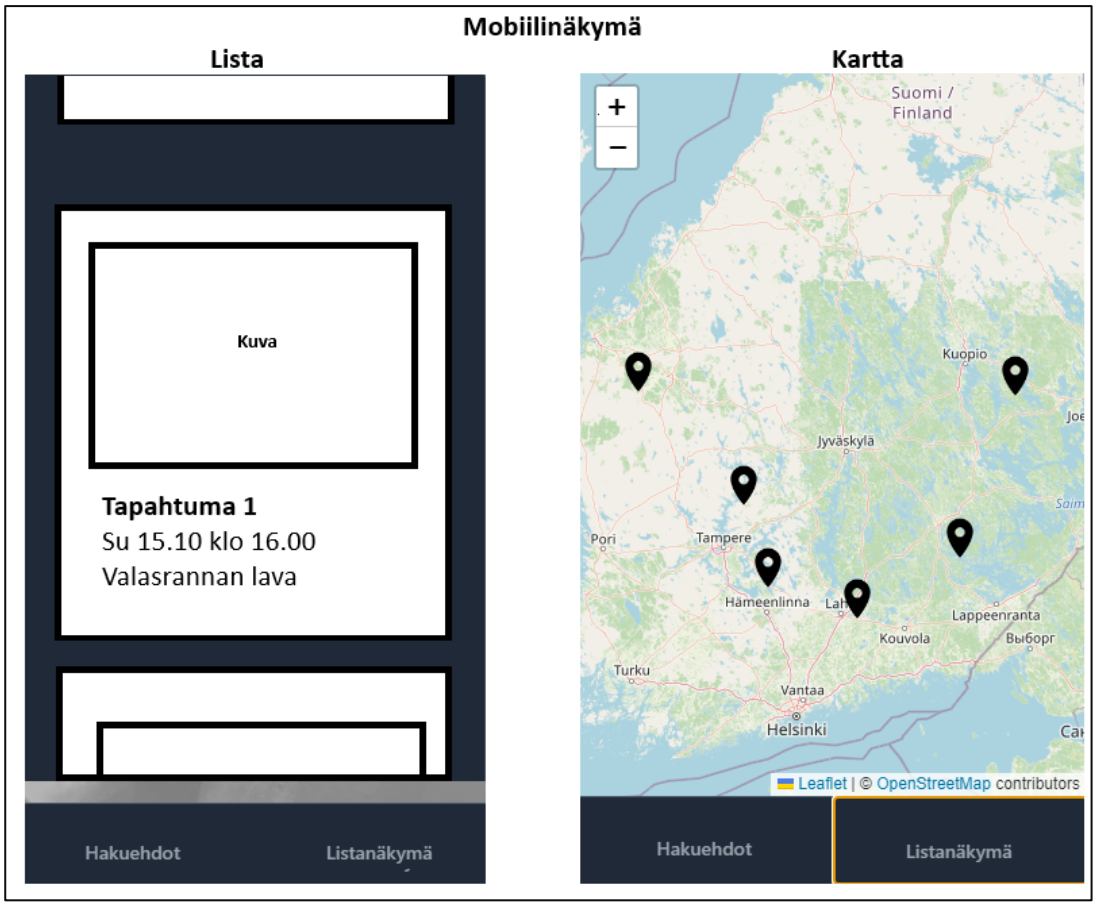
Ensimmäisenä lähdin toteuttamaan hakusivua. Se on applikaation ydinosa peruskäyttäjän näkökulmasta ja tärkeää oli suunnitella mobiili- sekä pöytäkoneversiot. Suunnitelmassa oli kolme selkeää pääkomponenttia: lista, kartta sekä hakuehtojen näyttäminen/asettaminen. Normaalikoossa lista ja kartta näkyvät vierekkäin, ja hakuehdot saa esiin painikkeesta (Kuva 21). Mobiilissa on näkyvissä joko lista, kartta tai hakuehdot, joita ohjataan napeista (Kuva 22). Listalla sekä kartalla tullaan näyttämään tapahtumia, joita painamalla näytetään lisätietoja käyttäjälle. Kartalla näytetään jollain indi-

kaattorilla paikkoja, joiden alle tapahtumat ovat linkitetty ja sitä painamalla saa näkyviin mitä tapahtumia siihen liittyy. Listalla olevat tapahtumat ovat erillisiä kortteja, joista käyttäjä näkee perustiedot sekä kuvan. Lisätiedoissa voi olla enemmän tietoja, joita korttiin ei mahdu laittamaan.

Karttakomponenttina käytin valmista kirjastoa nimeltä Leaflet (Agafonkin, 2023) sekä sen React yhteensopivaa react-leaflet-kirjastoa (React-Leaflet, 2023c), jotka ovat avoimen lähdekoodin JavaScript-kirjastoja, mitkä on suunniteltu interaktiivisten karttojen luomiseen verkkosivuille ja verkkosovelluksille. Se tarjoaa kehittäjille mahdollisuuden lisätä karttoja sivustoilleen ja mukauttaa niitä erilaisiin tarpeisiin. Leaflet-kartat ovat interaktiivisia, ja niihin voi liittää merkintöjä, pop-up-ikkunoita, värikoodeja ja monia muita ominaisuuksia. Kokeilin myös Googlen karttapalvelua, mutta päädyin pysymään Leaflet-kirjastossa, koska se on ilmainen ja siitä löytyi valmis toiminnallisuus, jota Googlen kirjastosta ei helposti saanut tehtyä.



Kuva 21. Hakusivun hahmotelma normaalikoossa. Hakunäkymässä lista ja kartta ovat vierekkäin yhtä aikaa näkyvissä.



Kuva 22. Hakusivun hahmotelma mobiilikoossa (vas. Listanäkymä, oik. Karttanäkymä). Vain joko lista tai kartta on kerrallaan näkyvissä. Näkymää vaihdetaan alavalikon nappuloista.

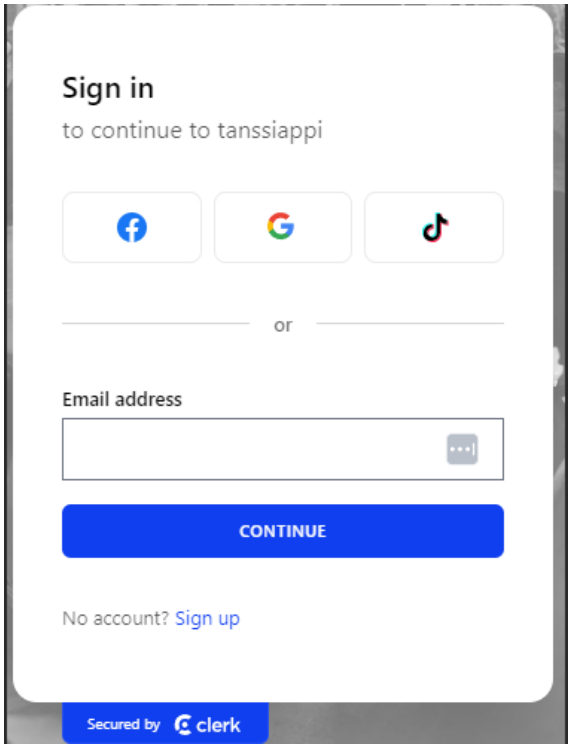
4.2.4 Kirjautuminen

Kirjautuminen on olennainen osa applikaation toimintaa ja se mahdollistaa tapahtumajärjestäjän tietojen hallinta eli tapahtumien lisäämistä, muokkaamista ja poistamista. Tässä asiassa halusin päästä helpolla, enkä halunnut ottaa riskiä salasanojen hallinnoinnista. Tästä syystä lähdin hakemaan mahdollisuuksia kolmannen osapuolen kirjautumispalveluista. Lähdin alussa toteuttamaan kirjautumista Vercelin dokumentaation ehdottamalla NextAuth.JS -kirjautumisviitekehyksellä (NextAuth.JS, 2023). Halusin antaa käyttäjälle kirjautumisvaihtoehdoiksi yleisimmät sosiaalisen median tarjoamat kirjautumiset kuten Facebook sekä Google, mutta näiden lisäksi myös salasanattoman sähköpostikirjautumisen. Kaikki nämä on mahdollista toteuttaa NextAuth.JS:n avulla, mutta törmäsin toteutuksen kanssa ongelmiin enkä saanut muuta kuin Facebookin toimimaan. Tästä syystä lähdin etsimään vaihtoehtoja ja löysin toisen palveluntarjoajan nimeltä Clerk (Clerk, 2023).

Clerk on hieman enemmän kuin pelkästään kirjautumispalvelu. Se tarjoaa myös käyttäjän hallintamoduulin, joka antaa erillisen käyttöliittymän oman tilin hallintaan (Kuva 23). Sen avulla käyttäjä voi helposti muokata tietojaan sekä poistaa oma tilinsä, ja kaikki tämä tapahtuu tietoturvallisesti. Clerk

on helposti asennettavissa ja sen tyylejä voi muokata tarvittaessa sopimaan oman applikaation tyyliin.

Clerkin API antaa kehittäjälle erilaisia työkaluja kirjautumistietojen käyttämiseen ja yhdessä nextj.js:n middlewaren kanssa (johon Clerk on integroitu), voidaan tietyt sivut estää käyttäjältä, jos se ei ole kirjautunut palveluun. Esimerkiksi tietojen hallinta -sivulle ei pääse, jos ei kirjautumista ole havaittu ja tällöin käyttäjä ohjataan etusivulle. Vastaavasti joitakin API reittejä on blokattu tai sallittu riippuen kirjautumisesta.



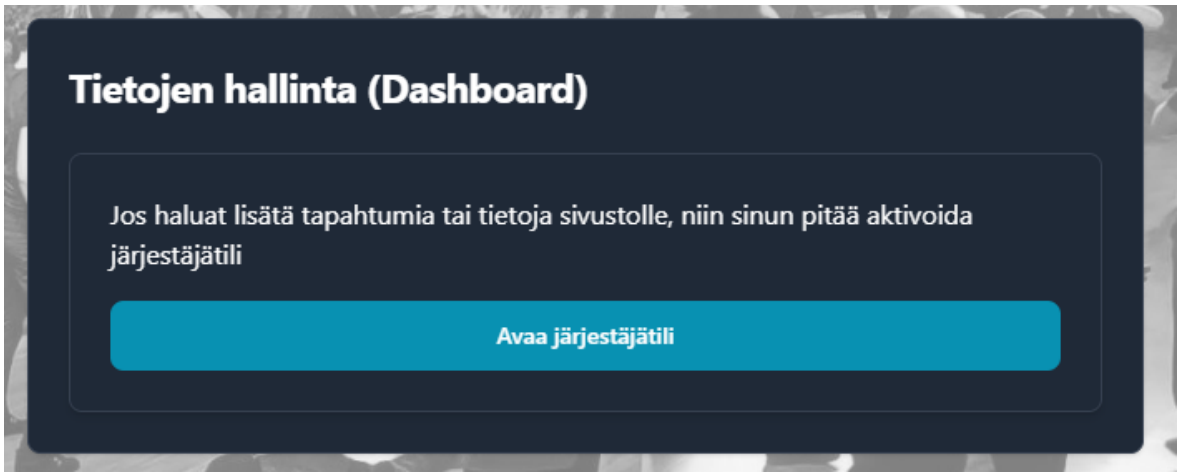
Kuva 23. Esimerkki Clerkin kirjautumiskäyttöliittymästä. Vastaavanlainen on myös rekisteröinnille

4.2.5 Tietojen hallinta (*dashboard*)

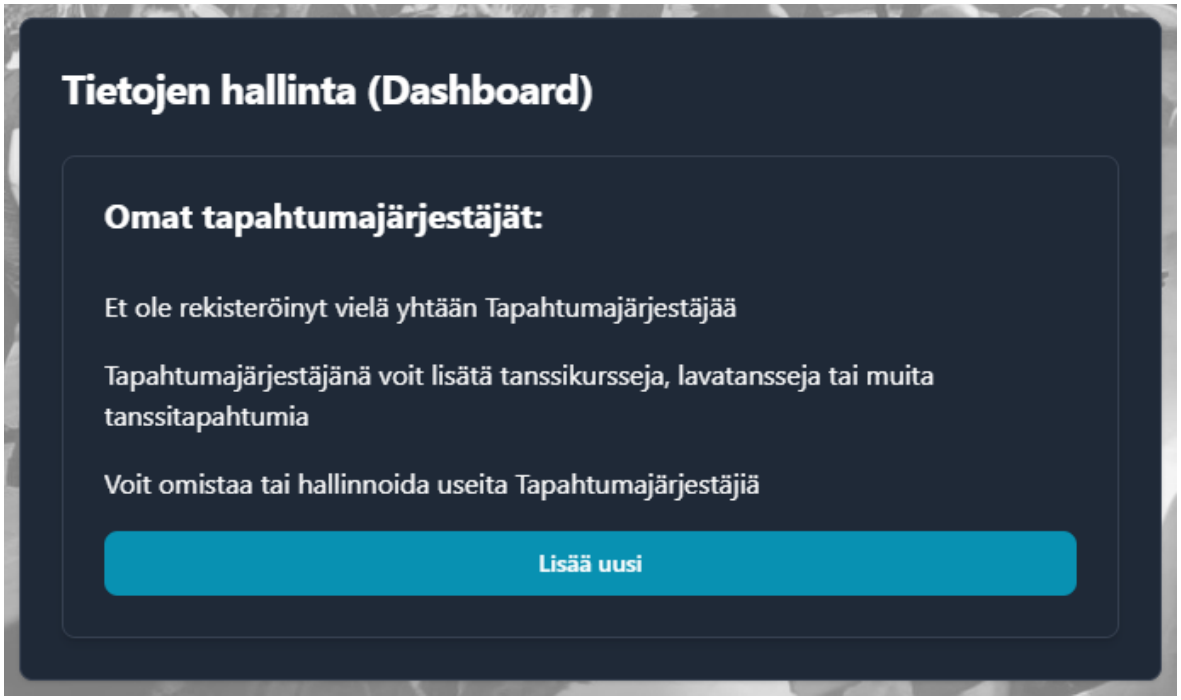
Tietojen hallinta on applikaation yksi olennaisin osa ainakin tässä kohtaa sen elinkaarta. Lähtökohtaisesti tarkoituksena oli tämän työn osalta kehittää palvelu, jossa tapahtumanjärjestäjällä on mahdollisuus syöttää, muokata sekä poistaa tietoja helposti. Keskityin tässä kohtaa vain tapahtumien hallintaan, enkä vielä esimerkiksi järjestäjän organisaatiohallintaan kuten käyttäjien määritykseen.

Kun käyttäjä rekisteröityy palveluun ja kirjautuu sisään, avautuu mahdollisuus mennä tietojen hallintasivulle. Hallintasivulla voi aktivoida järjestäjätilin, jonka jälkeen voi lisätä itselleen uusia tapah-

tumajärjestäjiä. Järjestäjä voi olla esimerkiksi tanssikoulu, tanssiyhteisö tai yksittäinen opettaja. Tämän jälkeen käyttäjä voi lisätä järjestäjänä tapahtumia. Tapahtumat ovat lisäksi yhdistetty kokoelmiksi, jotta niitä on helpompi hallita sekä näyttää loppukäyttäjälle hakusivulla.



Kuva 24. Järjestäjätilin aktivointi. Aktivointi tekee api-pyynnön, joka yhdistää clerk-tunnuksen tanssiapin käyttäjätauluun muodostaen uuden järjestäjätili.



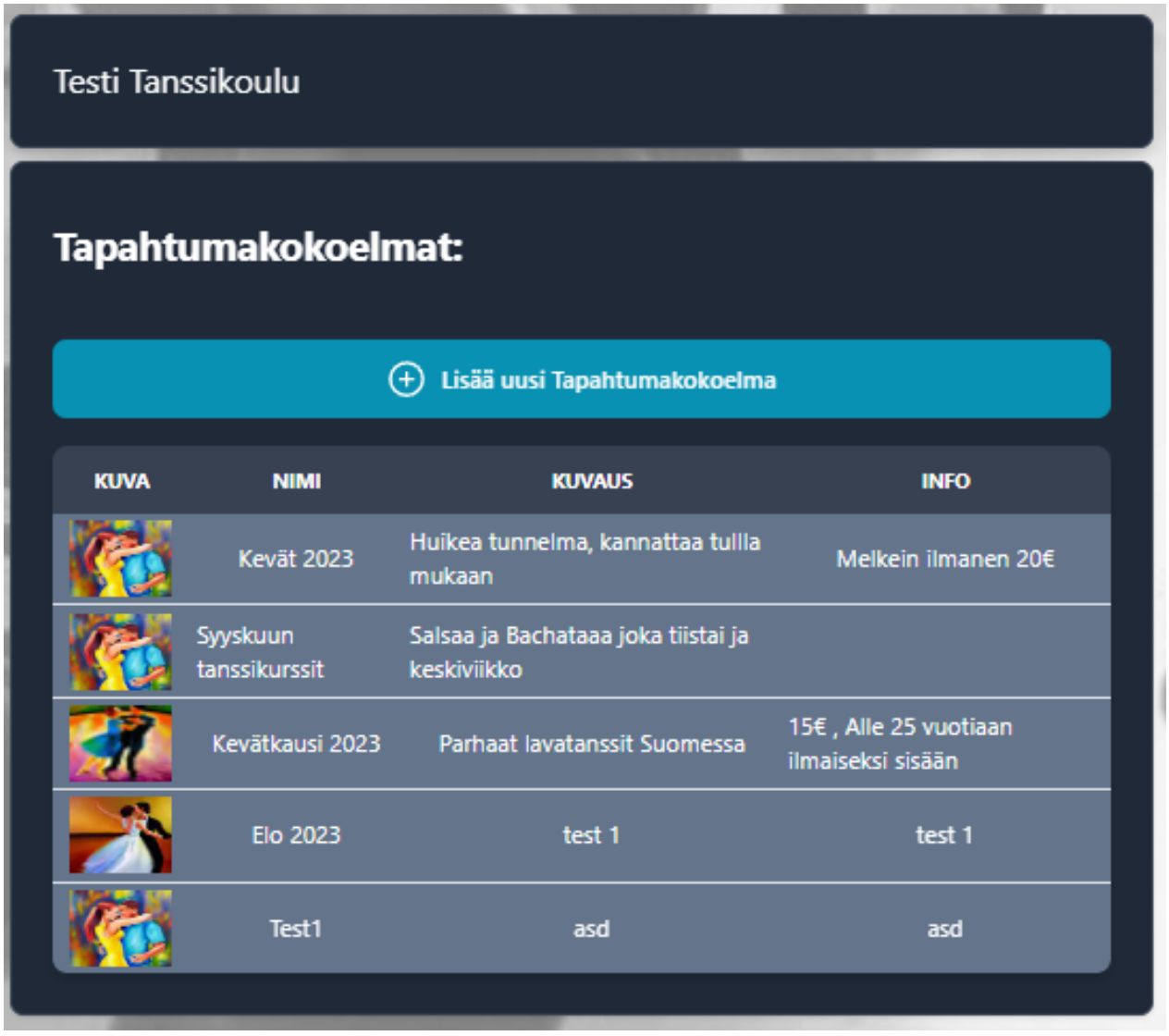
Kuva 25. Tapahtumajärjestäjien lisääminen

Tapahtumakokoelman on tarkoitus olla yksittäisen tapahtumien ylätaso, johon voi lisätä tietoja, jotka ovat yhteisiä alatason yksittäisille tapahtumille. Otetaan esimerkkinä eräänä lauantaina järjestettävä workshop, johon kuuluu 3 tuntia opetusta ja tanssibileet. Kaikki 3 tuntia ja bileet kuuluvat

samaan ylätason ”tapahtumaan”, mitä kutsun siis tapahtumakokoelmaksi. Olisi täysin turhaa asettaa samoja tietoja, kuten tapahtuman kuvaus, jokaiselle alatason ”tapahtumalle” eli tanssitunneille sekä bileille erikseen.

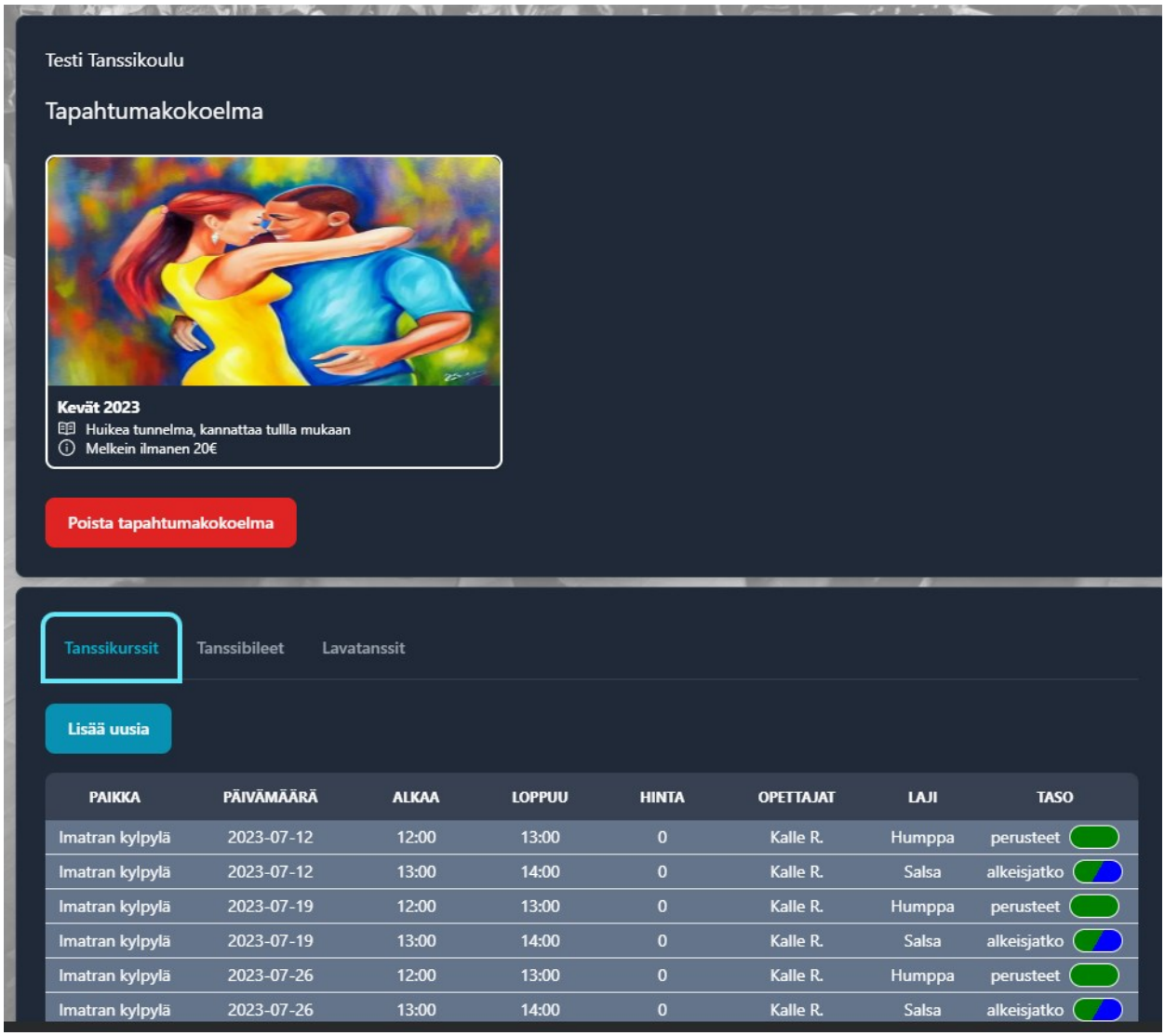
Toiseksi, kun järjestäjä lisää tietoja järjestelmään, kerääntyy ajan myötä kymmeniä tai jopa satoja pieniä alatason tanssitapahtumia, joita voisi olla hankala hallita erikseen, jos ne olisivat vain listattuna omille riveilleen. Kokoelmien avulla nähdään helposti tapahtumat ryhmiteltynä.

Kolmanneksi syyksi nostan esille vielä sen, että kun loppukäyttäjä etsii tapahtumia sivustolla, on hyvin mahdollista, että hakutuloksia on iso määrä, ja niiden lukeminen olisi hankalaa isosta listasta alatason tapahtumia. Kokoelmien avulla käyttäjälle näytetään tapahtumat tehokkaammin ja selkeämmin.

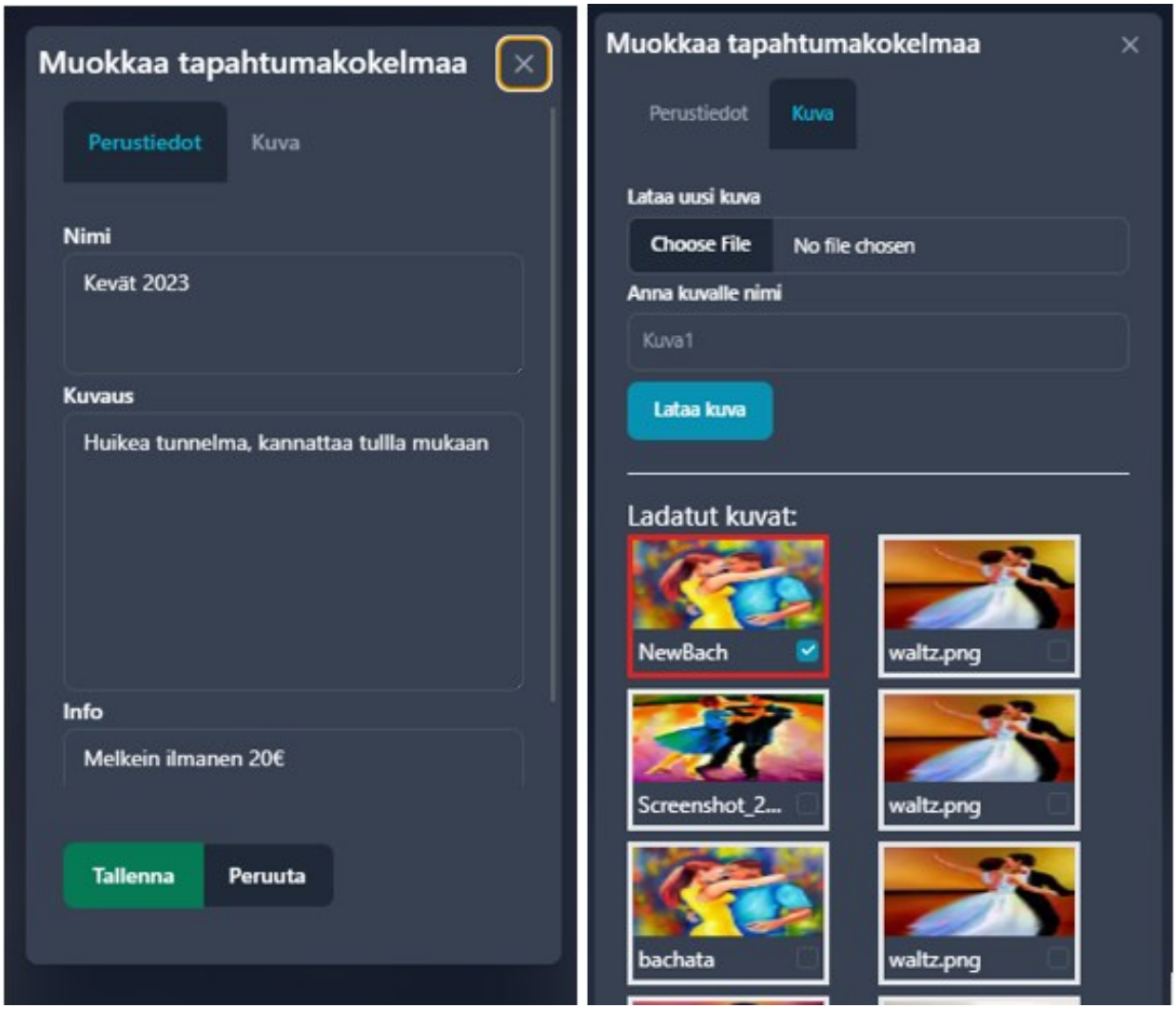


Kuva 26. Tapahtumajärjestäjän näkymä, mistä näkee tapahtumakokoelmat listattuna

Työn aikana tätä ideologiaa on työstetty useamman kerran, enkä usko tämän olevan viimeinen ratkaisu. Olen ajatellut sellaista, että tapahtumakokoelmat jaettaisiin vielä ylätason tapahtumakategorioihin kuten ”workshop”, ”tanssileiri”, ”lukukausi”, ”avoin”, ”lavatanssit” tai muita vastaavia, joiden näkymät ja lomakkeet olisivat hieman kustomoidumpia näille kategorioille tyypillisellä tavalla.

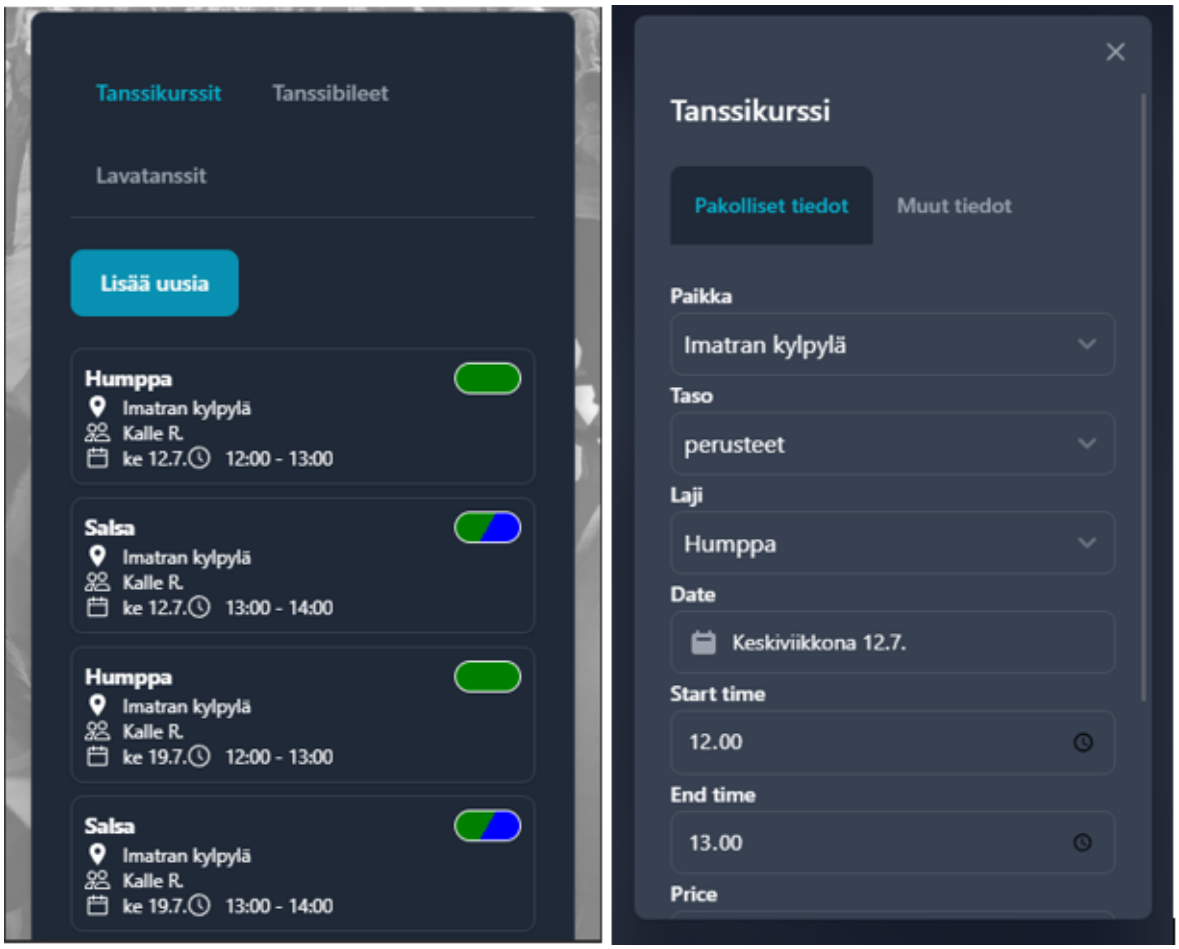


Kuva 27. Tapahtumakokoelman hallintasivu. Yläosassa kokoelman tiedot ja alhaalla alatason tapahtumat.



Kuva 28. Yläosan kokoelmaosion tietojen muokkaus avaan Modal-komponentin, jossa on lomake tietojen täyttöön sekä kuvan asettamiselle ja lisäämiselle.

Työn aika kävi selväksi myös se, että hallintasivua pitää miettiä myös mobiilikoossa. Keskustelut Antti Törmäsen kanssa olivat käytännössä aina puhelimitse, ja hän tarkasteli applikaation kehitystä niemenomaan puhelimelta. Tätä varten työstin erillisen näkymän alatason tapahtumille, jotta pää-tiedot saa kompaktisti näkymään korttikomponentin sisällä, jotka ovat listattu allekkain. Korttia pai-namalla avautuu tämän kaikki tiedot lomakkeella, jota voi muokata.

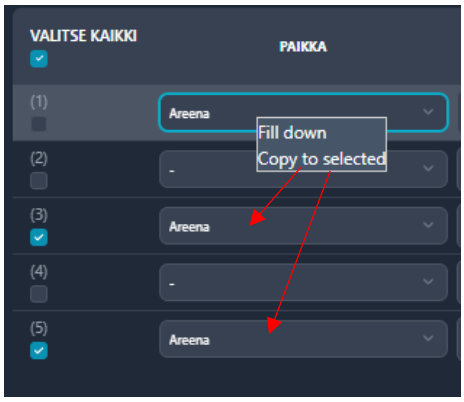


Kuva 29. Tanssikurssien listaus kokoelman alaosassa mobiilinäkymässä. Oikealla tiedot avattuna modal-komponenttiin lomakkeella.

Uusien tapahtumien lisäämiseen tein erilaisia nappuloita, jotka helpottavat useiden tapahtumien yhtäaikaista lisäämistä. Käyttäjä voi siis kerralla lisätä niin monta tapahtumaa, kuin haluaa ja tätä varten tarvitaan uusien rivien (tapahtumien) lisäämispainike ”Lisää uusi”, mikä lisää siis uuden tyhjän rivin. Tämän lisäksi käyttäjä voi valita jo täytetyn tapahtuman (yhden tai useamman) ja tehdä tästä kopion painikkeella ”Kopio valinnat”, mikä kopio valitut täytetyt tapahtumat uusille riveilleen vastaavilla tiedoilla. Kolmas kopiointinappi on ”Kopio jatkuvana”, jolla käyttäjä voi tehdä kopiota tapahtumasta yhden viikon intervallilla tiettyyn päivämäärään asti. Tämä on erityisen kätevää, kun lisätään viikkotunteja, jotka jatkuvat esimerkiksi koko kevään ajan. Neljäntenä nappulana on ”Poista valitut”, mikä nimensä mukaisesti poistaa valitut rivit kokonaan. Viimeinen nappula on ”Tallenna tiedot”, joka tietysti tallentaa kaikki rivit/tapahtumat tietokantaan. Tallennusvaiheessa tarkastetaan, että pakolliset tiedot on täytetty jokaisessa rivissä.

Kuva 30. Esimerkki valitun rivin kopioinnista. Valitaan vasemmalta valintaruudusta ensimmäinen tapahtuma ja sen jälkeen painetaan kopiointinapista. Valittu rivi kopioituu uudelle riville sisältäen kaikki tiedot valitusta rivistä.

Näiden nappuloiden lisäksi tein oman hiiren oikealla painikkeella aukeavan valikon, josta voi kopioida yksittäisen sarakkeen tiedot alapuolelle oleviin riveihin ”fill down” sekä valittuihin riveihin ”copy to selected”.



Kuva 31. Oikean hiirenpainikkeen toiminnot. Esimerkki valittuihin riveihin kopioimisesta

Tietojen hallintaan tarkoitettuja toimintoja tullaan varmasti lisäämään myöhemmin, mutta nämä olivat olennaisimmat toiminnallisuudet, mitä koin tarpeelliseksi. Yksi esiin tullut ehdotus oli excel-tiedostoihin liittyvät import/export toiminnot. Kokeilin tästä yhtä versiota alussa, mutta jätin tämän

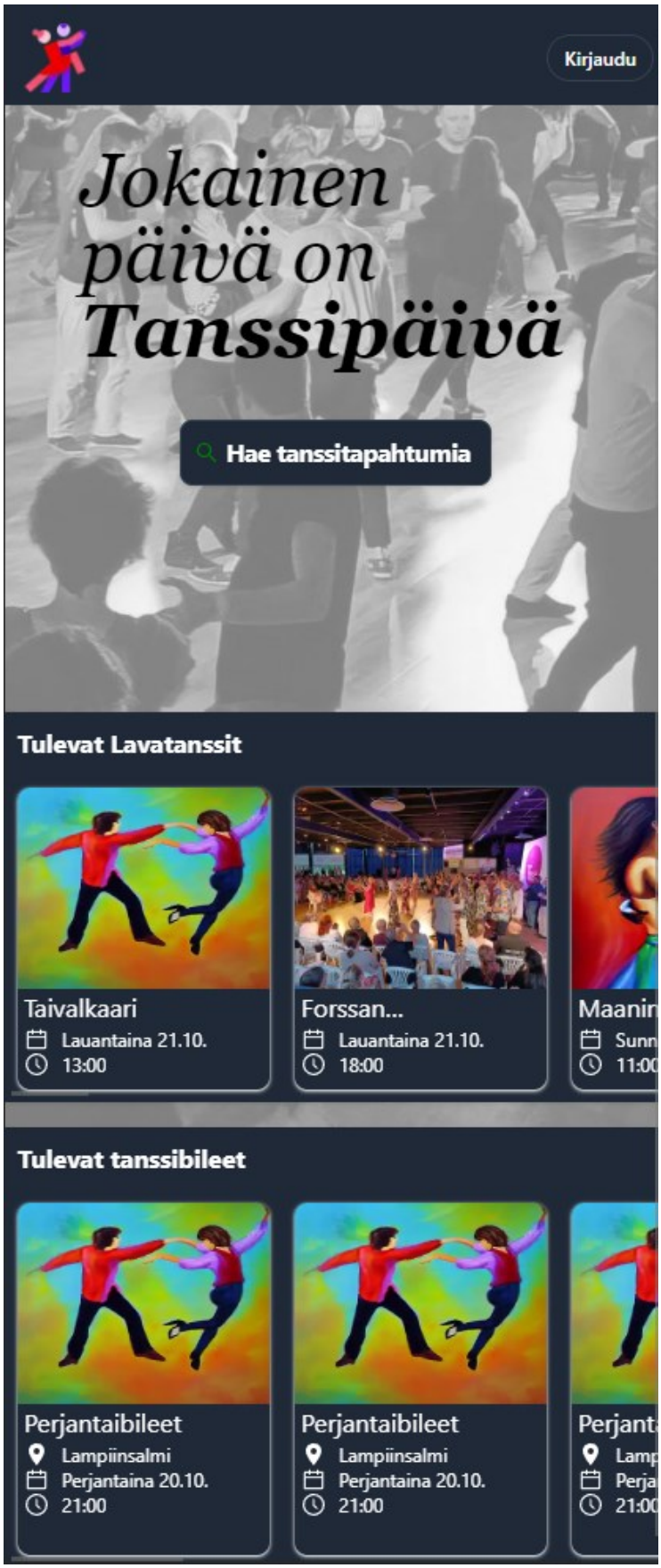
osion pois, sillä paikat/opettaja/lajit/tasot ynnä muut omassa tietokannassa olevat referenssit ovat hankalia toteuttaa ainakin import-toimintoon. Omasta mielestäni haluttavampi toimintatapa olisi se, että excelin käyttöä ei tarvitse, vaan tämä käyttöliittymä olisi niin helppokäyttöinen, että siitä tulisi ensisijanen tietojen hallintatapa. Tällöin järjestelmästä voisi kyllä exportoida excelissä tiedot, jos niitä tarvitsee lähettää eteenpäin.

4.2.6 Etusivu

Etusivua lähdettiin miettimään vasta työn loppuvaiheilla, kun muut toiminnallisuudet olivat jo pääsääntöisesti valmiina. Osittain tämä johtui myös siitä, että vastaani tuli muutamia teknisiä ongelmia, johon en halunnut jäädä jumiin. Nämä ongelmat korostuivat nimenomaan etusivulla, joten erillisen hakusivun tekeminen kiersi tämän ongelman. Etusivun tärkeys tuli esille ja korostui vasta siinä vaiheessa, kun lähdin pyytämään mielipiteitä kavereiltani. Koska heille applikaatio ei ollut vielä tuttu, olivat he usein hukassa jo alusta asti. Tässä kohtaa etusivulla oli ollut vain linkki hakusivulle, jonka jälkeen tietoja piti vielä erikseen etsiä hakuehtojen perusteella.

Sain testaajilta palautteeksi muun muassa seuraavaa: ”Haluaisin nähdä etusivulla heti tulevia tapahtumia” sekä ”Voisiko hakupainike olla vapaakenttähaku ja sijaita yläosan navigaatiopalkissa koko ajan näkyvillä?”. Jälkimmäistä muutosta en halunnut toteuttaa tässä kohtaa, sillä minulla ei ole osaamista vielä tehdä vapaamuotoista dynaamista hakua, enkä kokenut sitä niin tärkeäksi toteuttaa. Näistä lähdin toteuttamaan ensimmäistä muutosta, sillä se vaatisi vain automaattisen haun tekemisen, kun sivu latautuu. Tätä toteuttaessa löysin uusia ongelmia, mutta myös ratkaisuja uusiin sekä vanhoihin ongelmiin, jotka loppujen lopulta johtuivat Clark-integroinnin latautumisesta eri aikaan kuin sivusto. Koska Clark on yhteydessä middlewareen, tuli tässä kohtaa ristiriita, kun sivusto yritti tehdä komentoja, jotka käyttivät tätä. Tämä oli todennäköisin syy ongelmiin, joka ratkesi sillä, että kun applikaatio latautuu, niin heti alussa tehdään varmistus siitä, että Clerk latautuu ennen kuin sivuston muita toimintoja aletaan tekemään.

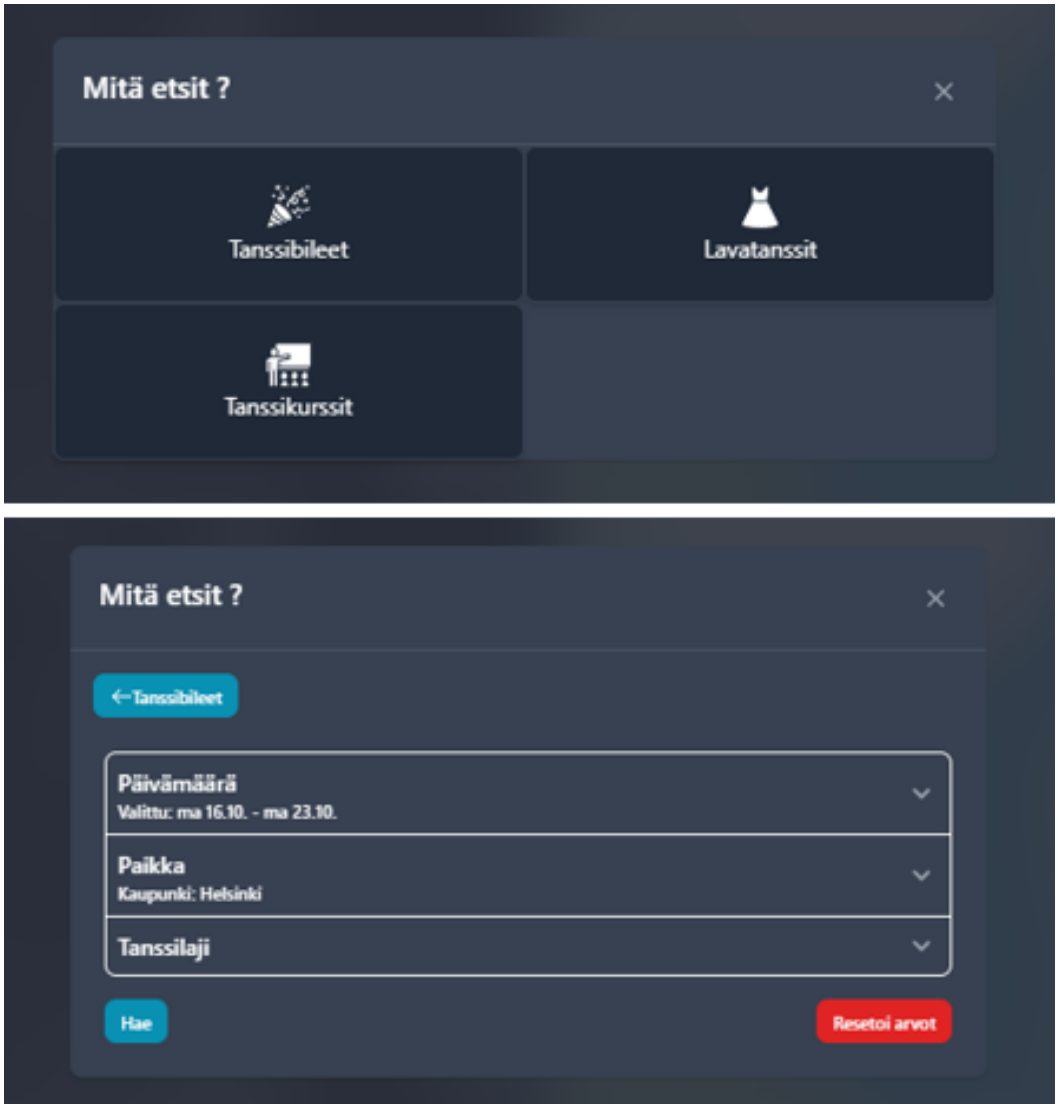
Kun etusivu oli saatu Kuva 32 mukaiseksi, olin todella tyytyväinen saamastani palautteesta, sillä etusivun tekeminen antoi paljon ryhtiä koko applikaatiolla, vaikka alun kehitysvaiheessa sillä ei vielä ollut niin paljon merkitystä.



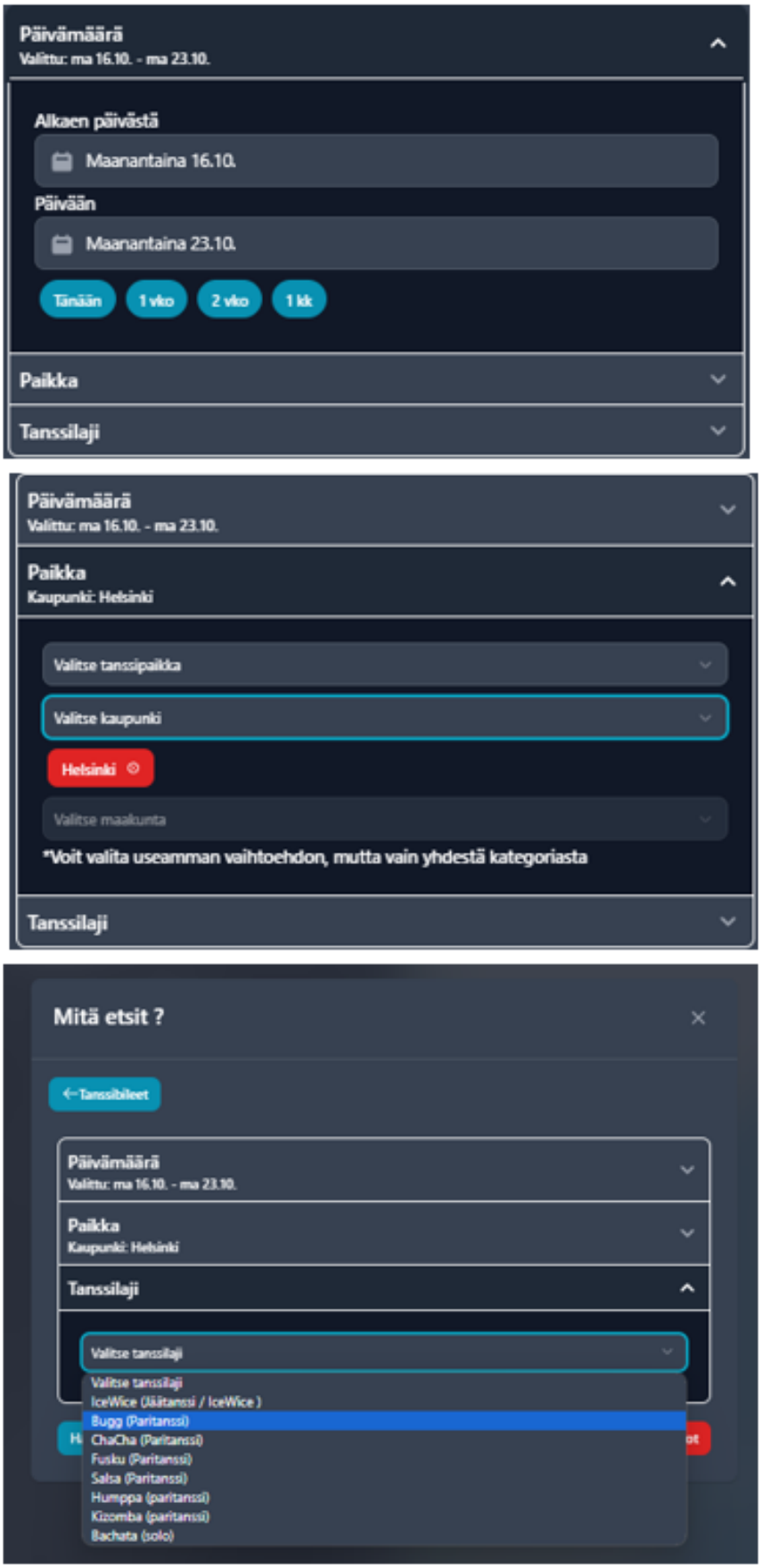
Kuva 32. Etusivun näkymä mobiilikoossa. Sivu näyttää automaattisesti tulevia tapahtumia, ja halutessaan käyttäjä voi mennä "Hae tanssitapahtumia" -painikkeesta hakusivulle

4.2.7 Hakusivu, tietojen haku ja filtterit

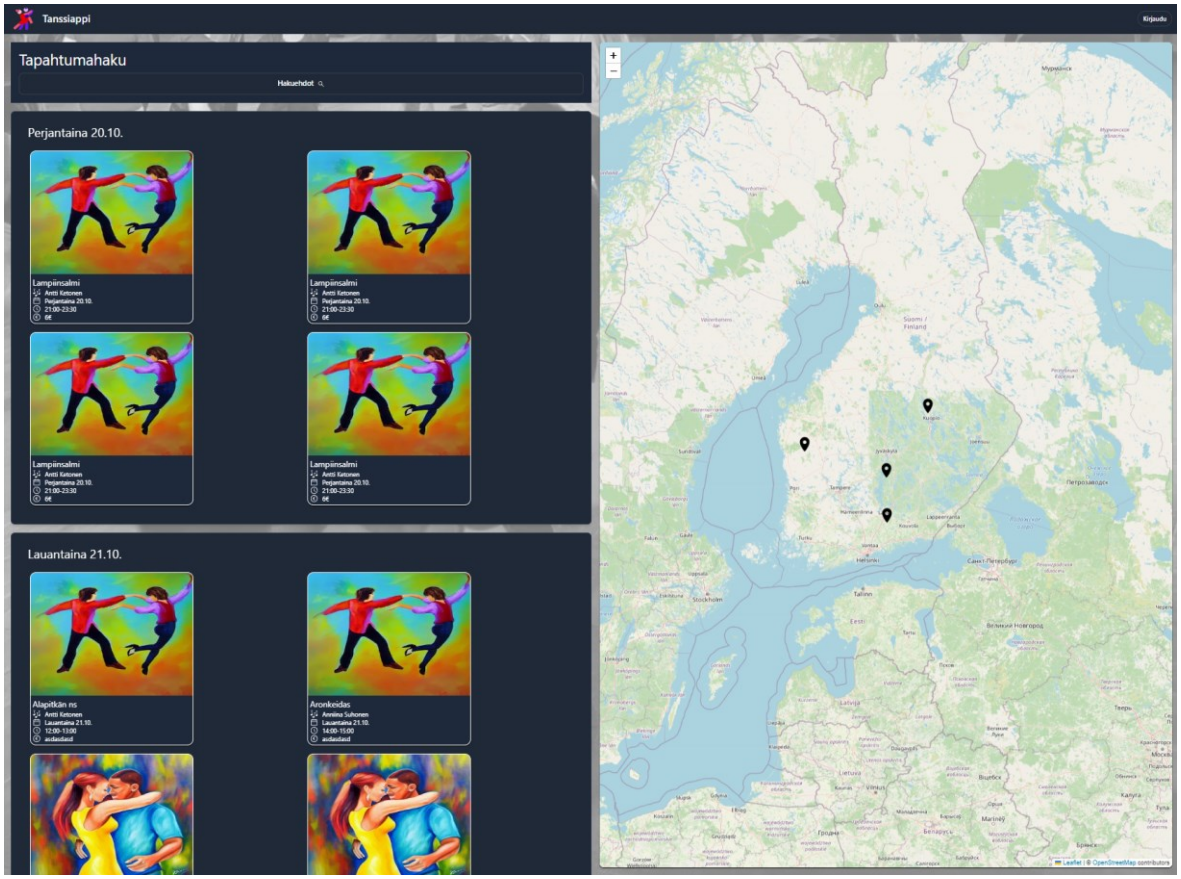
Loppukäyttäjän kannalta olennainen osa on tietojen haku ja filtterien asettaminen. Käsittelin aikaisemmin hakusivun hahmotelmaa ja lopullinen versio lähellä alussa tehtyä suunnitelmaa. Hakuehtojen asettaminen on ensimmäinen asia mitä hakusivulla pääsee tekemään (Kuva 33 ja Kuva 34). Kun hakuehdot on asetettu ja käyttäjä painaa ”hae”-nappia, niin tällöin applikaatio lähettää kutsun ensin next.js api reittiin ja sieltä tehdään kysely tietokantaan käyttäen annettuja filttereitä. Kun tulokset saapuvat, niin lista ja kartta päivittyy tapahtumista (Kuva 35).



Kuva 33. Hakuehtojen asettaminen. Käyttäjä voi hakea tietoa eri kategorioilla ja asettaa tarkempia hakuehtoja.



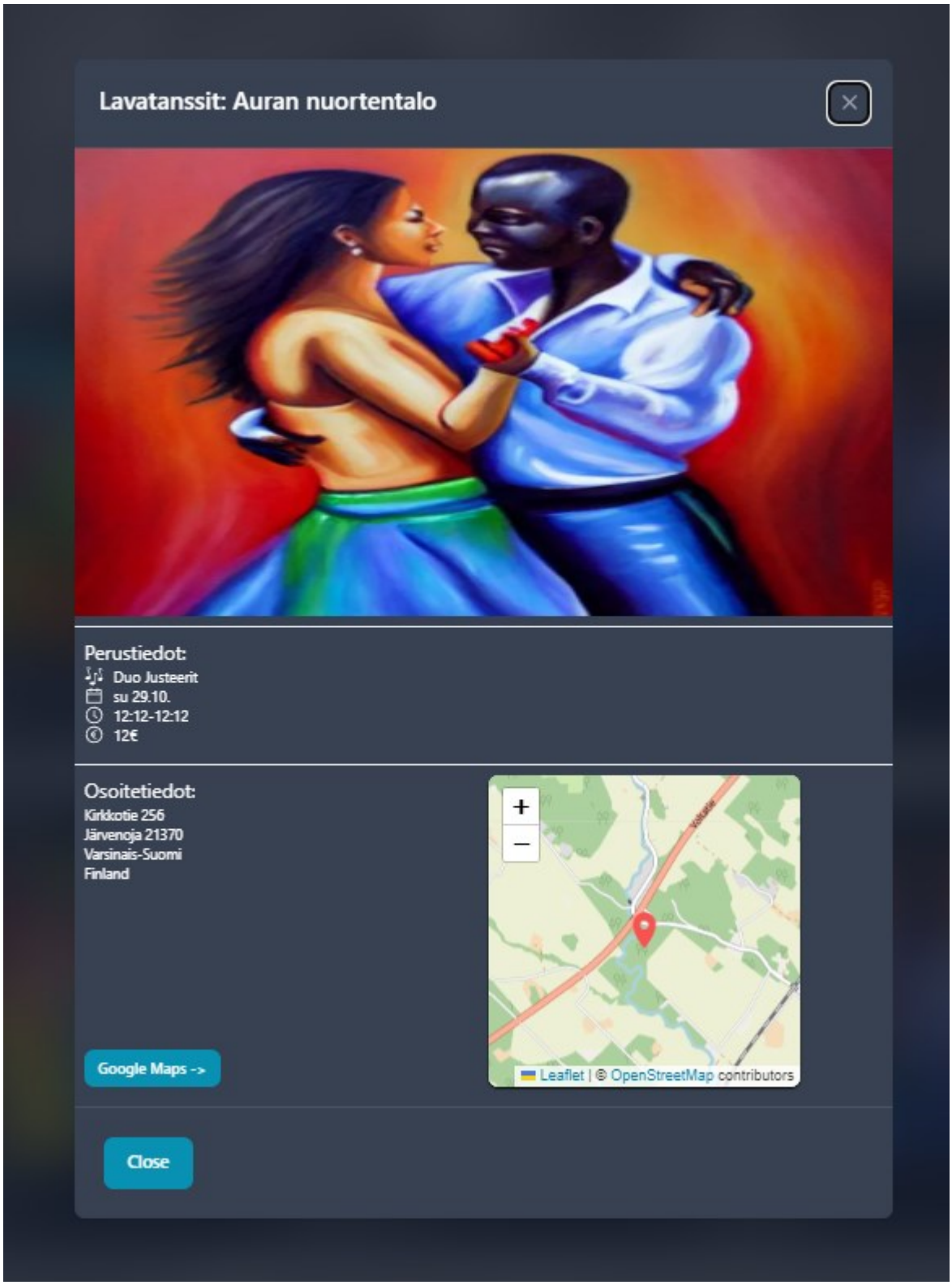
Kuva 34. Hakuehtojen tarkempien ehtojen valintakomponentit



Kuva 35. Haun tulokset esitettynä hakusivulla

Kun käyttäjä painaa tapahtumaa listasivulla, niin avautuu modal-komponentti, joka näyttää tapahtuman tarkemmat tiedot (Kuva 36). Toistaiseksi tapahtumilla ei ole omia sivuja eli yksittäistä tapahtumaa ei voi esimerkiksi etsiä url-osoitteen avulla. Myöhemmin tätä on tarkoitus muuttaa siten, että jokainen tapahtuma voidaan avata yksittäisenä sivuna. Next.js:n edistyneemmät reititykset (interceptin routes) mahdollistavat myös sellaisen tavan, että kun modal avautuu, niin sen sisällä oleva sisältö onkin jostain toisesta url-osoitteesta, vaikka taustalla oleva sivu on edelleen vanhassa url-osoitteessa. Tämä voisi olla tulevaisuudessa hyvä tapa näyttää tapahtumatietoja.

Lisäksi muita jatkokehityskohteita on miettiä mitä kaikkia tietoja tapahtumat saavat ja miten esimerkiksi järjestän, artistin, lajin tai muiden tietokannassa olevien objektien tietoja näytetään. Käyttäjä voisi esimerkiksi painaa järjestäjästä, jolloin avautuisi tietoja tästä tai vastaavasti avata uuden modal-komponentin. Koska koko tietokannan tarkoitus on linkittää eri objekteja toisiinsa, pystyy nopeasti saamaan todella monipuolisen ja interaktiivisen käyttöliittymän, mistä tiedonsaanti on helppoa ja nopeaa.



Kuva 36. Tapahtuman tiedot esitettynä modal-komponentissa. Osoitetiedoista näkee tärkeimmät tiedot, kartan sekä nappia painamalla pääsee suoraan google-karttoihin paikan osoittamaan koordinaattiin.

5 Lopputulokset ja päätelmät

Tämän työn tuloksena syntyi selainpohjainen mobiiliystävällinen web-applikaation MVP-versio (Wikipedia, 2023b), jonka keskeisimmät toiminnat ovat tanssitapahtumatietojen syöttäminen ja muokkaaminen sekä tapahtumien hakeminen ja tietojen näyttäminen loppukäyttäjälle listamuodossa sekä kartalla. Tietojen hakemiseen ei tarvita tiliä tai muuta kirjautumista, mutta tapahtumien lisääminen vaatii kirjautumisen, järjestäjäprofiilin aktivoinnin. Kirjautumispalvelu sekä käyttäjän henkilötietojen hallinta ulkoistettiin kolmannelle osapuolelle (Clerk).

Työn suurin haaste oli tehdä kokonainen ohjelmistoprojekti itsenäisesti alusta loppuun. Työn aikana jouduttiin soveltamaan ja muuttamaan asioita useasti, mikä johtui lähinnä tekijän vähäisestä kokemuksesta käytettyjen teknologioiden parissa. Tämä kuitenkin tarkoitti sitä, että tekijä oppi lukuisan määrän keskeisiä ohjelmistokehitykseen liittyviä asioita, ja pystyi mukautumaan sen mukana tuomiin haasteisiin.

Työn laajuuden hahmottaminen alkuvaiheessa oli hankalaa, ja tämä kostautui työmäärällisesti sekä aikataulullisesti. Jälkikäteen mietittynä monia asioita olisi voitu tehdä paremmin ja ehdottomasti pienemmän kokonaisuuden hallinta olisi helpottanut. Erityisesti työn loppuvaiheessa jatkuva toiminnallisuuksien iterointi toi haasteita, kun kokonaisuus alkoi hahmottua paremmin. Järkevämpi omien komponenttien ja sekä tilahallinnan käyttö olisi vähentänyt työmäärää huomattavasti, sillä monessa kohtaa jouduin toistamaan virheiden korjaamista monessa eri paikassa koodia.

Henkisesti työ oli kaksijakoinen, sillä tekijän motivaatio oli todella korkea, mutta oman applikaation julkaisuun liittyvät paineet olivat työn aikana suuri stressitekijä. Tavoitellut julkaisupäivät jouduttiin siirtämään useaan kertaan siitä syystä, että applikaatio ei ollut läheskään valmis eikä keskeneräistä tuotetta haluttu markkinoille. Opinnäytetyön osalta applikaatio saatiin tarpeeksi hyvään malliin, ja suunnitelmani on jättää se sellaisenaan elämään ikään kuin mallikappaleeksi. Applikaatiosta tehdään myöhemmin toinen versio todellista julkaisua varten, jolloin siihen lisätään vielä brändäys ja korjataan joitain toiminnallisuuksia.

5.1 Keskustelu ja jatkoehdotukset

Next.JS on teknologiana monipuolinen ja kattava viitekehys web-applikaatioiden tekemiseen. Se tukee erinomaisesti erityisesti React-kirjastoa ja yhdessä Vercel alustan kanssa, se muodostaa tehokkaan kombinaation ohjelmistokehitykselle, ylläpidolle ja julkaisualustaksi. Sen ilmaisversio on kattava ja siihen voidaan lisäksi liittää tietokanta. Suosittelen näitä teknologioita vahvasti niin pienten kuin suurempien ohjelmistojen kehitykseen, mutta sillä huomiolla, että opeteltavaa on paljon.

Sen takia kannattaa lähteä liikkeelle pienestä projektista ja edetä asia kerrallaan. Minusta tämä olisi erinomainen aihe yliopiston kurssitarjontaan syventäviin opintoihin.

Vaikka tanssiapplikaatio jäi vielä julkaisematta, niin koen todellisena mahdollisuutena rakentaa siitä sellaisen version, mistä voin aloittaa omaa yritystoimintaa. Työn aikana olen visioinut todella mielenkiintoisia asioita ja hyvin tehtynä, siitä voidaan tuottaa kansainvälinen menestystuote. Tähän on toki paljon matkaa, mutta pitäähän sitä tavoitteita olla.

Seuraavina steppeinä applikaation kehitykselle on saada tapahtumajärjestäjät mukaan ja oikeasti käyttämään vähintäänkin testimielessä applikaatiota. On tärkeää saada oikeaa dataa ja oikeita käyttäjiä, jotta todellista käyttöä päästään testaamaan. Hyvänä puolena on se, että minulla on kontaktit valmiina. Huonona puolena on se, että näistä pitäisi suurin osa saada mukaan, jotta applikaatiosta olisi hyötyä loppukäyttäjille. Haasteita siis vielä riittää.

Teknisesti erilaisia toimintoja olen miettinyt jo pitkälle tulevaisuuteen. Toiminnan ylläpitämiseksi olisi hyvä saada sovellus monetisoitua ja sen saada verran rahavirtaa, että sovellusta olisi järkevää lähteä kehittämään kunnolla. Vähintäänkin harrastuksen tasolla olen valmis tekemään tätä nollabudjetilla, mutta oikeassa bisnesmielessä haluan toki saada tästä jotain takaisin.

Applikaation on tarkoitus laajentua kattamaan kaikki mahdolliset tanssilajit, myös soolotanssin ja ryhmätanssit sekä kilpatanssin, ja tapahtumien lisäksi haluan saada tanssikoulut, opettajat, tanssipaikat, esiintyjät, tanssikaupat yms. tanssiin liittyvät tahot keskitettyä yhteen paikkaan, josta niitä on helppo etsiä, sekä yhdistää toisiinsa.

Lisäksi haaveilen keskustelupalstasta, videoiden ja kuvien postaamisesta sekä autokyytien järjestämisestä. Jos homma lähtee toimimaan, niin ehdottoman tärkeää olisi tehdä mobiilisovellus, mikä toimisi Suomessa (miksei myös kansainvälisesti) tanssitapahtumien hakupaikkana.

Lähteet

Agafonkin, V. 2023. Leaflet - an open-source JavaScript library for mobile-friendly interactive maps. Luettavissa: https://leafletjs.com. Luettu: 15.9.2023.

Bootstrap. 2023. Bootstrap - The most popular HTML, CSS, and JS library in the world. Luettavissa: https://getbootstrap.com. Luettu: 21.10.2023.

Clerk. 2023. Clerk - Authentication and User Management. Luettavissa: https://clerk.com. Luettu: 13.10.2023.

Flowbite. 2023. Flowbite - Build websites even faster with components on top of Tailwind CSS. Luettavissa: https://flowbite.com. Luettu: 15.10.2023.

Flowbite-React. 2023. Flowbite React - UI Component Library. Luettavissa: https://www.flowbite-react.com. Luettu: 21.9.2023.

Foundation. 2023. Foundation - The most advanced responsive front-end framework in the world. Luettavissa: https://get.foundation. Luettu: 15.10.2023.

GeeksforGeeks. 2023a. Top Front-End Frameworks in 2023. Luettavissa: https://www.geeksforgeeks.org/top-front-end-frameworks. Luettu: 23.9.2023.

GeeksforGeeks. 2023b. What is SQL. Luettavissa: https://www.geeksforgeeks.org/what-is-sql. Luettu: 3.11.2023.

Git. 2023. Git. Luettavissa: https://git-scm.com. Luettu: 26.10.2023.

GitHub. 2023. GitHub – Let's build from here. Luettavissa: https://github.com. Luettu: 29.10.2023.

Kysely. 2023. Kysely - The type-safe SQL query builder for TypeScript. Luettavissa: https://kysely.dev. Luettu: 4.11.2023.

Microsoft. 2023. Visual Studio Code - Code editing. Redefined. Luettavissa: https://code.visualstudio.com. Luettu: 23.10.2023.

Mozilla. 2023a. Fetch API. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API. Luettu: 15.9.2023.

Mozilla. 2023b. Introduction to the DOM. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. Luettu: 15.9.2023.

Mozilla. 2023c. What is CSS? Luettavissa: https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS. Luettu: 15.9.2023.

Next.JS. 2023a. Dynamic routes. Luettavissa: https://nextjs.org/docs/app/building-your-application/routing/dynamic-routes. Luettu: 15.9.2023.

Next.JS. 2023b. Installation. Luettavissa: https://nextjs.org/docs/getting-started/installation. Luettu: 15.9.2023.

Next.JS. 2023c. Introduction. Luettavissa: https://nextjs.org/docs. Luettu: 15.9.2023.

Next.JS. 2023d. Project Organization and File Colocation. Luettavissa: https://nextjs.org/docs/app/building-your-application/routing/colocation. Luettu: 15.9.2023.

Next.JS. 2023e. Route Handlers. Luettavissa: https://nextjs.org/docs/app/building-your-application/routing/route-handlers. Luettu: 15.9.2023.

Next.JS. 2023f. Routing Fundamentals. Luettavissa: https://nextjs.org/docs/app/building-your-application/routing. Luettu: 15.9.2023.

Next.JS. 2023g. Client Components. Luettavissa: https://nextjs.org/docs/app/building-your-application/rendering/client-components. Luettu: 15.9.2023.

NextAuth.JS. 2023. NextAuth.JS - Authentication for Next.js. Luettavissa: https://next-auth.js.org. Luettu: 29.9.2023.

OpenAI. 2023. ChatGPT - Get instant answers, find creative inspiration, learn something new. Luettavissa: https://openai.com/chatgpt. Luettu: 1.10.2023.

PostgreSQL.org. 2023. About PostgreSQL. Luettavissa: https://www.postgresql.org/about. Luettu: 4.10.2023.

React. 2023. Your First Component. Luettavissa: https://react.dev/learn/your-first-component. Luettu: 23.9.2023.

React-Leaflet. 2023. React-Leaflet - React components for Leaflet maps. Luettavissa: https://react-leaflet.js.org. Luettu: 9.10.2023.

Tailwind. 2023. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. Luettavissa: https://tailwindcss.com. Luettu: 16.10.2023.

Tanssi.net. 2023. Suomen Tanssipalvelin. Luettavissa: https://tanssi.net. Luettu: 4.12.2023.

Timma.fi. 2023. Timma - Kampaamot, parturit, kauneushoitolat ja hierojat. Luettavissa: https://timma.fi. Luettu: 1.6.2023.

Törmänen, A. 2023. Tanssikoulu Antti Törmänen. Luettavissa: https://www.anttitormanen.com. Luettu: 3.11.2023.

TypeScript. 2023. TypeScript for JavaScript Programmers. Luettavissa: https://www.typescript-lang.org/docs/handbook/typescript-in-5-minutes.html. Luettu: 29.9.2023.

Vercel. 2023. Vercel - Build and deploy the best Web experiences with The Frontend Cloud. Luettavissa: https://vercel.com. Luettu: 10.10.2023.

W3Schools.com. 2023a. SQL Syntax. Luettavissa: https://www.w3schools.com/sql/sql_syntax.asp. Luettu: 10.2023.

W3Schools. 2023b. HTML Introduction. Luettavissa: https://www.w3schools.com/html/html_intro.asp. Luettu: 20.10.2023.

Wikipedia. 2023a. ECMAScript. Luettavissa: https://en.wikipedia.org/wiki/ECMAScript. Luettu: 29.10.2023.

Wikipedia. 2023b. Minimum Viable Product. Luettavissa: https://en.wikipedia.org/wiki/Minimum_viable_product. Luettu: 3.11.2023.

Wikipedia. 2023c. Objekti-relaatiokartoitus. Luettavissa: https://fi.wikipedia.org/wiki/Objekti-relaatiokartoitus. Luettu: 3.11.2023.