LAPIN AMK
Lapland University of Applied Sciences

TUNTEIDEN VISUALISOINTI WEB-POHJAISELLA SOVELLUKSELLA ASIAKASKOKEMUKSEN ARVIOINNISSA

Niemi Kalle

Opinnäytetyö

Tieto- ja viestintätekniikka
Insinööri (AMK)

2024

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tieto- ja viestintätekniikka
Insinööri (AMK)

Tekijä	Kalle Niemi	**Vuosi**	2024
Ohjaaja	Tauno Tepsa		
Toimeksiantaja	Lapin AMK:n FrostBit-laboratorio		
Työn nimi	Tunteiden visualisointi web-pohjaisella sovelluksella asiakaskokemuksen arvioinnissa		
Sivumäärä	34		

Työn tavoitteena oli suunnitella ja toteuttaa mobiilisovellus, jolla pystyttäisiin ymmärtämään omia tunteita paremmin itsereflektion avulla ja jota voitaisiin käyttää asiakkaiden tunnetilan muuttumisen osoittamiseen tuotteen tai palvelun suhteen. Toimeksiantajana toimi Lapin AMK:n FrostBit-laboratorio.

Opinnäytetyö on jaettu kolmeen osioon. Ensimmäiseksi käydään läpi teoriapohjaa, jossa esitellään käytetyt teknologiat, jonka jälkeen syvennytään projektin vaatimusten esittelyyn. Viimeisenä käydään läpi projektin toteutuksen aikana tehtyjä valintoja ja lopullisen käyttöön liittyviä ratkaisuja.

Projektin tuloksena on verkkoselaimen sisällä toimiva sovellus, jolla yhtiöt tai organisaatiot voivat saada hyödyllistä tietoa heidän palvelunsa vaikutuksesta kuluttajien tunteisiin.

Avainsanat käyttöliittymät, sovellusohjelmat, verkko-ohjelmointi, www-sivut

LAPIN AMK
Lapland University of Applied Sciences

Abstract of the thesis

Information and Communication Technology
Bachelor of Engineering

Author Kalle Niemi **Year** 2024
Supervisor Tauno Tepsa
Commissioned by Lapland UAS FrostBit software laboratory
Title Visualization of emotions in a web-based application for customer experience evaluation
Number of pages 34

The objective of this thesis was to design and implement a mobile application that can be used to understand one's own emotions better by self-reflection, and to analyse changes in customer's feelings regarding a product or service. The commissioner for this project was Lapland UAS FrostBit software laboratory.

The thesis is divided into three sections. The first section covers the theoretical background, introducing the technologies used, followed by a more in-depth presentation of the project requirements. The final section discusses the choices made during the implementation of the project and the solutions adopted for the final deployment.

The result of the project is an in-browser application that allows companies or organisations to obtain useful information about the impact of their services on consumers' emotions.

Keywords applications (computer programmes), user interfaces, web pages, web programming

SISÄLLYS

KÄYTETYT LYHENTEET

API	Application Programming Interface, ohjelmointirajapinta
AR	Augmented Reality, lisätty todellisuus
CI/CD	Continuous Integration & Continuous Delivery, jatkuva integraatio ja toimitus
CSR	Client-side Routing
HTML	Hypertext Markup Language, hypertekstin ohjelmointikieli
IDE	Integrated Development Interface, ohjelmointiympäristö
SVG	Scalable Vector Graphics
URL	Uniform Resource Locator
VPS	Virtual Private Server, virtuaalipalvelin

1 JOHDANTO

Opinnäytetyön toimeksiantajana toimii Lapin AMK:n FrostBit-laboratorio (jäljempänä FrostBit Software Lab). FrostBit Software Lab tuottaa ohjelmistoratkaisuja EU-hankkeille ja yksityisille asiakkaille. Tämä projekti on tehty tilaustyönä Lapin yliopiston hankkeeseen.

Tässä opinnäytetyössä käsitellään teknologioita ja toteutuksia, joita käytetään modernin web-sovelluksen tekemisessä sekä sen julkaisussa. Lisäksi käsitellään selkeän käyttöliittymän ja käyttäjäkokemuksen luomista ja tutkitaan keinoja, joilla käyttäjä voi visualisoida omia tunteita sovelluksen sisällä. Tämä opinnäytetyö osoittaa osaamistani web-pohjaisen sovelluksen tekemisen suunnittelu-, toteutus- ja käyttöönottovaiheissa.

Sovelluksen tarkoitus on toimia työkaluna, jota organisaatiot, kuten museot, taidegalleriat, matkailyritykset tai muut asiakkaidensa tunteista kiinnostuneet palveluntarjoajat voivat käyttää. (Björn, Miettinen, Jylkäs & Sarantou 2024.) Sovellus perustuu Empathy Business -hankkeen Tunteiden itsereflektointi -proof-of-concept (PoC) -konseptiin.

Opinnäytetyön Teknologiat ja menetelmät -pääotsikko on teoriaa sisältävä osa, jossa käydään läpi käytettyä ohjelmistoa ja syvennytään projektin tarkoitukseen. Sovelluksen suunnittelu -pääotsikossa esitellään asiakkaan visiota sovelluksesta ja esitetään haasteet, jotka sovelluksen on tarkoitus ratkaista. Sovelluksen toteutus -osiossa näytetään, miten sovellus toteutettiin ja miten sen toimitus tapahtui. Viimeiseksi pohditaan projektin onnistumista, haasteita ja hylättyjä ideoita tai ominaisuuksia.

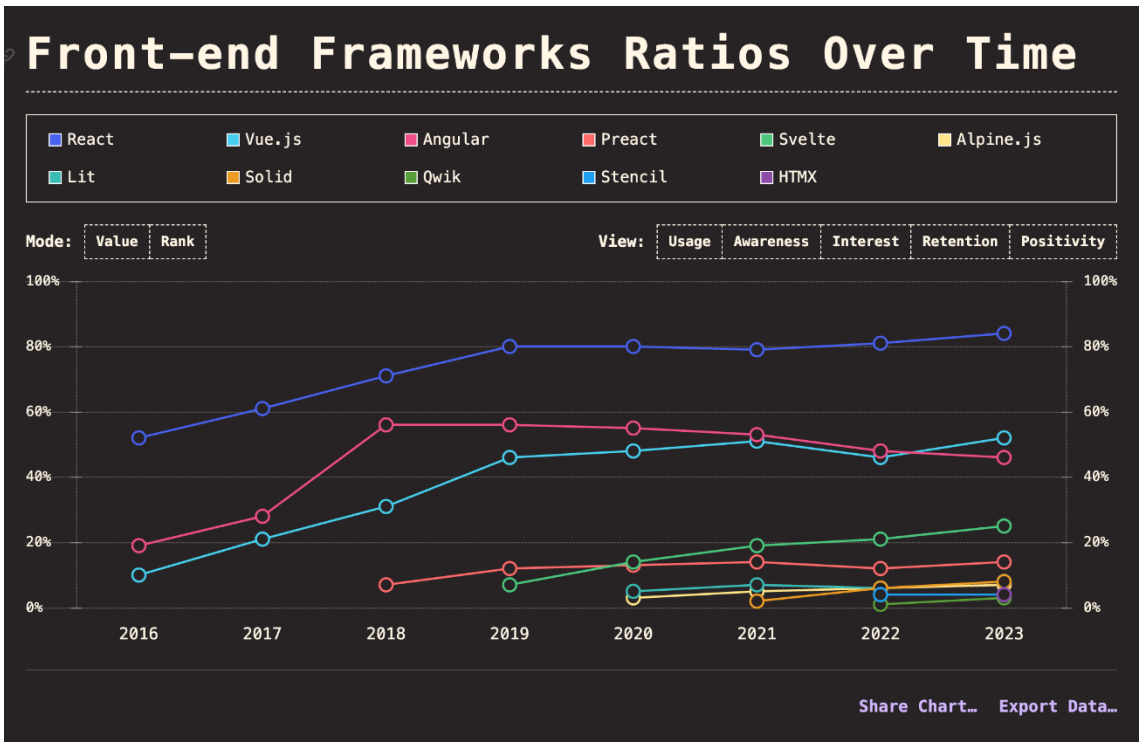
2 KEHITYKSEN TEKNISET VALINNAT

Web-pohjaisen sovelluksen rakentamiseen on useita eri menetelmiä. Sovelluskehykset tekevät helpoksi suurten sivustojen ja sovellusten luomisen, sillä komponenttien uudelleenkäytettävyydellä voi välttää saman painikkeen tai valikon tyylin ja toiminnallisuuden kirjoittamista useampaan kertaan. Lisäksi React-sovelluskehyksellä sovellus voidaan tuoda natiivisti mobiilialustoille eli sovellus voidaan muuttaa ladattavaksi puhelinsovellukseksi. Tämä oli tärkeää huomioida, sillä sovellus on tähdätty asiakkaille esimerkiksi museokäynnin aikana. Täytyi myös ottaa huomioon, että mikäli sovellus on saatavilla vain asennettuna, se lisää esteitä sovelluksen käyttämiseen.

2.1 React

Sovellus kehitettiin React-nimisellä sovelluskehyksellä, joka on alunperin Meta Platformsin (tunnettu myös nimellä Facebook) kehittämä. Tähän valintaan päädyttiin, koska React on pitänyt ensimmäistä sijaa suosituimpana sovelluskehyksenä web-sovellusten kehittämiseen monen vuoden ajan, ja sen tuki ei näy vähenevän lähitulevaisuudessa (State of JS 2023).

Web-kehittäjien kesken Reactin suosio on hieman yli 80 prosenttia, minkä jälkeen toisena on Vue noin 50 prosentin suosiolla (kuvio 1). State of JS:n diagrammi perustuu kyselyn tuloksiin, jossa web-kehittäjiltä on kysytty tietoisuutta ja kiinnostusta eri sovelluskehyksistä. Kyselyssä on huomioitu myös mielipiteiden positiivisuus ja kehittäjien määrän pysyvyys. (State of JS 2023.)



Kuvio 1. Front-end-sovelluskehysten suosio kehittäjien kesken (State of JS 2023)

2.2 React Router

Käytettäessä Reactin kaltaista sovelluskehystä voi hyödyntää Client-side Routingia eli asiakaspään reititystä. Tavallisesti web-sivu latautuu palvelimelta ja minkä tahansa linkin klikkaaminen pyytää palvelimelta täysin uuden sivun. CSR:ää käyttämällä näkymää tai dataa näytöllä voidaan vaihtaa ilman, että käyttäjä pyytää uutta sivua palvelimelta. (React Router 2024.)

React Routerin CSR:n avulla käyttäjä voi siirtyä selainhistoriassa eteen- ja taaksepäin käyttämällä selaimen historiapainikkeita. Lisäksi URL-osoitekenttään voidaan kirjoittaa suoraan linkki sovelluksen tiettyyn näkymään, jolloin käyttäjä navigoi suoraan haluttuun näkymään.

2.3 Vite

Vaikka JavaScript on tulkattu kieli, web-sovellukset tulee pakata ja optimoida, jotta kaikki tarvittavat koodit ja resurssit, kuten kuvat ja fontit ovat mukana sivustossa. Verkkoselaimet ymmärtävät suoraan HTML:ää, joten resurssit ja niiden

väliset yhteydet pitää muuttaa HTML-tiedostojen tukemaan muotoon. Tässä prosessissa käytetään työkaluja, jotka kääntävät ja pakkaavat koodin tehokkaasti. (Vite 2024; Webpack 2024.)

React-sovelluksen luonnin yhteydessä oletusarvona projektiin sisällytetään pakkausohjelmaksi Webpack. Tässä projektissa käytettiin kuitenkin Viteä, jonka väitetään olevan Webpackia nopeampi pakkaamisessa. Vite nopeuttaa pakkaamisen lisäksi myös sovelluksen kehittämistä, sillä muokatessa sivun koodia se mahdollistaa välittömät päivitykset selaimessa ilman, että selainikkunaa täytyy käydä uudelleenlataamassa tai virkistämässä (Vite 2024). Viten sivuilla tätä kutsutaan nimellä ’hot reload’, ja se tekee sovelluksen kehittämisestä sujuvampaa ja tehokkaampaa (Vite 2024).

Kun sovelluksen uusi versio julkaistaan, vanhat pakatut sivut korvataan uusilla, jolloin palvelin pitää käynnistää uudelleen. Kun koodimuutokset lisätään tuotantoon, nopeampi pakkaaminen on eduksi. Palvelin voi käsitellä muutokset pienemmässä ajassa, mikä vähentää käyttökatkojen pituutta ja parantaa käyttäjäkokemusta.

2.4 ESLint

ESLint on työkalu, jota käytetään koodin laadun ja tyylin valvontaan. Se suorittaa ennalta määritettyjä sääntöjä koodin läpi ja ilmoittaa virheistä ja puutteista. ESLint mahdollistaa erilaisten sääntöjen määrittämisen, mikä auttaa pitämään koodin yhtenäisenä ja helposti luettavana. (ESLint 2024)

Projektissa käytettiin ESLintin oletussääntöjä, jotka sopivat projektiin. ESLintin integrointi kehitysympäristöön tarkoitti, että kirjoitusvirheiden aiheuttamat virheet ja käyttämättömät koodin pätkät voitiin siivota pois nopeasti. ESLintin käyttö paransi myös tiimin yhteistyötä, koska kaikki kehittäjät noudattivat samoja koodityylejä ja käytäntöjä.

ESLintin hyödyntäminen tässä projektissa varmisti, että lopputuote oli helposti ylläpidettävä. Se auttoi myös varmistamaan, että koodi oli yhdenmukaista ja seurasi parhaita käytäntöjä, mikä oli tärkeää projektin onnistumisen kannalta.

2.5 I18next

Sovelluksen paikallistaminen on yksinkertainen, mutta hyödyllinen ominaisuus. Projektin alussa käyttöliittymän tekstit oli kirjoitettu englanniksi. Mahdollisissa testaustilaisuuksissa käyttäjäkunta olisi mahdollisesti monikulttuurinen, mutta suomen kielen tuli olla myös saatavilla. Projektiin haettiin ratkaisua, jolla olemassa olevaa englanninkielistä käyttöliittymää voitaisiin hyödyntää.

Parhaita käytäntöjä tutkittaessa kehityksen aikana tutustuttiin I18next-kirjastoon, joka on yksi suosituimmista ja käytetyimmistä ratkaisuista tällaiseen tarkoitukseen. I18next on alunperin luotu 2011 ja saa jatkuvaa tukea kehittäjiltä (I18next 2022).

2.6 Material UI

Material UI on Reactille tehty käyttöliittymäkirjasto, joka perustuu Googlen suunnittelemaan Material Design -suunnittelumalliin. Muut sovelluskehykset, kuten Vue ja Angular käyttävät erilaisia toteutuksia, kuten Vuetifyä ja Angular Materialia. (Material Design 2024b.)

Material UI tarjoaa modernin ulkoasun moniin HTML-elementteihin, mutta se eroaa Cascading Style Sheets -kirjastosta (CSS) siten, että sen komponentit eivät rajoitu HTML:n vakiokomponentteihin. CSS-tyylitiedosto on tärkeä osa websivun ja -sovelluksen elekieltä ja selkeän käyttöliittymän luomista. Material UI:n komponentteja käyttämällä voi luoda selkeän käyttöliittymän ja säästää aikaa pienissä vaivannäöissä, kuten tekstikentän automaattisen rivien kasvamisen itse ohjelmoinnissa käyttämällä sen sijaan valmiita komponentteja.

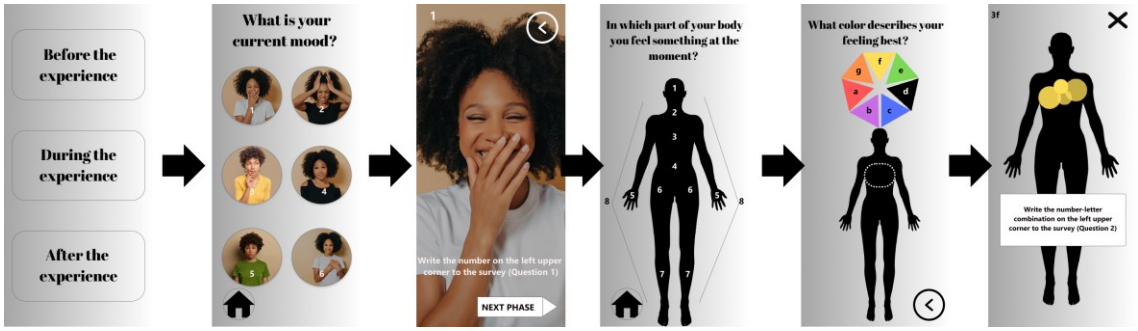
3 SOVELLUKSEN SUUNNITTELU

3.1 Prototyypin vaatimukset

Sovelluksen suunnittelu alkoi interaktiivisella ”rautalankamallilla”, jonka Lapin yliopisto oli toteuttanut hankkeen briefing-asiakirjan yhteydessä (kuvio 2). Tämä prototyyppi toimi pohjana sovellukselle.

Prototyypissä ensimmäisessä ruudussa on kolme painiketta, jotka avaavat tunnetilaa kartoittavan kyselyn. Painikkeet kuvastavat jonkin palvelun tai tapahtuman vaiheita, ja samat kysymykset kysytään kussakin vaiheessa. Ensimmäiseksi pyydetään valitsemaan nykyistä tunnetilaa parhaiten kuvastava ilme rajatusta perusilmeiden listasta.

Kasvojen ilmeen valitsemisen jälkeen napautetaan kehonosaa, ja käyttäjää pyydetään paikantamaan, missä tunnetila vaikuttaa hänen kehossaan. Lisäksi tunnetta pyritään kuvastamaan värillä. Eri kehonosille on annettu oma numero, ja jokaiselle värille on annettu oma kirjain. Kysymysten loputtua oma vastaus määrittyy valitun kehonosan ja sijoitetun tunteen värin mukaan.



Kuvio 2. Prototyypin toiminta

Prototyypissä vastaajia pyydettiin kirjoittamaan oma vastaus numero- ja kirjainyhdistelmänä erilliselle lomakkeelle. Asiakkaat olivat aikaisemmin käyttäneet paperilomakkeita, joissa ihmiskehoon sai laittaa ympyränmuotoisia värillisiä tarroja tuntemuksien kuvailun avustamiseksi. Projektissa lähdettiin liikkeelle siitä, että paperilomake syrjäytettäisiin pois ja kaikki data kerättäisiin sähköisesti. Sähköi-

sessä kyselyssä on myös mahdollista tehdä automaattinen yhteenveto vastauksista, mikä ei ollut vaatimus, mutta pyrin mahdollistamaan tämän ominaisuuden lopputulokseen.

Jotta useampi yritys voisi hyödyntää sovellusta asiakasanalyysissa, ”tapahtumiin” liittymisen pitää olla mahdollista. Kysymykset voivat vaihdella tapahtuman mukaan, joten aiemmin mainitun prototyypin toteuttamisen lisäksi sovellukseen tarvittiin kysymyseditori. Myös yhteenvetojen tarkastelua ja kysymysten editointia varten jonkinlainen tunnistautuminen olisi tarpeellinen.

3.2 Käyttöliittymän helppokäyttöisyys

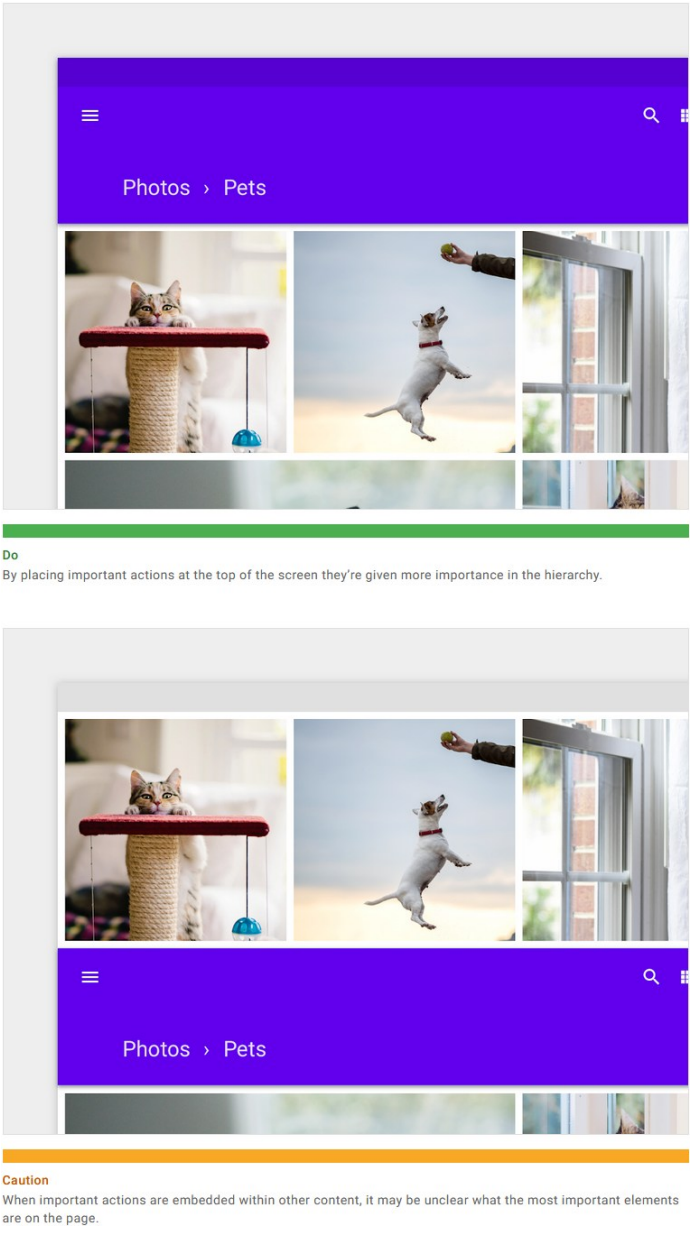
10 käytettävyyden heuristiikkaa käyttöliittymäsuunnittelua varten on Nielsen & Norman Groupin kehittämä ohje nyrkkisäännöistä selkeän ja ymmärrettävän käyttöliittymän luomiseen. (Nielsen & Norman Group 2024.) Näitä sääntöjä seuraamalla sivuston tai sovelluksen saavutettavuus paranee. Tässä on muutamia pääkohtia:

- Mikäli käyttäjä painaa painiketta, joka poistaa käyttäjän tai tekee muun tuhoavan toiminnon, kysytään aina vahvistus.
- Jos data ei ole välittömästi näkyvillä, käyttäjälle tulee näyttää latausmerkki.
- Toiminnan täytyy olla tutun periaatteen mukaista, ja painikkeiden tulee käyttää laajalti ymmärrettyjä symboleja.
- Käyttäjälle mahdollisesti tuntemattomien ammattisanojen käyttöä vältetään.
- Helppokäyttöisyyden vuoksi käyttäjälle tulee tarjota asetuksia esimerkiksi kontrastin, tekstin koon tai lukijan muodossa.

Material UI:ta käyttävän projektin tulee myös seurata Googlen avoimen projektin Material Designin helppokäyttöisyyssuosituksia. Yleisien nyrkkisääntöjen sijaan

ne tarjoavat käytännön esimerkkejä. Kuvio 3 osoittaa, miten sisällön sekoittaminen staattisten elementtien, kuten valikoiden kanssa rikkoo odotetun tärkeysjärjestyksen lukijalle. Näihin kuuluvat:

- Riittävä kontrasti ja koko
- Selkeä tärkeysjärjestys (esim. Navigaatiopalkissa ei tule olla sivun sisältöä)
- Tärkeimmän tiedon saatavuus yhdellä silmäyksellä



Kuvio 3. Material Designin ohjeistus sisällön tärkeysjärjestyksestä (Material Design 2024a)

Material UI:n hyödyntäminen teki kontrastin säilyttämisestä helppoa. Sovellus saatiin noudattamaan valkoista tai tummaa teemaa siten, että teksti vaihtaa väriä taustavärin mukaisesti niin, että teksti pysyy luettavana.

3.3 Tietosuoja ja tietoturva

Koska projektiin kysyttiin käyttäjäksi ulkopuolisia henkilöitä, applikaation käyttäjille täytyi näyttää tietosuojaseloste ja oli syytä kysyä heidän hyväksyntäänsä ennen tietojen keräämistä. Euroopan yksityisyydensuojalainsäädännön vuoksi projekti vaati GDPR-selosteen, jossa tarkennetaan, mitä tietoja sovellus kerää käyttäjistä ja mihin tarkoitukseen niitä käytetään.

Jotta projektiin ei tarvittaisi tutkimuslupaa, kerätty data täytyi olla minimaalista. Lapin yliopisto halusi saada tietoa vastaajien kansalaisuudesta ja ikäluokasta, sillä tunteita mitatessa eri-ikäinen henkilö voi olla eri mieltä tunteiden tarkoituksesta ja toisessa maassa kasvanut voi käyttää eri väriä tunteen kuvaamiseen, jossa suomalainen käyttäisi toista väriä. Sovelluksen käyttämisen esteiden vähentämiseksi käyttäjän ei tarvitse kirjautua liittyäkseen tapahtumaan ja täyttääkseen vastauksia. Tämän seurauksena tiedot olivat anonyymejä.

Sovelluksen ja palvelimen/tietokannan vastuun vähentämiseksi kirjautumistietojen hallintaan käytettiin OAuth2-standardia. Näin salasanan salausmenetelmää ei tarvinnut toteuttaa itse. Huolimattomasti toteutetussa tietokannan suojaamisessa ja salasanojen salaamisessa on riski tietomurtoihin tai datan menettämiseen. Kirjautumalla OAuth2-palveluntarjoajan, kuten Googlen avulla käyttäjästä saadaan vain tarvittu tieto. Omien tapahtumien muokkaamista ja vastanneiden tuloksien katsomista varten pelkkä Google-käyttäjän täsmääminen on riittävää.

4 SOVELLUKSEN TOTEUTUS

Sovelluksen toteuttaminen alkoi aluksi prototyypin toiminnallisuuden luomisella. Kyselynäkymän lisäksi käyttäjien tunnistautumisen lisääminen oli välttämätöntä, jotta Lapin yliopisto voisi katsoa yhteenvetoja ja vastauksia sovelluksen sisällä.

4.1 React-sovelluksen luominen

JavaScript-sovelluskehyksen käyttäminen edellyttää JavaScriptiä tulkkaavan ohjelman asennusta, mihin kuuluvat esimerkiksi NodeJS ja Deno. NodeJS:ää käyttäessä muiden luomaa koodia voidaan käyttää lisäämällä paketteja. Paketti voi olla esimerkiksi kirjasto, joka sisältää valmiita toimintoja ja ominaisuuksia, joita voidaan käyttää sovelluksessa. Ulkoisten pakettien hallintaan käytetään paketinhallintaohjelmistoja, jotka toimivat keskitettynä paikkana kaikille jaetuille koodipaketeille. Yleisimmät paketinhallintaohjelmistot JavaScriptille ovat Node Package Manager (NPM) ja Meta Platformsin luoma Yarn.

React-sovelluksen luomiseen on monta tapaa, joista ehkä tunnetuin on ajamalla NodeJS:llä komento *'npx create-react-app'*. Tämä käyttää Webpackia, ja mainitsin jo aikaisemmin projektissa käytettävän ViteJS:ää. Reactin oma dokumentaatio on alkanut suosia Next.js-sovelluskehystä tämän opinnäytetyön kirjoittamisen aikana.

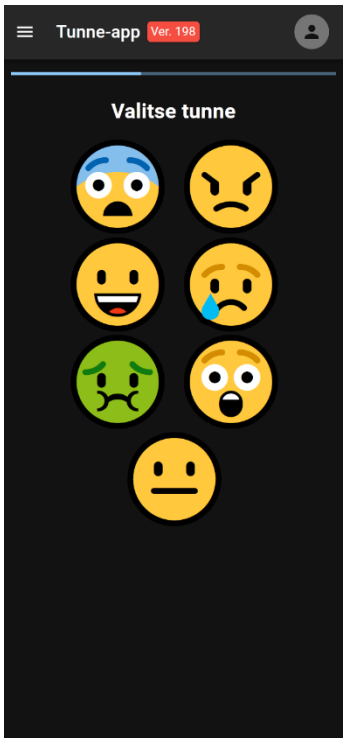
Sovellus saatiin alkuun Viten ”Getting Started” -ohjeiden mukaisesti komennon *'npm create vite@latest'* avulla. Komennon ajaminen loi React-sovelluksen pohjan sijaintiin, jossa komentokehote oli avattu.

4.2 Prototyypin toiminnallisuuden mukaileminen

Prototyypin kyselypohja oli sopiva mobiilisovellukseen ja näkymä pyrittiin pitämään yksinkertaisena niin, että se mahtuu yhdelle ruudulle, kuten kuviossa 4 näkyvä puhelimen kuvankaappaus osoittaa. Siten kysymyksen keskeinen sisältö voidaan näyttää isommalla ilman, että ruutua tarvitsee raahata alas. Vastaamalla kysymyksiin vaiheittain käyttäjän ei tarvitse myöskään selata sivua edestakaisin tarkistaen, että häneltä ei puutu vastausta mihinkään kohtaan.

Prototyypissä erilaisia ilmeitä kuvaavat valokuvat olivat Adobe Stock -kuvavalikoimasta, joka tarjoaa arkistokuvia digitaaliseen suunnitteluun. Adoben arkistokuvien käyttäminen on integroitu muihin palveluihin, kuten Adobe XD -ohjelmistosuunnitteluohjelmaan, jolla prototyyppi oli tehty.

Lopullisessa tuotteessa tuli käyttää kuvia, joihin FrostBitillä ja Lapin yliopistolla on oikeudet. Projektia toteuttaessa harkittiin valokuvaajan ja näyttelijän palkkaamista, jolla saataisiin yhdenmukaiset ihmiskasvojen perusilmeet. Sovellusta ohjelmoitiin samanaikaisesti, ja kuvien paikalla pidettiin tunteita kuvaavia Emojimerkkejä (kuvio 4).



Kuvio 4. Nykyisen tunnetilan perusteella kasvojen ilmeen valitseminen

Emojit voivat olla ulkonäöltään hieman erilaisia laitteesta riippuen, mutta niille on asetettu tarkat tekstikuvaukset ja ulkonäkövaatimukset, joista päättää voittoa tavoittelematon yhtiö Unicode (Unicode 2024). Laitevalmistajat voivat toteuttaa Emojit seuraten näitä ohjeistuksia. Perusilmeiden kuvaamiseen Emojit olivat tarpeeksi yhtenäisiä.

4.2.1 Tunteiden sijoittaminen kehoon

Tunteen paikallistamista käsittelevä kysymys oli rajoitettu prototyypissä vain ennalta määritettyihin kehonosiin, mutta vaatimuksena oli pystyä sijoittamaan värillisiä palloja täysin valittuun kohtaan. Palloja tuli myös olla mahdollista lisätä useaan kehonosaan siten, että eri kehonosille voidaan antaa eri värejä.

Pallojen lisääminen oli toteutettu siten, että koko keho on yhden div-elementin sisällä, joka kaappaa hiiren osoittimen sijainnin napautuksen yhteydessä. Kaikki kehonosat ovat tämän div-elementin sisällä (kuvio 5).

```
const handleMouseMove = (event) => {
    const boundingRect = event.target.getBoundingClientRect();
    const x = event.clientX - boundingRect.left;
    const y = event.clientY - boundingRect.top;
    setMousePosition({x, y});
    console.log('Mouse Position: ', mousePosition)
}


const handleClick = (id: string) => {
    setColorModalOpen(true);
    setSelectedPart(true);
    setSelectedPartName(id);
}
```

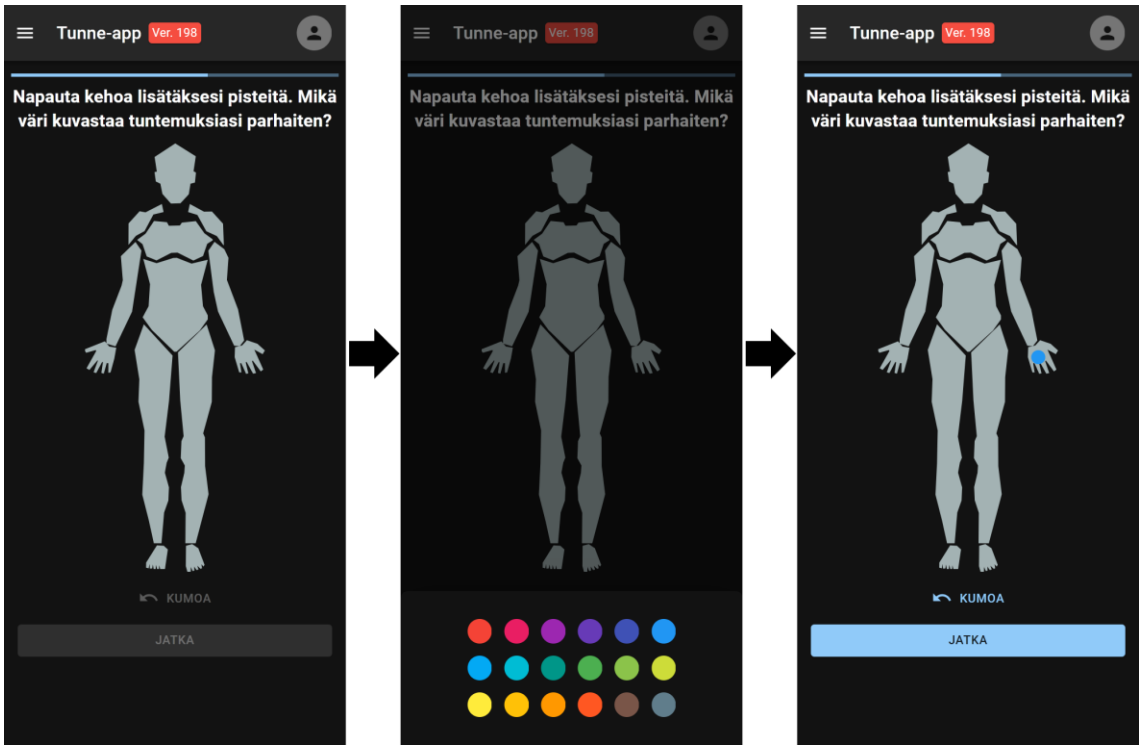
```
return (
    <>
        <div onClick={handleMouseMove} className="body">
            <HeadIcon onClick={() => handleClick('head')}>
            </HeadIcon>
            <LeftShoulderIcon onClick={() => handleClick('leftShoulder')}>
            </LeftShoulderIcon>
            <RightShoulderIcon onClick={() => handleClick('rightShoulder')}>
            </RightShoulderIcon>
            <LeftArmIcon onClick={() => handleClick('leftArm')}>
            </LeftArmIcon>
            <RightArmIcon onClick={() => handleClick('rightArm')}>
            </RightArmIcon>
            <ChestIcon onClick={() => handleClick('chest')}>
            </ChestIcon>
            <StomachIcon onClick={() => handleClick('stomach')}>
            </StomachIcon>
            <LeftLegIcon onClick={() => handleClick('leftLeg')}>
            </LeftLegIcon>
            <RightLegIcon onClick={() => handleClick('rightLeg')}>
            </RightLegIcon>
            <RightHandIcon onClick={() => handleClick('rightHand')}>
            </RightHandIcon>
            <LeftHandIcon onClick={() => handleClick('leftHand')}>
            </LeftHandIcon>
            <LeftFootIcon onClick={() => handleClick('leftFoot')}>
            </LeftFootIcon>
            <RightFootIcon onClick={() => handleClick('rightFoot')}>
            </RightFootIcon>
        </div>
```

Kuvio 5. Napautuksen kuuntelijafunktiot ja kehon osien hierarkia ohjelmakoodissa

Mikäli napautus on kehonosan kohdalla, värin valintaruutu asetetaan näkyville. Napautettu kehonosa merkitään myös nimellä. Jos tietoa halutaan tutkia esimerkiksi Excel-muodossa, osoittimen sijainti ei ole avuksi ja ainoa sijainnillinen tieto on kehonosan nimi.

Kehonosat toimivat HTML:n Scalable Vector Graphics (SVG) vektorigrafiikkaelementeillä. SVG-kuvat koostuvat pikseleiden sijaan erilaisista matemaattisista kuvioista koordinaatistossa. Näitä käyttämällä voidaan luoda kuvia, joita voidaan suurentaa rajattomasti. HTML:n vektorigrafiikkatoteutus mahdollisti sen, että halutun SVG-elementin päälle pystyttiin lisäämään ympyröitä, jotka pysyvät paikallaan näytön koosta riippumatta (kuvio 6).

Kuviossa 6 nuolet osoittavat toimintakaavan kulkua napautuksesta värin valintaan ja pallon ilmestymiseen. Käyttäjän osuudelta näkymä toimii niin, että kehon napauttamisen jälkeen värin valintaruutu avautuu alareunaan, minkä jälkeen värin valitseminen viimeistelee ympyrän lisäämisen.



Kuvio 6. Pallojen lisääminen kehoon

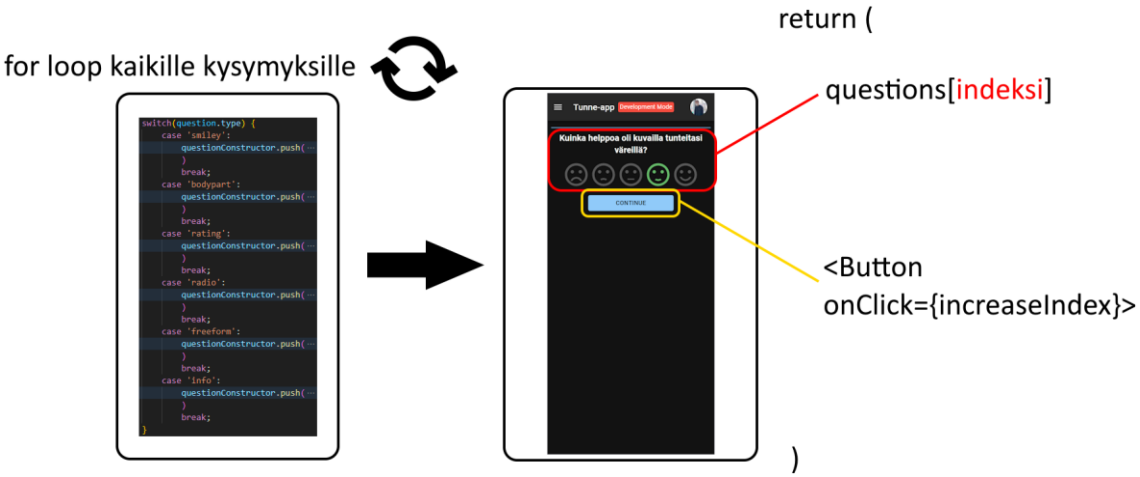
4.2.2 Kyselylomakkeen toiminta

Koska kysymyksiä oli syytä pystyä muokkaamaan tapahtumakohtaisesti, oli oleellista käyttää datapohjaista kyselylomaketta. Tapahtuman kysely sisältää alio-bjektina listan kustakin kyselyn vaiheesta tapahtumassa (kuvio 7). Näitä voivat olla esimerkiksi ’ennen taidenäyttelyä’ tai ’taidenäyttelyn jälkeen’. Kuviossa 7 kyselyn toiselle vaiheelle on annettu nimeksi ”Museokäynnin jälkeen”. Vaiheen sisällä on lista kysymyksistä, jotka kuuluvat tähän vaiheeseen.

Kuvio 7. Tapahtumalistan ja tapahtumien tietorakenne.

Kysymykset luotiin siten, että jokaisen kysymyksen teksti ja tyyppi voidaan määritellä. Näin ollen uusia kysymystyyppejä voidaan lisätä helposti tulevaisuudessa.

Kysymystyypille tehdään oma komponentti, joka sisältää siihen vastaamisen logiikan. Kun käyttäjä täyttää kyselyä, ohjelma näyttää kysymyksen tyypin mukaan erilaisen vastausruudun kyselyn jokaiselle kysymykselle (kuvio 8). Painike etenee kyselylistan seuraavassa indeksissä olevaan kysymykseen.



Kuvio 8. Kysymysten muuttaminen tekstiformaatista komponenteiksi

4.3 Tiedon muoto ohjelman ja tietokannan välillä

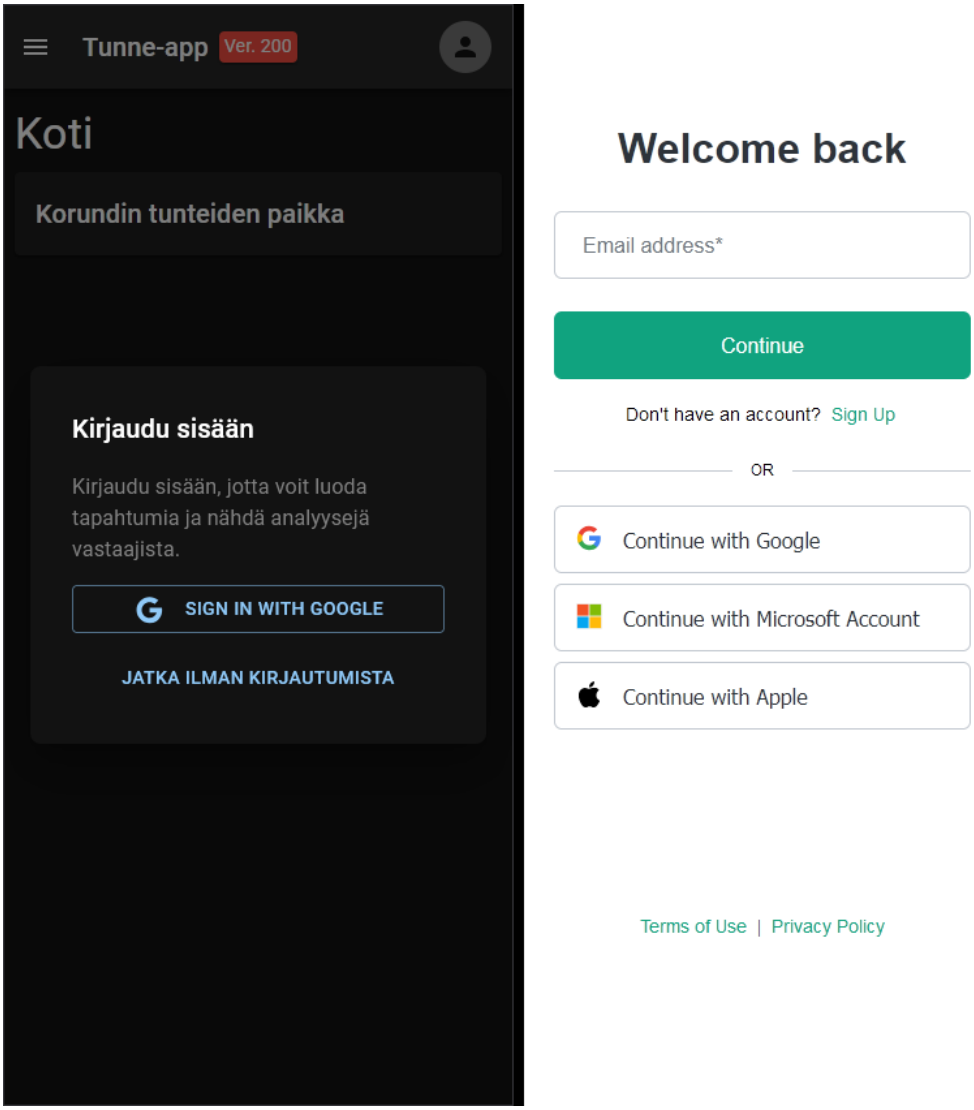
Sovelluksen ja back-end-palvelimen välisissä pyynnöissä käytettiin JavaScript Object Notation (JSON) -datatyyppiä. Tiedon välittämiseen valikoitui JSON siksi, koska JavaScript sisältää metodit JSON-datan koodaamiseen ja lukemiseen vakiovarusteluna ja se mukailee läheisesti JavaScriptin objekti-datatyyppiä (Mozilla 2024a).

Tapahtumien kyselyn kysymyspohja voitiin antaa aliobjektina tapahtuman tietoja haettaessa. Myös vastausten lähettäminen toimi siten, että niin sanottu ’vastauslomake’ oli yksi JSON-objekti, joka koostui vastausten listasta. Vastausten lähetyksessä ei otettu huomioon kyselyn vaiheita, vaan kaikki kysymykset lähetettiin siinä järjestyksessä, kun ne ilmenivät.

4.4 Käyttäjän kirjautumisen toteutus

Kirjautumisnäkymässä käytettiin tuttua ja helposti tunnistettavaa painiketta, joka perustuu OAuth2-kirjautumispainikkeiden ulkoasuun laajalti joka puolella inter-

nettiä (kuvio 9). Kuvassa näkyy myös OpenAI:n kirjautumisnäkymä, jonka ulkoasu muistuttaa hyvin paljon Firebasen tai Supabasen oletuskirjautumiskomponenttia. Kuvassa esiintyvät painikkeet viestivät käyttäjälle, että kirjautumiskokemus on samanlainen kuin muissa sovelluksissa, joissa käyttäjä kirjautuu jotain palveluntarjoajaa käyttäen.



Kuvio 9. Kirjautumisnäkymä painikkeineen

Kirjautuminen on toteutettu back-endin kautta. Kirjautuminen tapahtuu erillisellä verkko-osoitteella, jossa autentikaatiolle on oma ’endpoint’ eli polku. Kirjautumalla tällä sovelluksesta irrallisella sivulla palvelimelle jää eväste, joka sisältää aikaleiman kirjautumisen ajankohdasta ja vanhentumisesta. Back-end-palvelin toimii rajapintana, josta haetaan tälle palvelimen hallitsemalle sivulle kirjautuneen käyttäjän tiedot. Käyttäjän metatieto on käytännössä vain peilattu palvelimelta.

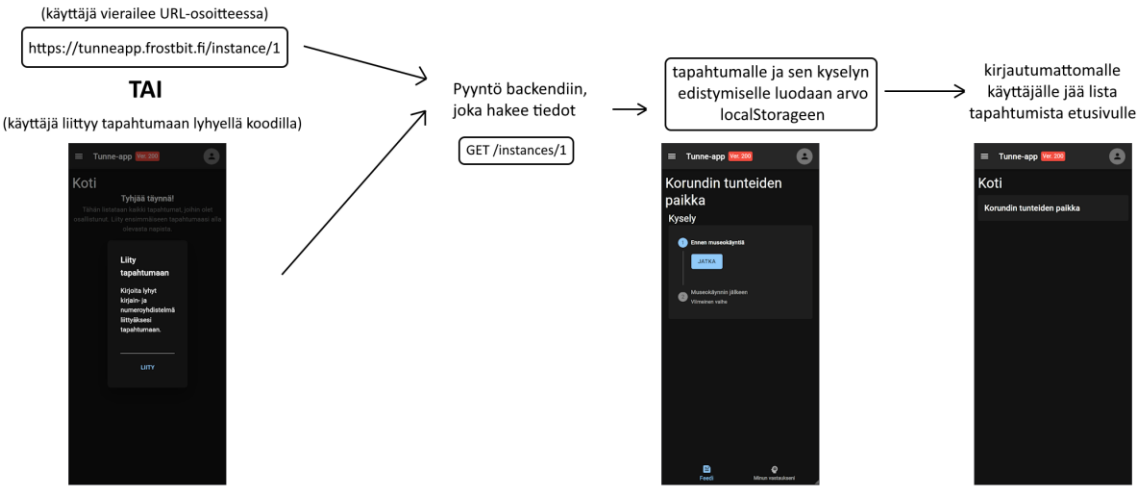
Sovelluksen näkökannasta kirjautumaton käyttäjä eroaa kirjautuneesta vain siten, että kaikki käyttäjään viittaavat muuttujat ovat tyhjiä.

4.5 Paikallinen tietojen pysyvyys

Koska käyttäjiä ei vaadittu kirjautumaan sovellukseen, keskeneräisten vastausten tallentaminen oli haasteellista. Käyttäjän tuli pystyä navigoimaan sovelluksen eri näkymiä siten, että asetukset, vastaukset ja tapahtumat, joihin on liitytty, pysyisivät tallessa tapahtuman alusta loppuun asti.

Yksi keino on tallentaa käyttäjän laitteelle eväste, jonka arvo on uniikki teksti- ja numeroyhdistelmä, ja käyttää tätä arvoa käyttäjän todentamiseen. Back-end-palvelimen tiimin kanssa yhdessä päätettiin, että tätä keinoa ei käytettäisi, sillä kertakäyttöisistä käyttäjistä tulisi suuri määrä turhia rivejä tietokantaan.

Ratkaisuna oli käyttää verkkoselaimeen rakennettua storage-ominaisuutta. SessionStoragen eli istunnon tallentamisen lisäksi selaimet tarjoavat localStoragen, joka on pitkäaikainen tallennustila tiedolle käyttäjän laitteella (Mozilla 2024b). LocalStoragea käyttämällä pystyttiin näyttämään sovelluksen etusivulla kaikki tapahtumat, joihin käyttäjä on liittynyt, vaikka selainikkuna suljettaisiin. (kuvio 10).

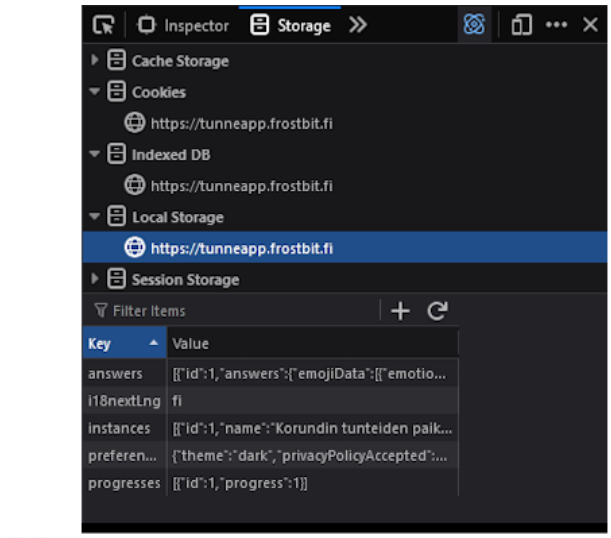


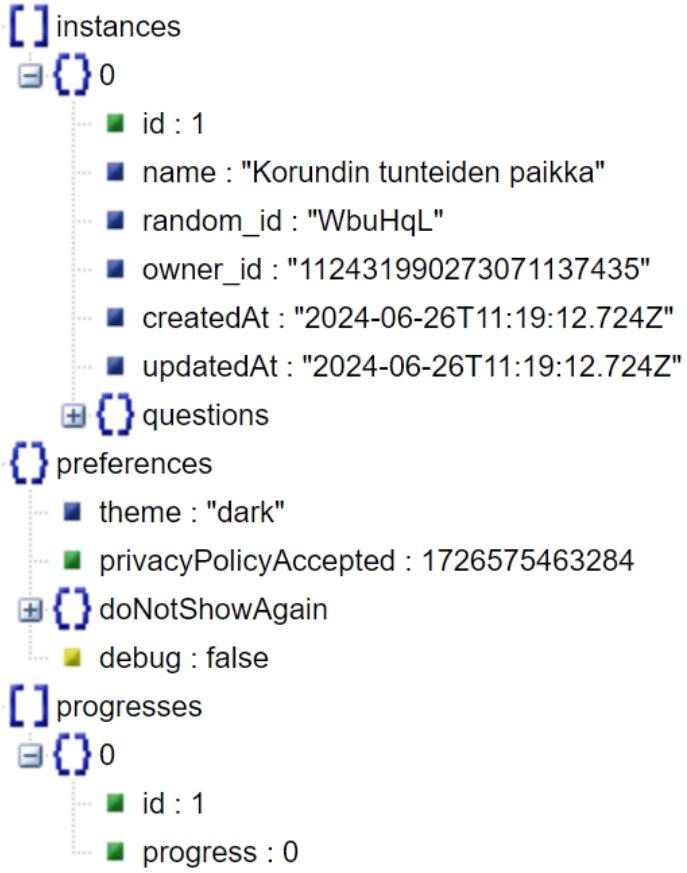
Kuvio 10. Tapahtumaan liittyminen

Jotta käyttäjä tietää vastanneensa tapahtuman kyselyyn, sovelluksessa rajoitettiin käyttäjää täyttämästä kyselyä uudelleen, kun vastaukset oli lähetetty. Tätä

varten keskeneräiset vastaukset ja tapahtumakohtainen edistyminen oli syytä tallentaa localStoragea käyttäen.

Tapahtumat säilytetään tapahtumakohtaisen ID:n perusteella, joka vastaa tapahtumaa tietokannassa (kuvio 11). Etusivulla tapahtuman nimellä näkyvä linkki hakee käytännössä backend-palvelimen kautta tätä ID:tä vastaavan tapahtuman tiedot.





Kuvio 11. Tapahtumien ja kyselyiden edistymisen formaatti localStoragessa

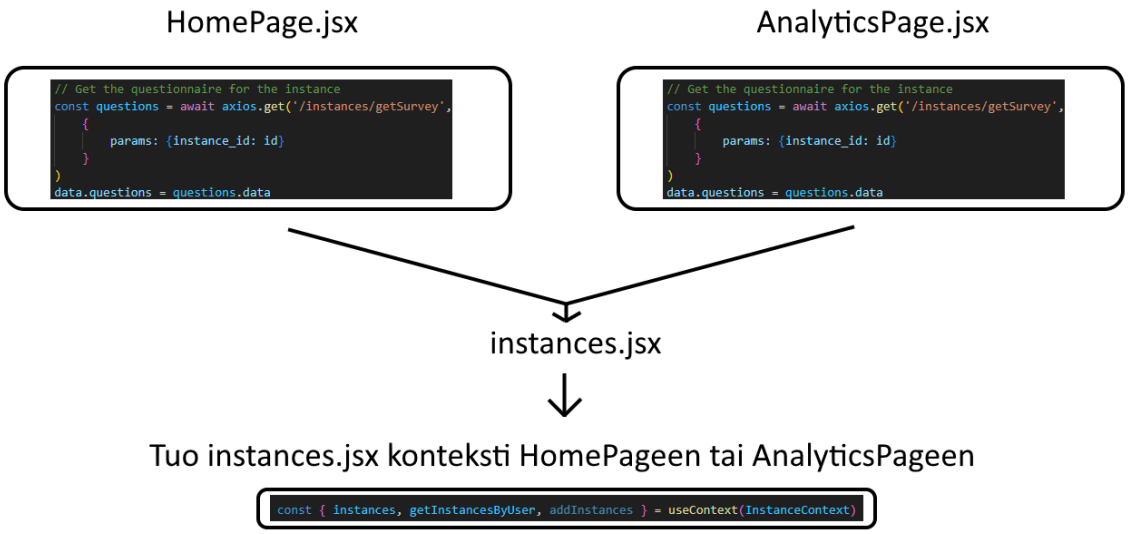
Koska palvelimen ei tarvinnut tallentaa käyttäjän edistymistä kyselyissä, ei ollut mahdollista asettaa edistymistä kyselyssä käyttäjän tietojen mukaisesti. Tämän vuoksi oli oleellista tallentaa kyselyiden keskeneräiset vastaukset ja nykyinen vaihe kyselyssä localStoragea käyttäen. Tämän toteutuksen heikkous on se, että käyttäjä aloittaa kyselyn aina alusta, jos laitetta vaihdetaan kesken tapahtuman.

4.6 Optimointi

Projektin koon kasvaessa ohjelmakoodi vaati erilaisia optimointikeinoja, jotta koodi pysyi helposti muokattavana ja yhdellä silmäilyllä luettavana. Optimointi toteutettiin luomalla uudelleen käytettäviä komponentteja, yksinkertaistamalla vertailulauseita ja pilkkomalla koodia pienempiin toimiviin osiin.

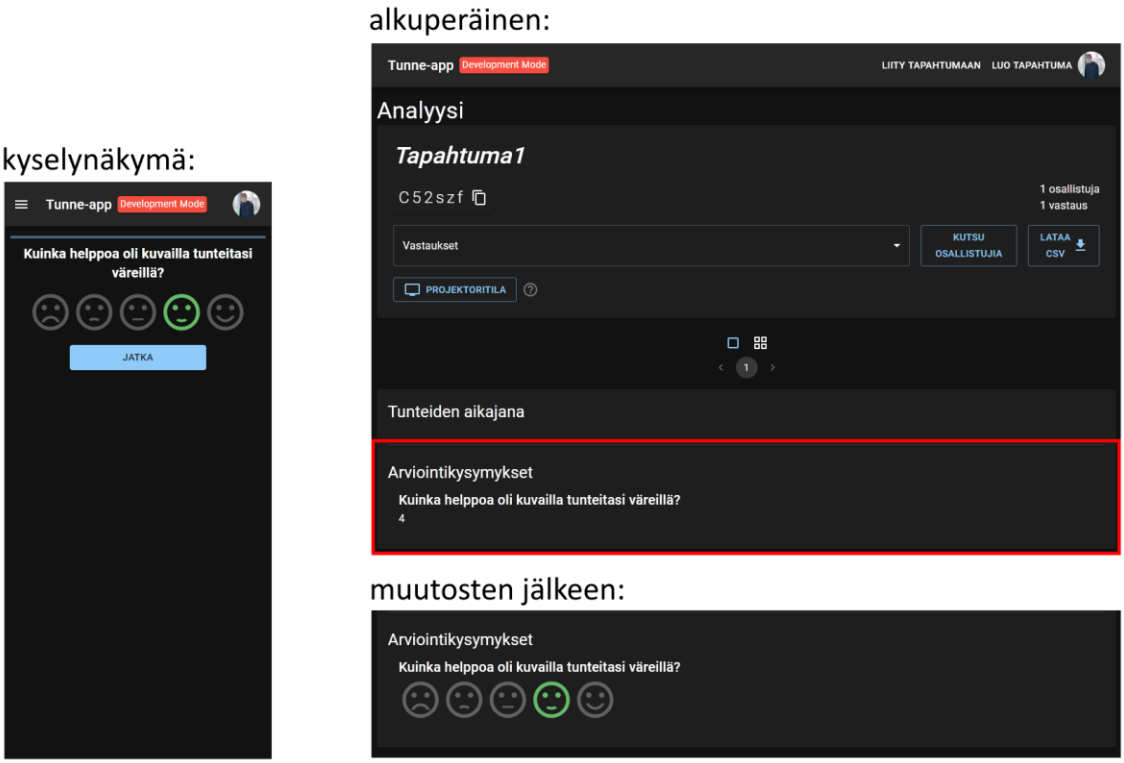
4.6.1 Koodin uudelleenkäytettävyys

Koodikannan suurentuessa sen ylläpidettävyys voi koitua vaivalloiseksi, kun tietokantakyselyt ja tyylitellyt komponentit täytyy päivittää useammasta paikasta muutoksia tehdessä. Projektin laajuus kasvoi alkuvaiheessa siihen pisteeseen, että oli oleellista tehdä ’välikerros’ komponenttien ja tietokantakyselyjen väliin. Käyttämällä Reactin context-koukkua välipalveluna funktiot logiikkoineen sijaitsevat vain yhdessä paikassa ja ne voidaan tuoda komponentteihin käytettäviksi (kuvio 12). Kuten kuvio 12 osoittaa, koodi on alun perin kahdessa paikassa, joka lisää työtä ja erehdyksien mahdollisuutta. Kontekstille tehdään tiedosto ’instances.jsx’. Kontekstin apufunktioita voidaan kutsua näissä kahdessa komponenteissa ja myös muualla. Näin voidaan myös varmistaa, että funktiot palauttavat tiedon aina samassa muodossa.



Kuvio 12. Kontekstin toteutus

Jotta tärkeät komponentit, kuten kyselypohjaan liittyvät kehonosien kysymys tai 1–5 asteikon arviointi näyttivät samoilta joka paikassa, oli oleellista antaa näille komponenteille mahdollisuus ’read only’-tilaan, jossa vastaukset ovat lukittuja. Antamalla esiasetetun arvon ja estämällä vastausten muokkauksen samoja komponentteja oli mahdollista käyttää yhteenveto- ja vastausten katselunäkymässä (kuvio 13).



Kuvio 13. Vastausten katselunäkymä ennen komponentin uudelleenkäyttämistä ja sen jälkeen

Sovelluksen käyttöliittymää kehitettäessä tultiin asiakkaan kanssa siihen tulokseen, että arviointi 1–5-asteikolla pelkkiä numeroita käyttäen oli liian teknisen näköinen. Hymynaamojen käyttäminen loi ”eloisamman” käyttökokemuksen. Ohjelman toiminnan kannalta tieto oli numeroarvona, mutta yhtenäisen ulkoasun luomiseksi oli syytä visualisoida vastaukset samanlaisesti molemmissa näkymissä.

4.6.2 Refaktorointi

Koodin uudelleenkäytettävyyden lisäksi oli oleellista pitää koodi mahdollisimman yksinkertaisena ja ymmärrettävänä. Koodin yksinkertaistamiseksi ja toistojen vähentämiseksi oli oleellista refaktoroida koodia. Refaktoroinnilla tarkoitetaan koodin uudelleenjärjestelyä ja yksinkertaistamista ilman, että koodin toiminnallisuus muuttuu (Fowler & Beck 1999). Refaktoroinnin avulla koodin ylläpidettävyys paranee ja koodin toiminnallisuus säilyy samana.

Käyttäjän kirjautumisen tila vaikuttaa näkyvillä oleviin elementteihin piilottaen elementtejä, jos käyttäjä ei ole kirjautunut sisään. Mikäli kirjautumaton tai väärä käyt-

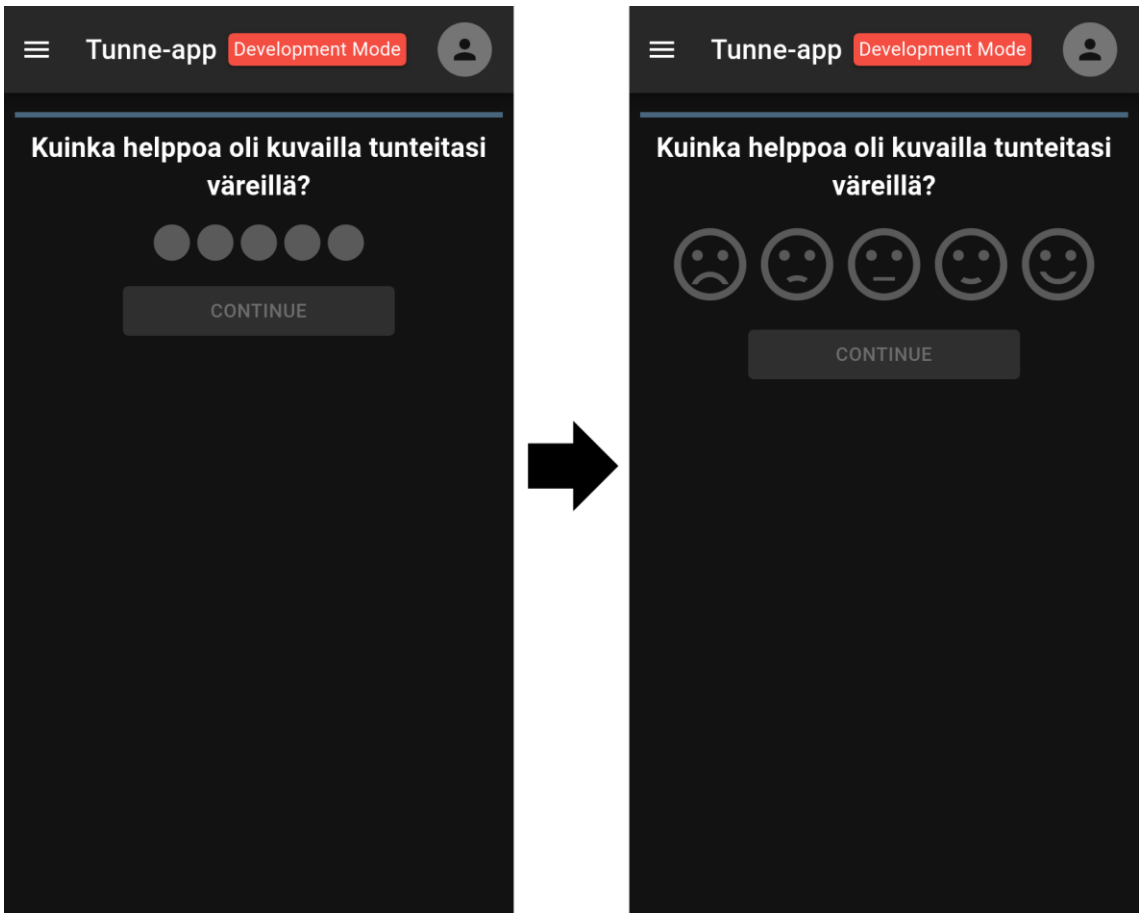
täjä yrittää tarkastella osallistujien vastauksia, voidaan välttyä peräkkäisiltä pyynnöiltä. Vaikka palvelin kieltäytyisi pyynnöstä, sovellus voi tehdä oman osansa estääkseen palvelimen tukkiutumista turhista pyynnöistä.

4.7 Julkinen testaaminen käyttäjien kanssa

Jatkuvan integroinnin ja toimituksen mukaisesti sovellusta testattiin kehityksen eri vaiheissa ja sovellukseen tehtiin muutoksia käyttäjäpalautteen perusteella. Asiakkaan kanssa etsittiin mahdollisia testauskumppaneita, jotka voisivat sopia sovelluksen visioon. Testauksella saataisiin tuloksia, jota Lapin yliopisto voisi näyttää hankkeen tukena jatkokehittämiselle. Kulttuuritalo Korundin kanssa viestien vaihtamisen tuloksena sovellusta päädyttiin testaamaan siten, että Korundin asiakkaita pyydettiin vastaamaan sovelluksen kysymyksiin ennen taidenäyttelyä ja sen jälkeen.

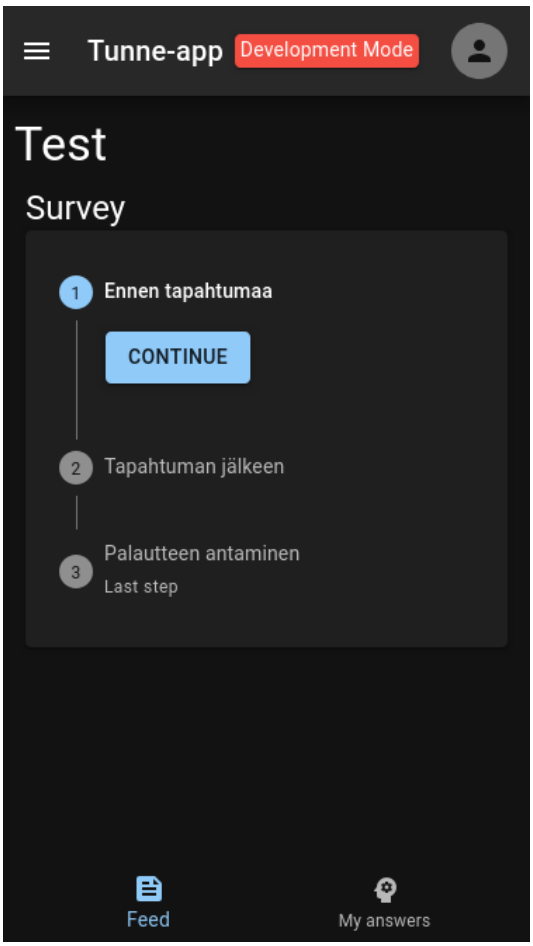
Testivaiheessa käyttäjät antoivat palautetta sovelluksen toiminnoista. Sovellukseen lisättiin kumoamispainike, sillä aiemmissa versiossa käyttäjät eivät voineet korjata, jos painoivat esimerkiksi väärää väriä tai kehonosaa. Kehittäjänä tällaiset jäävät huomioimatta, sillä jos värin valintaruutu tuli esille, täytyi vain klikata muualle, jotta pallon sijoittaminen peruuntui.

Kehittämisvaiheessa sovelluksen ulkoasu oli elottoman ja teknisen näköinen, sillä toiminnallisuus oli helppokäyttöisyyden edellä. Arviointipainikkeet muutettiin palloista hymiöiksi, koska testaamistilanteessa yksi testikäyttäjistä sanoi odottaneensa jonkin latautuvan, kunnes hän huomasi pallojen olevan 1–5-asteinen arviointipainike, kuten kuviossa 14 näkyy vasemmalla.



Kuvio 14. Arviointinäkymään tehdyt muutokset

Palautteen perusteella sivusta tehtiin vähemmän ”skrollattava”, koska käyttäjät luulivat, että sivun alaosasta jäi jotain näkemättä. Alaosasta pystyi kuitenkin tarkastelemaan pelkästään omia vastauksia. Ongelma ratkaistiin tekemällä näkymää vaihtavat painikkeet kuten kuvio 15 osoittaa. Jokainen näkymä oli tärkeä saada mahtumaan näytölle ilman, että näkymää täytyi vierittää alas.



Kuvio 15. Painikkeet näkymän vaihtamiseen

4.8 Sovelluksen käyttöönotto

Palvelinta päädyttiin ajamaan Viten preview-palvelimella Dockeria käyttäen. Tällaisen kokoisessa projektissa esikatseluun tarkoitettu palvelin toimii riittävän hyvin. Kehitys- ja esikatselupalvelimet eivät välttämättä ole turvassa haavoittuvaisuuksilta, joten tämä käytäntö ei sovi kaikkiin tilanteisiin.

4.8.1 Docker

Jotta palvelinta voi ajaa eri ympäristöissä ilman, että NodeJS on asennettu tietokoneelle, projektiin tuli ottaa käyttöön Docker. Docker on kontittaja, jonka avulla pystytään paketoimaan sovelluksia mahdollisimman saumattomasti eri ympäristöihin.

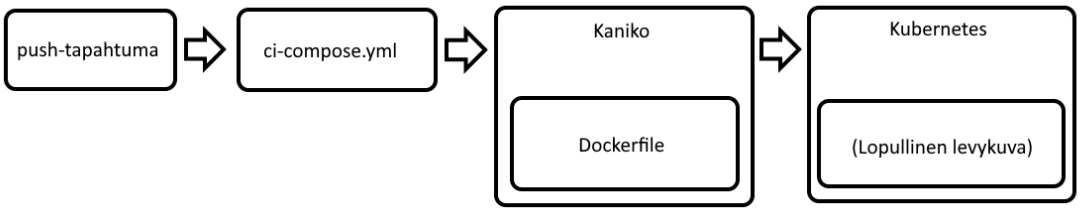
Kontit sisältävät kaikki tarvittavat riippuvuudet sovelluksen ajamiseksi, joten eri ympäristöistä ja niille asennettavista ulkoisista ohjelmista ei tarvitse huolehtia. Näin varmistetaan, että sovellus toimii samalla tavalla jokaisessa ympäristössä. Docker-kontit ovat kuin virtuaalikoneita, joilla voi ajaa monia eri palveluita yhdellä fyysisellä tietokoneella samanaikaisesti. (Docker 2024.)

4.8.2 GitLab CI/CD

Koska projekti luotiin GitLabia käyttäen, vaihtoehtona oli käyttää GitLabin sisäistä julkaisujärjestelmää, jossa voidaan tehdä automaattisia toimintoja eri laukaisukriteereillä. GitLabin CI/CD-pipelinellä voidaan esimerkiksi laukaista koodin rakennusvaihe, joka muuttaa ohjelmakoodin ajettavaan muotoon. Projektin yhteydessä käytettiin ”push”-tapahtumaa, joka tapahtuu, kun koodimuutokset siirretään versionhallintajärjestelmään (kuvio 16).

GitLabin CI/CD-pipeline konfiguroitiin ajamaan Docker-compose-tiedosto, joka kertoo Dockerille tarvittavat paketit NodeJS-sovellusten ajamiseen. Tähän konttiin kopioidaan projektin tiedostot ja ajetaan tarvittavat asennukset. Kaniko rakentaa Docker-compose tiedoston määrittelyn mukaisesti Dockerfile-tiedostosta levykuvan, joka on lopullinen ajettava palvelin (kuvio 16). Sovellusta voidaan ajaa levykuvaa käyttämällä esimerkiksi Kubernetes-järjestelmässä. Tämän jälkeen kontti käynnistetään ja palvelin on käyttövalmis. Tämä tapahtuu joka kerta, kun koodimuutos lähetetään versionhallintaan.

CI/CD-prosessi korvaa vanhan tuotantoversion, jos rakennusvaihe onnistuu ilman virheitä. Näin edellinen versio on vielä asiakkaiden käytettävissä, vaikka jokin virhe tapahtuu uuden version julkaisun yhteydessä.



Kuvio 16. Sovelluksen automatisoitu rakennusvaihe

4.8.3 Sijoituspaikka

Palvelinta ajettiin FrostBitin omalla GitLab-palvelimella koko kehityksen ajan, mikä mahdollisti tuotantoversion nopean päivittämisen testaustilaisuuksiin. Projektin vaatimuksena oli tuottaa ratkaisu, jota Lapin yliopisto voi käyttää omatoimisesti tai halutessaan maksaa FrostBitille ylläpidosta.

Projektin lopullinen tuote jaettiin GitLab-repositoryn kautta niin, että kohtuullisella Docker-tietotaidolla Docker-kontti voidaan sijoittaa mille tahansa VPS-palveluntarjoajalle, joka hyväksyy Docker-kontteja. Näin ollen käyttöympäristö voi olla joko fyysinen palvelin tai pilvipalvelin. Vaihtoehtoisesti ratkaisu voidaan ajaa paikallisella koneella Docker Desktopilla testausta varten.

5 POHDINTA

Tässä opinnäytetyössä käsiteltiin modernin web-sovelluksen tekemisessä sekä sen julkaisussa hyödynnettäviä teknologioita ja toteutuksia. Lisäksi opinnäytetyössä selvitettiin, miten luoda selkeä käyttöliittymä, helppo käyttäjäkokemus ja tutkittiin keinoja, joilla käyttäjä pystyy sekä visualisoimaan omia tunteitaan että ymmärtämään muutoksia tunnetilassa paremmin sovelluksen avulla. Kehitetty sovellus toimii työkaluna esimerkiksi museoiden, taidegallerioiden ja matkailuyritysten asiakaskokemusten aiheuttamien tunteiden mittaamisessa. Opinnäytetyö oli Lapin yliopiston hankkeen tilaus Lapin AMK:n FrostBit-laboratoriolta, joka toimi opinnäytetyön toimeksiantajana.

Opinnäytetyössä käytettiin monenlaista teknologiaa, mihin liittyvää teoriatietoa löytyi paljon ja helposti, mutta minkä toimivuus oli testattava käytännössä eri asiakasryhmille sovelluksen kehittämisvaiheissa. Testaustilaisuudet vahvistivat epäilyni siitä, että sovellusta ei kannattanut tehdä ladattavaksi. Eläkeikäiset testaajat jakoivat kokemuksia siitä, että erilaisten sovellusten lataaminen puhelimeen vaati Google- tai Apple-käyttäjän viimeistelyn, joka oli liian pitkä prosessi. Web-sovellus vähensi kynnystä halukkaiden testaajien saamiseksi.

Tämä herätti myös kysymyksiä siitä, miten omalla puhelimella käytettävään sovellukseen pyydetään navigoimaan testaustilanteessa. Käyttäjien on jotenkin päästävä sovellukseen omalla puhelimellaan, jos esittelypöydällä ei ole käytettävää laitetta. QR-koodi sovelluksen web-sivulle on yleinen käytäntö, ja se toimii lähes kaikissa puhelimissa, mutta muutamassa poikkeustilanteessa puhelimesta ei löytynyt QR-koodin lukijaa.

Sovelluksen kehittämisen alkuvaiheessa asiakas halusi sisältää AR- eli lisätyn todellisuuden ominaisuuksia, jonka avulla kameralla voitaisiin kuvata ryhmää ja henkilöistä ikään kuin kuplisi värillisiä palloja heidän vastaustansa mukaillen. Toteutusta esti henkilöiden tunnistuksen yhdistäminen vastauksiin. Nykyisillä AR-työkaluilla voidaan tunnistaa yksi henkilö ja saada palloja huokumaan ylöspäin hänen kehostaan.

Useamman henkilön seuranta on mahdollista, mutta vastausten yhdistäminen kuvassa oleviin henkilöihin vaatisi tiedon siitä, miltä he näyttävät. Ryhmää kuvatessa pitäisi tietää vastaajien kasvot, jotta kameran voisi suunnata keneen tahansa siten, että heidän tunnetilansa olisi heijastettu oikein. Ryhmän tunnetilan yleiseen kuvastamiseen teknologia voisi toimia ilman monimutkaisia ratkaisuja.

Dockerin käyttäminen projektissa lisäsi tietotaitoani etenkin Dockerin osalta. Ottaisin mukaan Dockerin mihin tahansa web-pohjaisen sovelluksen kehitykseen, koska se tekee palvelimen sisällyttämisestä itse sovelluksen kanssa vaivatonta. Sovelluksen lokalisointi oli myös kannattavaa, jos on esimerkiksi tekemässä sovellusta turistialalle. Koska sovelluksen back-end-palvelin oli täysin riippumaton käyttöliittymästä, Reactin valitseminen sovelluskehykseksi oli impulsiivinen valinta. Se ei kuitenkaan hidastanut kehitystä, jonka vuoksi se oli mielestäni hyvä päätös. Web-sovellusta kehittäessä voi olla vaikea tietää, mikä sovelluskehys on paras projektille ja se saattaa aiheuttaa esteitä kehityksen keskivaiheessa.

Projektin loppuvaiheessa puheeksi otettiin laajojen kielimallien käyttämisen mahdollisuudet. Vastaajille olisi mielenkiintoista antaa räätälöityjä kysymyksiä heidän jakamien tunteidensa perusteella. Myös pallojen sijaintien ja värien avulla sovellus voisi tarjota valmiita ehdotuksia vastaukseen siitä, mitä käyttäjä tarkoitti valitsemillaan sijainti- ja väriyhdistelmillä. Projektin into riitti loppuun asti, mutta projektin loppu oli niin lähellä, että lisäystä ei kannattanut tehdä. Tämän takia yrityksille tarjottavasta sovelluksesta puuttuu mielestäni joitain kätevyyttä ja saumattomuutta parantavia ominaisuuksia. Sovellus valmistui kuitenkin suunnitellussa aikataulussa ja täyttää sille asetut vaatimukset.

LÄHTEET

Björn, E., Miettinen, S., Jylkäs, T. & Sarantou, M. 2024. Effective analytical approaches as precursors for proof-of-concept generation. Teoksessa M. Sarantou & S. Miettinen (toim.) Empathic Service Design. Bloomsbury. Yksityinen arkisto.

Docker 2024. Get Started. What is Docker. Viitattu 9.10.2024 https://docs.docker.com/get-started/docker-overview/.

ESLint 2024. Getting Started. Viitattu 4.7.2024 https://eslint.org/docs/latest/use/getting-started.

Fowler, M. & Beck, K. 1999. Refactoring. Improving the design of existing code. Massachusetts: Addison-Wesley.

I18next 2022. Overview. Comparison to others. Viitattu 4.7.2024 https://www.i18next.com/overview/comparison-to-others.

Material Design 2024a. Accessibility. Viitattu 16.10.2024 https://m2.material.io/design/usability/accessibility.html#implementing-accessibility.

Material Design 2024b. Develop. Web. Viitattu 30.11.2024 https://m2.material.io/develop/web.

Mozilla 2024a. References. JavaScript. Reference. Standard built-in objects. JSON. Viitattu 3.10.2024 https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON.

Mozilla 2024b. References. Web APIs. Window. LocalStorage. Window: localStorage property. Viitattu 11.9.2024 https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage.

Nielsen & Norman Group 2024. 10 Usability Heuristics for User Interface Design. Viitattu 2.9.2024 https://www.nngroup.com/articles/ten-usability-heuristics.

React Router 2024. Feature Overview. Viitattu 3.7.2024 https://reactrouter.com/en/main/start/overview.

State of JS 2023. Front-end Frameworks Ratios Over Time. Viitattu 1.7.2024 https://2023.stateofjs.com/en-US/libraries/front-end-frameworks/.

Unicode 2024. Guidelines for Submitting Unicode® Emoji Proposals. Viitattu 9.9.2024 https://www.unicode.org/emoji/proposals.html.

Vite 2024. Guide. Why Vite. Viitattu 3.7.2024 https://vitejs.dev/guide/why.

Webpack 2024. Concepts. Why Webpack. Viitattu 3.7.2024 https://webpack.js.org/concepts/why-webpack/.