jamk

Mobiilisovellus React Native -kehyksellä käytettävyyden näkökulmasta

Taneli Manninen

Opinnäytetyö
Toukokuu 2024
Tieto- ja viestintätekniikan tutkinto-ohjelma (AMK)

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Manninen, Taneli

Mobiilisovellus React Native -kehyksellä käytettävyyden näkökulmasta.

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2024, 49 sivua.

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: Suomi

Julkaisulupa avoimessa verkossa: Kyllä

Tiivistelmä

Opinnäytetyön tarkoituksena oli tutkia käytettävyyttä React Native -kehyksellä toteutetun mobiilisovelluksen avulla. Tutkimuksen aikana keskityttiin sovelluksen kehittämiseen erilaisten suunnittelumenetelmien ja React Native -kehykselle yhteensopivien teknologioiden avulla.

Tutkimus toteutettiin soveltavasti ja perustui jo olemassa olevaan tietoon. Tietoperustaa kerättiin runsaasti sovelluksen arkkitehtuurin ja kehitysmenetelmien ymmärtämiseksi. Tietoa sovellettiin React Native -kehyksellä toteutetussa mobiilisovelluksessa, jonka avulla syntyi tutkimuksen tulokset.

Tutkimuksen tulokset osoittivat useita tekijöitä, jotka vaikuttavat mobiilisovelluksen käytettävyyden laatuun. Lisäksi saatiin tietoa siitä, miksi React Native -kehys on sopiva työkalu käytettävän mobiilisovelluksen kehittämisessä. Tuloksia voidaan hyödyntää tulevissa mobiilisovellusprojekteissa laadukkaan käytettävyyden varmistamiseksi ja React Native -kehyksen soveltuvuuden arvioimiseksi.

Yhteenvetona tutkimuksessa havainnollistettiin käytettävyyden varmistamista tarkkaan harkitun suunnittelun ja hyväksi todettujen menetelmien ja teknologioiden avulla.

Avainsanat (asiasanat)

Mobiilisovellus, sovelluskehitys, React Native, käytettävyys, käyttöliittymä, JavaScript, Node.js, GraphQL, MongoDB

jamk

Description

Manninen, Taneli

Usability perspective of a mobile application with React Native.

Jyväskylä: JAMK University of Applied Sciences, May 2024, 49 pages.

Degree Programme in Information and Communication Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

The purpose of the thesis work was to examine usability through a mobile application implemented using the React Native framework. The focus during the research was on developing the application using various design methods and technologies compatible with the React Native framework.

The research was conducted in an applied manner and was based on existing knowledge. Extensive information was gathered to understand the architecture and development methods of the application. This knowledge was then applied in the mobile application implemented with the React Native framework, resulting in the research findings.

The results of the research indicated several factors that affect the quality of usability in mobile applications. Additionally, insights were gained into why the React Native framework is a suitable tool for developing mobile applications. These findings can be utilized in the future mobile application projects to ensure high-quality usability and to evaluate the suitability of the React Native framework.

In summary, the research illustrated the assurance of usability through careful design and proven methods and technologies.

Keywords/tags (subjects)

Mobile application, application development, React Native, usability, user interface, JavaScript, Node.js, GraphQL, MongoDB

Sisältö

Kuviot

Taulukot

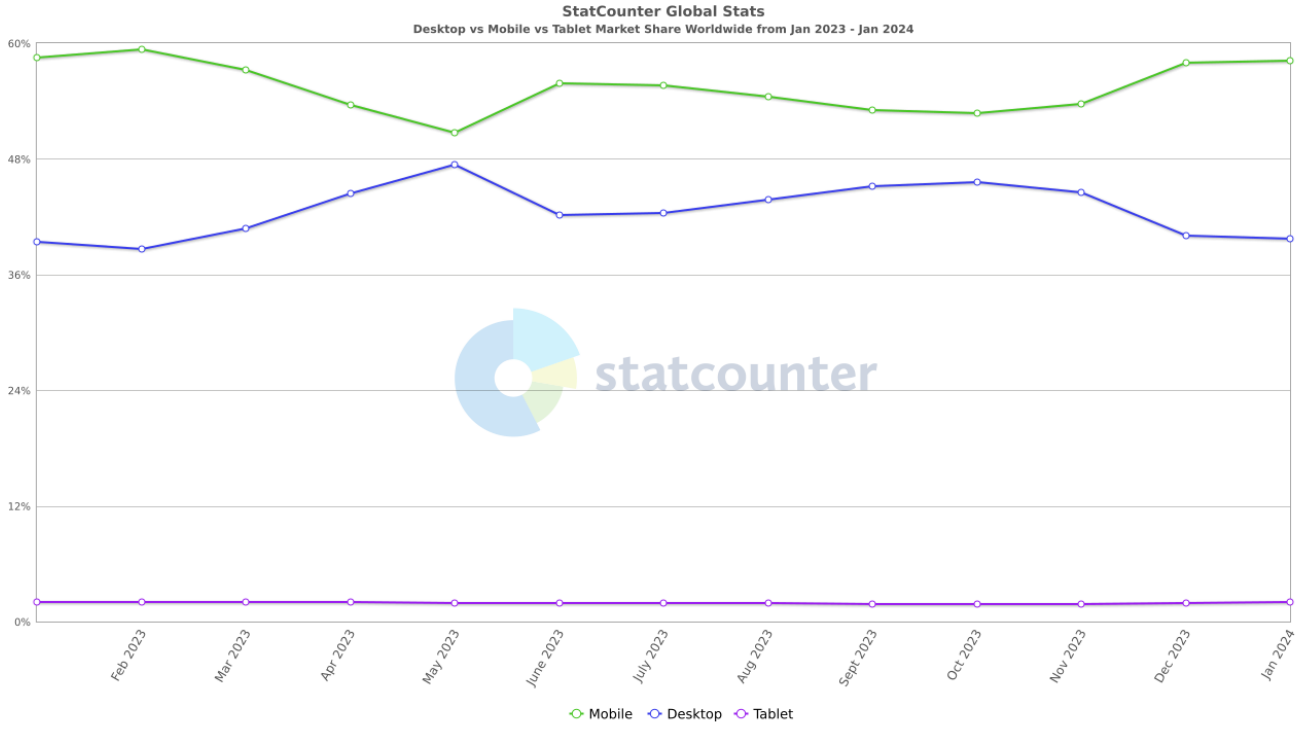
No table of figures entries found.

1 Johdanto

1.1 Sovelluskehityksen kehittyminen

Sovelluskehitys sai käytännössä alkunsa sinä hetkenä, kun tietokone keksittiin. Ensimmäiset sovellukset olivat siis luonnostaan tarkoitettu pöytäkoneille, koska muita alustoja ei vielä ollut olemassa. Lisäksi niiden käyttö oli suurimmaksi osaksi yritysten tiloista ja tietoverkoista riippuvaisia. Internetin laajennettua globaaliksi ilmiöksi, myös sen julkinen saatavuus helpottui ja monipuolistui. Selaimet olivat nyt kuluttajan saatavilla ja uudet käyttöalustat, kuten mobiililaitteet avasivat uusia mahdollisuuksia sovelluksille. (What's the Difference Between Web Apps, Native Apps, and Hybrid Apps? N.d.)

Tänä päivänä mobiililaitteiden käyttö on maailmanlaajuisesti ylittänyt pöytäkoneiden käytön. Kuviossa 1 havainnollistetaan mobiililaitteiden, tietokoneiden, sekä tablettien markkinaosuuksia vuoden 2023 tammikuusta vuoden 2024 tammikuuhun.



Kuvio 1 Desktop vs Mobile vs Tablet Market Share Worldwide (StatCounter 2024.)

1.2 Mobiilisovelluksien alustat

Mobiilisovellukset jaetaan yleensä web-, natiivi-, sekä hybridisovelluksiin (Hanna & Wigmore 2023). Websovelluksia käytetään tunnetusti selaimella ja ne ovat saatavilla pöytäkoneille, sekä mobiililaitteille. Natiivisovellukset ovat puolestaan täysin rakennettu mobiililaitteita varten ja niiden käyttöönotto vaatii yleensä lataustoimenpiteen sovelluskaupasta käsin. Hybridisovellukset yhdistävät ominaisuuksia web-, sekä natiivisovelluksista. (What's the Difference Between Web Apps, Native Apps, and Hybrid Apps? N.d.)

Natiivisovelluksien vahvuuksina nähdään parempi käyttäjäkokemus ja tehokkuus. Natiivisovellus on mobiiliskaalaan optimoitu, mahdollistaa notifikaatioiden käytön viestinnässä käyttäjän kanssa, sekä on nopea ja responsiivinen käyttää. Heikkoutena voidaan nähdä, että natiivisovelluksen kehitys on kalliimpi ja monimutkaisempi vaihtoehto. Lisäksi natiivisovellus vaatii käyttäjältä ajoittaista sovelluksen päivittämistä. (What's the Difference Between Web Apps, Native Apps, and Hybrid Apps? N.d.)

Websovellukset puolestaan ovat vaihtoehtona halvempia, nopeampia, sekä yksinkertaisempia kehittää. Websovellus kärsii kuitenkin laadussa käytettävyyden näkökulmasta, kun se on riippuvainen selaimesta. (What's the Difference Between Web Apps, Native Apps, and Hybrid Apps? N.d.)

1.3 Käytettävyys

Yksi olennaisimmista yhtä aikaa toimivan, sekä laadukkaan sovelluksen tekijöistä on sen käytettävyys. Käytettävyyttä ajatellessa mietitään yhteyttä sovelluksen ja käyttäjän välillä. Ideana on, että sovelluksen tulisi olla helppokäyttöinen, tehokas, sekä selkeälukuinen. Lisäksi viestintä käyttäjän kanssa tulee toimia, jos hän esimerkiksi tekee virheen. (Käytettävyys ohjelmistokehityksessä: Miksi ja miten sitä tutkitaan? 2023.)

Käytettävyydeltään toimivan sovelluksen merkki on myös se, että käyttäjä onnistuu siinä mitä tuli sovellukseen tekemään. Onnistumisen tunnetta haetaan, koska se jättää käyttäjälle yleensä hyvän mielen. Kehittäjän tehtävä on mahdollistaa tämä toimivalla sovelluslogiikalla ja yrittää samalla tehdä toiminnallisuudet mahdollisimman selkeäksi käyttäjälle. (Smith 2017.)

Sovelluksen käytettävyyttä testataan yleensä käyttäjätestauksella, missä käyttäjät testaavat sovelluksen eri toiminnallisuuksia käytännössä. Näin pyritään havaitsemaan jo kehitysvaiheessa käyttäjän tarpeet, sekä mahdolliset ongelmakohdat. (Käytettävyys ohjelmistokehityksessä: Miksi ja miten sitä tutkitaan 2023.)

1.4 Tutkimusasetelma

Tässä luvussa käsitellään opinnäytetyön lähtökohtia, tutkimustyön tavoitteita ja tarkoituksia, sekä muita tutkimukseen liittyviä yksityiskohtia. Lisäksi esitellään opinnäytetyölle asetetut tutkimuskysymykset.

Tutkimustyön tarkoituksena oli keskittyä natiivimobiilisovelluksen suunnittelun ja ohjelmoinnin kautta käytettävyyden tutkimiseen. Sovelluksen vaatimusmäärittelyksi asetettiin laadukkaasti toteutetut käyttöliittymät, joiden välille tuli toteuttaa looginen ja tehokas navigointi. Lisäksi sovelluksen taustalle tuli rakentaa sovelluslogiikkaa tukeva palvelinteknologia ja tietokanta. Sovelluksen toteutuksessa oli tarkoitus käyttää tavoitteita tukevia ja hyödylliseksi todistettuja teknologioita, sekä kehitysmenetelmiä. Kehittäjän tavoitteena oli myös kartoittaa ja soveltaa mahdollisimman paljon tietoa tutkimustyön aikana, ja kehittyä sitä kautta sovelluskehittäjänä, sekä käyttöliittymien suunnittelijana.

Tutkimuksessa käytettiin soveltavaa tutkimusmenetelmää, kun kyseessä on valmiiseen tietoon ja teknologiaan perustuva käytännön tutkimustyö. Tutkimuksen aikana pyrittiin ensisijaisesti vastaamaan seuraaviin tutkimuskysymyksiin:

- Miten luoda käytettävyydeltään laadukas mobiilisovellus?
- Mitkä tekijät vaikuttavat käytettävyyteen sovelluskokonaisuudessa?
- Miksi React Native on tähän tarkoitukseen sopiva teknologia?

Tavoitteena oli luoda natiivialustalla toimiva mobiilisovellus, joka osoittaisi laadukasta käytettävyyttä sen toimivuudessa, tehokkuudessa, ulkoasussa, sekä viestinnässä käyttäjän kanssa. Kehitystyön kohteeksi muodostui keskustelu-sovellus, jonka projektinimeksi annettiin ”Chat-App”. Sovelluskokonaisuus rajattiin kolmeen ennalta määritettyyn käyttäjäpolkuun, jotka perustuivat kaltaiselleen mobiilisovellukselle tyypillisiin toiminnallisuuksiin.

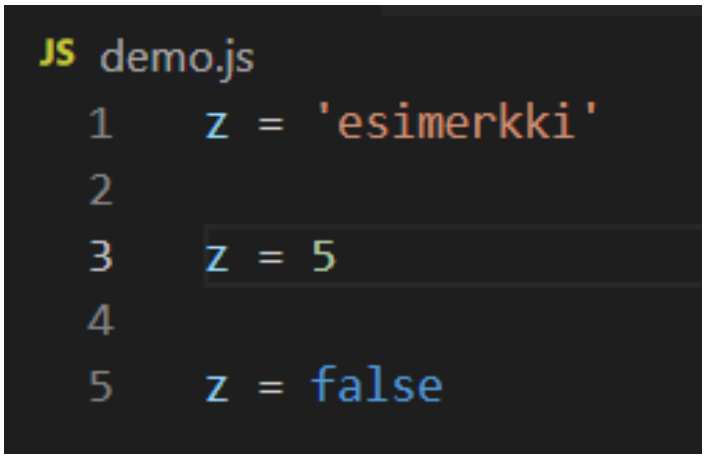
Opinnäytetyö suunniteltiin, toteutettiin ja dokumentoitiin täysin itsenäisesti. Tästä syystä tutkimustyön kehittäjä toimi itse sen toimeksiantajana.

2 Käytettävissä olevat teknologiat

2.1 JavaScript-ohjelmointikieli

JavaScript on laajasti web- ja mobiilikehityksessä käytetty oliopohjainen ohjelmointikieli. Se on tunnettu parhaiten käytöstään web-sivujen parissa, mutta nykyään JavaScriptiä käytetään myös paljon web-selaimesta riippumattomissa ympäristöissä. (JavaScript n.d.)

JavaScriptin toimintaa voidaan kuvailla prototyyppipohjaiseksi, yksisäkeiseksi, sekä dynaamiseksi ohjelmointikieleksi. Prototyyppipohjaisuus on olio-ohjelmointia, jossa luokkia johdetaan lisäämällä metodeja ja ominaisuuksia toisen luokan instanssiin (Prototype-based programming n.d). Yksisäkeisyydellä tarkoitetaan ohjelmointikieltä, joka pystyy suorittamaan vain yhden tehtävän kerrallaan, ketjumaisessa järjestyksessä. Uutta tehtävää ei koskaan aloiteta ennen kuin edellinen on valmis. (Long 2023.) Dynaamisella kielellä tarkoitetaan sitä, että ohjelman ajoaikana sen tyyppejä, sekä arvoja voidaan molempia halutessaan vaihdella (Bashir n.d). Kuviossa 2 havainnollistetaan JavaScript-ohjelmointikielen dynaamisuutta.



Kuvio 2 Esimerkki JavaScriptin dynaamisuudesta

Sekä Long (2023), Bashir (n.d) että Prototype-based programming (n.d) kuvaavat JavaScriptin tyyliä ja toiminnallisuutta ohjelmointikielenä (JavaScript n.d). Voidaan todeta, että JavaScript-ohjelmointikielen toimintalogiikka pohjautuu moneen eri tekijään ja ne ovat kaikki otettava huomioon sitä käyttäessä.

Seuraavaksi tarkastellaan JavaScript-ohjelmointikielen kehityksen eri vaiheita. JavaScript oli alkujaan vain sisäiseen käyttöön tarkoitettu yleiskäyttöinen skriptikieli, jonka tarkoituksena oli varmistaa verkkosivujen yhteentoimivuus selainten ja laitteiden välillä. Mozilla Firefoxin ja Google Chromen myötä JavaScript on jatkanut kasvavaa kehitystään. Tänä päivänä JavaScriptin käyttöön on jo tarjolla lukuisia eri kehyksiä ja kirjastoja helpottamaan suuria projektikokonaisuuksia. JavaScriptin tarjoamat koodikirjastot mahdollistavat aikaa ja työtä säästävän valmiin koodin käytön. Sovelluksen käyttäjäpäässä, eli frontend-puolella JavaScriptin kehyksistä päällimmäisinä tarjolla ovat AngularJS, jQuery, sekä React.js. (Jordana 2024.)

2.2 React-kirjastot

2.2.1 React-kirjasto

React.js on Facebookin kehittämä avoin koodikirjasto JavaScriptille. Kyseinen kirjasto on tarkoitettu käyttöliittymien rakentamiseen uudestaan käytettävien komponenttien avulla. React-kirjasto tarjoaa erilaisiin käyttöliittymän tarpeisiin useita valmiita elementtejä, kuten nappeja, listoja ja navigaatiotyökaluja. Teknologian peruspilareina ovat komponentit, propsit ja state hookit. Nämä peruspilarit mahdollistavat monimutkaistenkin sovelluskokonaisuuksien rakentamisen. (Overview of React.js n.d.)

React-sovelluksen käyttöliittymät rakentuvat komponenteista. Niiden avulla käyttöliittymä jakautuu omiin uudestaan käytettäviin osiin, jotka näkyvät käyttäjälle ruudulla. Propsit ovat taas komponenttien sisään liitettyä dataa, jota on mahdollista liikutella eri komponenttien välillä. Kuviossa 3 on nähtävissä propsien kutsuminen React-komponentissa. State-hook on puolestaan työkalu komponentin tilanhallintaan, joka mahdollistaa tilan muuttumisen sovelluksen elinaikana. (Overview of React.js n.d.) Hookit ovat funkioita, jotka toimivat ainoastaan React-sovelluksen renderöidessä, eli päivittäessä tilaansa (State: A Component’s Memory. Meet your first hook n.d.).

```
const Button = ({ route, text, color }) => {
    const dynamicStyles = color === "blue" ? styles.backgroundBlue : styles.backgroundPersimmon;

    return (
        <Pressable style={[styles.buttonContainer, dynamicStyles]}>
            <Link to={route}>
                <Text fontSize='subheading'>{text}</Text>
            </Link>
        </Pressable>
    );
};

export default Button;
```

Kuvio 3 Esimerkki propseja sisältävästä, uudelleen käytettävästä Button-komponentista

React-sovelluksissa hyödynnetään usein JSX-syntaksia, joka yhdistää HTML- ja JavaScript-koodia. Tarkoituksena on koodin luettavuuden, sekä oppimisen helpottaminen. Selaimet eivät kuitenkaan JSX-syntaksia tue, joten sen tulkkaamiseen tarvitaan Babel-kääntäjää. (React Introduction n.d.) JSX-syntaksin käyttö ei kuitenkaan Reactin kanssa ole pakollista (Introducing JSX. Why JSX? n.d.).

2.2.2 React Native -kirjasto

React Native -kirjasto on JavaScriptia ja Reactia yhdistävä työkalu, joka on tarkoitettu Android-, sekä iOS-käyttöjärjestelmän natiivisovelluksien kehittämiseen. Tärkeinpänä hyötynä React Nativessa on React-kirjastojen tapaan kehitystyön nopeus ja käytön helppous. React Native hyödyntää natiivisovelluksen kehityksessä React-kirjastosta tuttuja ominaisuuksia, kuten komponentteja, propseja, state hookeja, sekä JSX-syntaksia. (Introduction to React Native n.d.)

Facebookin kehittämä vuonna 2015 julkaistu React Native ponnahti vain muutamassa vuodessa mobiilisovelluskehityksen kärkinimiksi. Tänä päivänä sen teknologiaan pohjautuu esimerkiksi Instagram, Facebook, sekä Skype. Suuren suosion takana pidetään ainakin seuraavia seikkoja. Ensimmäiseksi React Native mahdollistaa saman koodikannan käytön useilla eri käyttöjärjestelmillä samanaikaisesti, mikä säästää huomattavasti resursseja. Toiseksi, se pohjautuu jo valmiiksi suosittuun Reactin JavaScript-kirjastoon. Kolmanneksi, React Native mahdollisti web-pohjaisista sovelluksista riippuvuuden poistamisen ja sitä kautta frontend-mobiilikehityksen mullistamisen. (Budzinski n.d.)

Borozenetsin (2022) mukaan yksi hyvä vaihtoehto React Native -sovelluksen rakentamisen avuksi on Expo CLI, eli komentoliittymä. Expo CLI tarjoaa työkaluja helpottamaan sovelluksen luomista ja testausta. (Borozenets 2022.) Näistä päällimmäisenä apuna kehittäjälle on Expo Go -puhelinsovellus, joka on ladattavissa Google Playsta ja App Storesta. Kehittäjälle Expo Go mahdollistaa sovellusprojektin avaamisen kehitysvaiheessa omalla älypuhelimella. Kun sovellus käynnistetään terminaalissa, tulee näkyville QR-koodi. Koodin voi sen jälkeen lukea Iphonen omalla kameralla, tai Androidilla Expo Go -sovelluksen kautta, jonka jälkeen puhelin avaa kehitettävän sovelluksen automaattisesti Expo Go -sovelluksessa. (Patrun 2023.) Sekä Borozenets että Patrun (2023) korostavat Expo CLI -työkalun hyötyjä ja painottavat sen helpottavaa vaikutusta natiivisovelluskehityksessä.

2.2.3 React Router -kirjasto

Julkaisuvuodestaan 2013 lähtien React Router -kirjasto on ollut dominoiva reititysteknologia sovelluskehityksessä (Why Should You Use a Router in React.j? n.d). React Router mahdollistaa reitityksien luomisen, hallinnan, sekä reittien välisen navigoinnin React-sovelluksissa. Tarkoituksena on, että käyttäjän olisi helppo navigoida sovelluksessa eri näkymien välillä. (Shaban 2023.)

Navigointi React Router -kirjaston kanssa perustuu siihen, että sovelluksen URL-reitit ovat liitettyinä tiettyihin käyttöliittymien näkymiin, jotka ovat React-komponentteja. React Router mahdollistaa URL-reittien vaihtelun sovelluksen ajoaikana, jotta oikea näkymä saadaan aina renderöityä ruudulle käyttäjän tarpeen mukaan. (React Router n.d.)

React Router -kirjaston käyttöön liittyy muutama keskeinen komponentti. Nämä komponentit ovat nimeltään Route, Routes, sekä Router. Route-komponentti perustuu ehdollisuuteen. (React Router n.d.) Se sisältää yleensä path-, sekä element-kentät. Path-kenttä sisältää URL-reitin, joka on yhdistetty aina haluttuun komponenttiin. Element-kenttä sisältää reitityksen osoittaman komponentin. Routes on komponentti, mihin Route-komponentit listamaisesti sijoitetaan. Reitityksen infrastruktuurista vastaa Router-komponentti, joka toimii pääkomponenttina. (Shaban 2023.) Kuviossa 4 havainnollistetaan Route-komponenttien käyttöä sovelluksen reitityksessä.

```
return (
  <View style={styles.container}>
    <Routes>
      <Route path='/' element={<Feed setToken={setToken} />} />
      <Route path='/new-post' element={<NewPost />} />
      <Route path='/post/:id' element={<SinglePostView />} />
    </Routes>
  </View>
);
```

Kuvio 4 Sovelluksen Route-komponentit

React Router -kirjasto toimii myös React Native -sovelluksissa (Michael 2023). Remix-nimisen tiimin kehittämä React Router Native perustuu samaan koodikantaan, kuten React Router -kirjasto. Tästä syystä React Router -kirjastoon tutustuneille, web-kehittäjätaustaisille käyttäjille on varsin helppoa ja suositeltavaa käyttää React Native -sovelluksiin tarkoitettua React Router Native -kirjastoa. (Yusufu 2024.) React Router Native tarjoaa omia React Native -sovelluksen reititykseen tarkoitettuja komponentteja, joista päällimmäisenä NativeRouter-komponentti alustamaan sovelluksen infrastruktuuri (Nasir n.d).

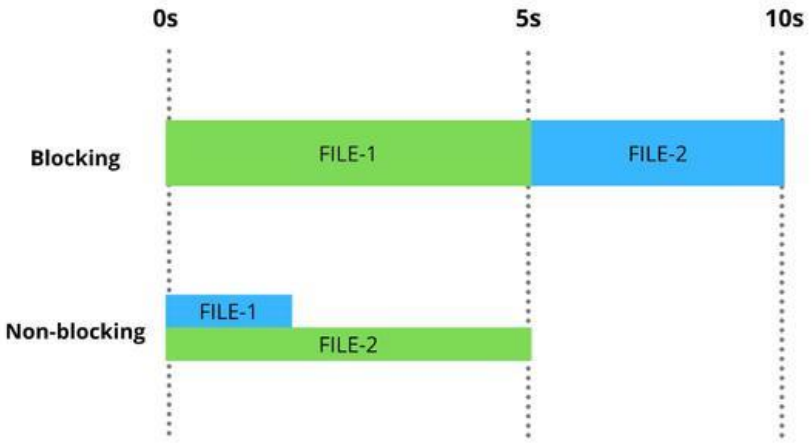
2.3 Palvelinteknologiat

2.3.1 Node.js-ajoympäristö

Node.js on JavaScript-ajoympäristö, joka pohjautuu avoimeen lähdekoodiin (Introduction to Node.js. n.d). Avoimuudella tarkoitetaan sitä, että lähdekoodi on julkista ja sitä ylläpitää kehittäjät ympäri maailman. Lisäksi Node.js on monialustainen, kun se ei ole riippuvainen tietystä käyttöjärjestelmästä. Se toimii kaikillä merkittävillä järjestelmillä, kuten Windowsilla, macOSilla ja Linuxilla. (Semah 2022.)

Ennen Node.js-ajoympäristöä JavaScriptin ajaminen oli selaimista riippuvaista. Lisäksi JavaScriptiä oli mahdollista käyttää vain käyttöliittymien ohjelmointiin. Saavuttuaan Node.js on antanut JavaScriptille vaihtoehtoisen ajoympäristön, sekä mahdollisuuden käyttää JavaScriptiä myös palvelinpuolen, eli backendin ohjelmointiin. (Semah 2022.)

Node.js-ajoympäristön toiminnallisuus perustuu asynkronisuuteen sen yksisäikeisyyden, sekä tapahtumapohjaisen arkkitehtuurin takia. Asynkronisessa ohjelmoinnissa mahdollistetaan useiden samanaikaisten toimintojen suorittaminen ilman, että toisen valmistumista täytyisi odotella. Tätä kautta Node.js pystyy suorittamaan tehokkaasti useita operaatioita samanaikaisesti. (Acharya 2023). Tehokkuutta kuvataan käsitteellä non-blocking I/O operaatiot, kun ajoympäristön pääsäettä ei hidasteta tai tukita (Dziuba 2023). I/O-lyhenne tarkoittaa sanoja input/output ja kuvaa yleisesti kahden eri ohjelman tai laitteen välistä datan siirtoa (Gallo 2023). Non-blocking operaatiota kuvataan kuviossa 5.



Kuvio 5 Blocking vs Non-blocking operation (How Node.js overcome the problem of blocking of I/O operations n.d.)

Node.js-ajoympäristön hyötyinä nähdään sen helppokäyttöisyys. Aloittelijalle Node.js on hyvä teknologia käyttöönotettavaksi, koska sen yhteisö ja dokumentaatio ovat laajoja. Hyötyinä nähdään myös ajoympäristön laajamittainen skaalautuminen, runsas valikoima avoimen lähdekoodin paketteja, sekä vahva palvelintoiminta ja monialustaisuus. Monialustaisuudella viitataan siihen, että Node.js on käytettävissä kaikkien sovellusmuotojen, kuten nettisivujen, työpöytä-, sekä mobiilisovelluksien luonnissa. (What Is Node.js and Why You Should Use It n.d.)

Komentorivityökalu Node.js-ajoympäristön hallintaan on nimeltään NPM, joka on lyhenne sanoista Node Package Manager. NPM-työkalun avulla on mahdollista asentaa eri Node.js-paketteja, sekä

suorittaa versionhallintaa. (Node.js – NPM n.d.) NPM on suurin ilmainen avoimen lähdekoodin kirjastojen ekosysteemi. Se latautuu automaattisesti Node.js-ajoympäristön latauksen yhteydessä ja sen lukuisat paketit on laajasti internetissä dokumentoituna. Pakettien lataus tapahtuu lokaalisti yksittäisillä komennoilla. (What Is Node.js and Why You Should Use It n.d.)

2.3.2 GraphQL-palvelin

GraphQL on ohjelmointirajapinnoille, eli API:lle tarkoitettu kyselykieli ja palvelinpuolen ajoaika (What is GraphQL 2019). Facebook kehitti sen ensin vuonna 2012, minkä jälkeen julkaisi sen avoimen lähdekoodin muodossa vuonna 2015. GraphQL-kyselykieltä käytetään palvelinpuolen kommunikoinnissa, kuten datan hakemisessa ja muokkaamisessa. (Alam-Naylor & Fateh 2021.) GraphQL on tuettu käyttäjänsä koodilla ja datalla, eikä ole tästä syystä sidonnainen mihinkään tiettyyn tietokantaan tai tallennusmoottoriin. (Introduction to GraphQL n.d.) Sitä voi käyttää käytännössä kaikkien ohjelmointikielien ja viitekehyksien yhteydessä, mukaan lukien myös käyttöliittymäpuolen ohjelmoinnissa (Alam-Naylor & Fateh 2021).

Seuraavaksi kerrotaan hieman GraphQL-teknologian keskeisimmistä käsitteistä. Kaikki GraphQL-palvelimella käsiteltävä data kuvaillaan schemassa (What is GraphQL 2019). Schema rakentuu type-objekteista, joiden sisältämät kentät määrittävät palvelimella saatavilla olevan datan. Type-objektien lisäksi schema yleensä sisältää myös ainakin yhden datan hakemiseen tarkoitetun query-objektin, sekä tarvittaessa mutation-typejä datan luomiseen. (Schemas and Types n.d.) Lisäksi jokaista scheman sisältämää kenttää kohtaan liitetään resolveri-funktio. Resolveri vastaa datan hakemisen ja käsittelyn funkionaalisuudesta. (Abbas 2023.) Kuvio 6 osoittaa esimerkin typen määrittämisestä tavallisen objektin, sekä kyselyn muodossa.

```
type User {
  firstname: String!
  lastname: String!
  username: String!
  image: String
  id: ID!
}

type Query {
  allUsers: [User!]!
}
```

Kuvio 6 Esimerkit scheman type-objekteista

2.3.3 Apollo-kirjastot

GraphQL-kyselykielellä toteutetun palvelimen yhteyteen suositellaan sen kaikki standardit täyttävää Apollo Server -palvelinkirjastoa. Apollo Server on JavaScriptillä kirjoitettu, avoimen lähdekoodin palvelinkirjasto, joka on tarkoitettu GraphQL-kyselykielellä toteutetun API:n rakentamiseen ja käynnistämiseen. (Bolat 2020.) Apollo Server -kirjaston hyötyinä nähdään sen asennuksen suoraviivaisuus, käyttöönoton vaiheittaisuus, sekä tuotantovalmius. Lisäosana se on yhteensopiva kaikentyyppisten Node.js-sovelluksien kanssa. (Introduction to Apollo Server n.d.)

Serverin lisäksi Apollo tarjoaa myös Apollo Client -kirjaston, joka käsittelee API:n käytön sovelluksen käyttäjäpuolella, eli frontendissä (Raji 2022). Apollo Client -kirjasto mahdollistaa datan hakemisen, välimuistamisen, muokkauksen, sekä käyttöliittymän päivityksen automatisoinnin. Erityisesti Apollo Client on yhteensopiva React-kirjaston kanssa, koska sen ydinkirjasto sisältää Reactille käyttövalmiin integraation. (Introduction to Apollo Client n.d.)

Kolmantena hyödyllisenä Apollon työkaluna tarjolla on Apollo Studio Explorer, joka tunnettiin aikaisemmin nimellä Apollo Graph Manager. Apollo Studio Explorer -työkalua voi käyttää esimerkiksi kyselyjen ajamiseen palvelimella (GraphQL-palvelin n.d). Hallintapaneelina Apollo

Studio Explorer on hyödyllinen monitori kehitysvaiheessa, kun halutaan testata GraphQL-palvelimen eri toimintoja käytännössä (Bulat 2020).

2.3.4 MongoDB-tietokanta

MongoDB on dokumenttipohjainen tietokanta, joka tallentaa joustavasti dataa JSON-tiedostojen mukaisesti (What Is MongoDB? n.d). JSON tulee sanoista JavaScript Object Notation, ja on 2000-luvun alussa kehitetty tiedostomuoto. JSON perustuu tiedon siirtoon ihmisluettavassa muodossa. (JSON and BSON n.d.) Tallennuksen joustavuudella tarkoitetaan sitä, että dokumentit voivat MongoDB-tietokannassa erota toisistaan ja niitä on aina mahdollista muuttaa. (What Is MongoDB? n.d.) MongoDB-tietokanta toimii tehokkaasti suurienkin tietomäärien tallentamiseen ja on tästä syystä isojen organisaatioden suosiossa (Botelho & Gillis n.d).

Data rakentuu MongoDB-tietokannassa dokumenteista, ja dokumenttien joukot muodostavat kokoelmia. Dokumentit koostuvat kenttä- ja arvopareista. Kentät sisältävät käyttäjän tallentaman tiedon. JSON-tiedostojen mukailevaisuus tulee MongoDB-dokumenttien käyttämästä BSON-rakenteesta, joka tulee sanoista Binary JSON. (Botelho & Gillis n.d.) BSON-rakenteen binäärisyys mahdollistaa tiedon läpikäynnin nopeammin verrattuna JSON-rakenteeseen (JSON and BSON n.d). Kuviossa 7 havainnollistetaan MongoDB-tietokannassa sijaitsevaa dokumenttia.

Kuvio 7 Esimerkki MongoDB-dokumentista

MongoDB-tietokannan käyttöönotto on mahdollista paikallisesti asennettavalla ohjelmistolla, mutta varteenotettavana vaihtoehtona on myös tarjolla selaimella ajettava MongoDB Atlas (Tietojen tallettaminen MongoDB-tietokantaan n.d). MongoDB Atlas on käyttäjän valitsemalla

pilvipalveluntarjoalla toimiva pilvitietokanta. Atlas tarjoaa yksinkertaistetun tavan rakentaa, skaalata, sekä ajaa MongoDB-tietokantaa pilvessä. (MongoDB Atlas Tutorial n.d.)

2.3.5 Mongoose-kirjasto

Mongoose-kirjasto on Node.js-teknologiaa käyttävien kehittäjien hallitsevasti suosima ODM-kirjasto MongoDB-tietokannan käyttöön (Hall 2022). ODM-lyhenne rakentuu sanoista Object Data Model. Terminä Object Data Model tarkoittaa objektimuotoisen datan tallennuksen prosessia. Käytännössä ODM helpottaa kehittäjien työtä, kun dataa haetaan tietokannoista. (Object Data Model n.d.)

Mongoose-kirjastoa käytetään tietojen mallintamiseen, mallien validoimiseen ja yleiseen tietojen käsittelyyn. Sovellustasolla sen käyttöönottoon vaaditaan schema-, sekä model-objektin määrittelyt. (Hall 2022.) Schema-objekti määrittelee dokumenttien rakenteen kokoelmissa ja kokoelmat peilataan MongoDB-tietokannassa vastaaviin kokoelmien määrittelyihin. Schema-objekti täytyy liittää model-objektiin, joka huolehtii vuorovaikutuksen kokoelman kanssa MongoDB-tietokannassa. Model-objekti mahdollistaa kyselyjen, kuten datan hakemisen, päivittämisen, sekä poistamisen ajamisen. (Kukic & Vlaeva 2021.) Model-objektin määrittelyä schema-objektin avulla kuvataan kuviossa 8.

```
const schema = new mongoose.Schema({
    username: {
        type: String,
        required: true,
        unique: true,
    },
    passwordHash: {
        type: String,
        required: true
    }
});

const User = mongoose.model('User', schema);
```

Kuvio 8 Esimerkki Mongoosen Schema- ja Model-objektien määrittelystä

3 Sovelluksen toteutus

3.1 Sovelluksen suunnittelu

3.1.1 Käyttäjäpolut

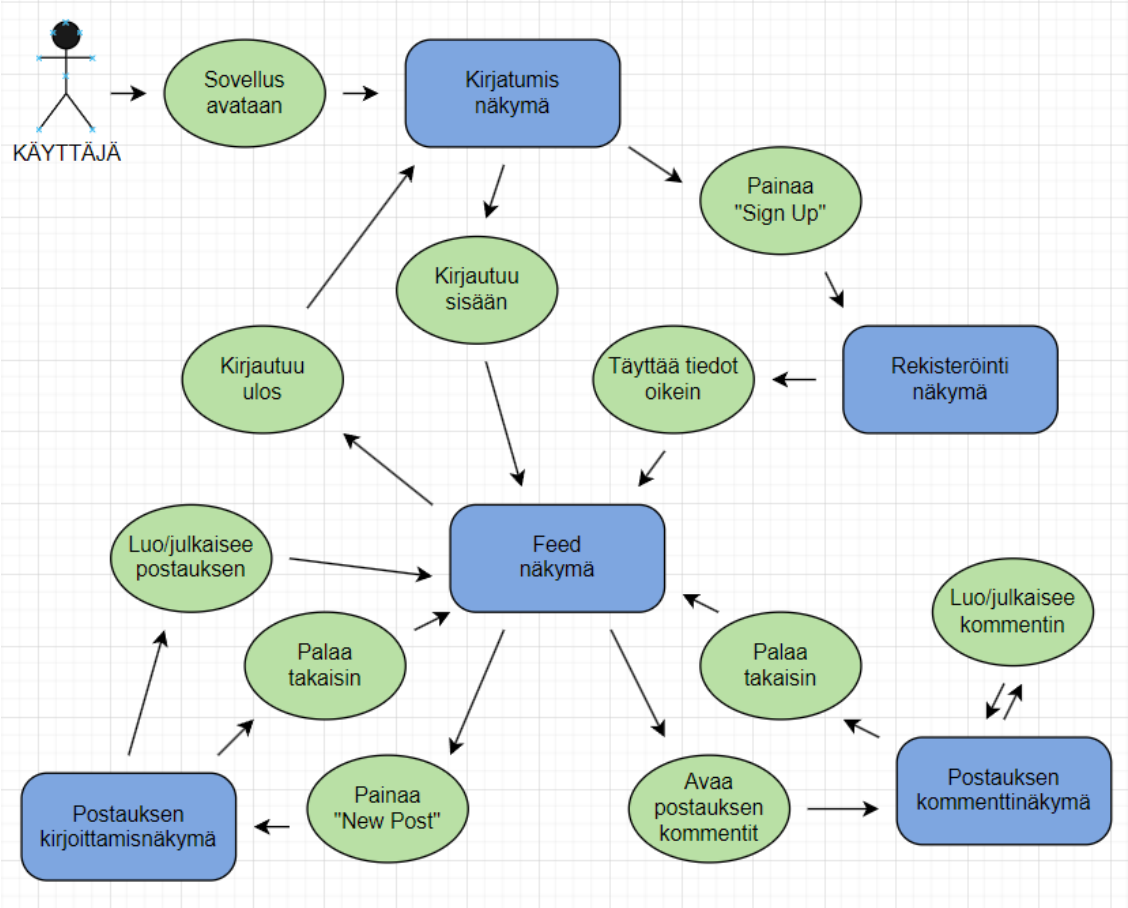
Sovelluksen sisältöä, navigointia ja toimintaa lähdettiin suunnittelemaan käyttäjäpolkujen avulla. Käyttäjäpolut ovat suunnitteluvaiheessa määrättyjä reittejä, mitä pitkin sovelluksen tai nettisivun käyttäjä kulkee päästäkseen tavoitteeseensa. Hyvin suunnitellun käyttäjäpolutuksen avulla voidaan ohjata käyttäjä tehokkaasti kohti tavoitteitaan ja pitää samalla käyttöliittymät sisällöiltään selkeinä. (Pasanen 2022, 11.)

Sovelluksen toiminnallisuus päätettiin rajata kolmeen pääkäyttäjäpolkuun, koska se nähtiin projektin toteutuksen laajuutena sopivana. Nämä käyttäjäpolut valikoituivat käyttäjäprofiilin luonniksi, julkaisun luomiseksi, sekä julkaisun kommentoinniksi. Käyttäjäpolut pyrittiin luomaan Pasasen (2022) painoittaman kolmen klikkauksen muistisäännön mukaisesti (Pasanen 2022, 11). Kaikki päätoiminnot haluttiin siis saada suoritettua näissä määrissä.

Kaikki käyttäjäpolut alkoivat kirjautumisnäkymästä. Käyttäjäprofiilin luomiseen suunnitellun käyttäjäpolun mukaan käyttäjä siirtyi kirjautumisnäkymästä rekisteröintinäkymään, jossa tapahtuu rekisteröinti. Onnistuneen rekisteröinnin päätteeksi käyttäjä siirrettiin feed-näkymään, joka päättää kyseisen käyttäjäpolun.

Postauksen luominen alkaa kirjautumisesta, jonka onnistuttua käyttäjä eteni feed-näkymään. Feed-näkymästä käyttäjä valitsi uuden julkaisun luomisen, jonka jälkeen hänet siirrettiin julkaisun kirjoittamisnäkymään. Käyttäjäpolku päättyi onnistuneeseen julkaisun tekemiseen, jonka jälkeen käyttäjä palautettiin automaattisesti feed-näkymään.

Julkaisun kommentointia varten suunniteltu käyttäjäpolku erosi julkaisun luomista siten, että kirjautumisen jälkeen käyttäjä valitsi feed-näkymässä toiminnoksi kommentoinnin. Käyttäjä siirrettiin postauksen kommenttinäkymään, minkä sisällä komentti sitten luotiin. Julkaisun kommentoinnin käyttäjäpolku päättyi kommenttinäkymään. Kuvion 9 sisältämä kaavio havainnollistaa sovellusnäkymien välille suunniteltuja käyttäjäpolkuja.



Kuvio 9 Sovelluksen pääkäyttäjäpolut kaaviomuodossa

3.1.2 Käyttöliittymät

Sovelluksen suunnitteluvaiheessa käyttöliittymien visualisointi toteutettiin mockup-muodossa. Mockupit ovat staattisia versioita sovelluksen eri näkymistä, ja eivät yleensä sisällä sovellukseen kuuluvaa funktionaalisuutta. Niiden avulla havannoidaan, miten käyttöliittymän eri komponentit toimivat yhdessä. Tyypillistä mockupeissa on testata käyttöliittymien asetelmia, värimaailmaa, sekä sisältöä. (Hufford 2022.)

Mockupit auttavat määrittämään sovelluksen käytettävyyttä, kun havannoidaan miten käyttäjä tulkitsee palvelua sen visuaalisen identiteetin kautta (Nkemchor 2023). Työkaluna mockup-mallien luomisessa käytettiin Figma-sovellusta, koska se on ilmainen ja helppokäyttöinen, sekä nettiselaimella toimiva työkalu käyttöliittymien mallintamiseen.

Suunnitteluvaiheessa käyttöliittymien mallintamiseen kuului väriteemojen, logojen, kuvien, sekä rakenteen yksityiskohtaisen asettelun suunnittelu. Pasasen (2022) mukaan käyttäjät turvautuvat sovelluksissa yleensä järjestykseen ja ennalta-arvattavuuteen. Näkymien tuttavuuden tunnetta ja yhteneväisyyttä on siis hyvä tuoda esiin väriteemoin, kuvin, sekä fontein. (Pasanen 2022, 12.)

Sovelluksen väriteeman hahmottaminen alkoi logosta. Logosta muodostettiin lopulta kaksi versiota, jotka suunniteltiin Adoben Illustrator-työpöytäsovelluksella. Logon pääväriksi muodostui persimonin oranssi, koska sen nähtiin olevan vahva ja erottuva väriteemassa. Vaihtoehtoisessa logon versiossa oli myös kuvion lisäksi kermanvalkoisena tekstinä sovelluksen nimi.

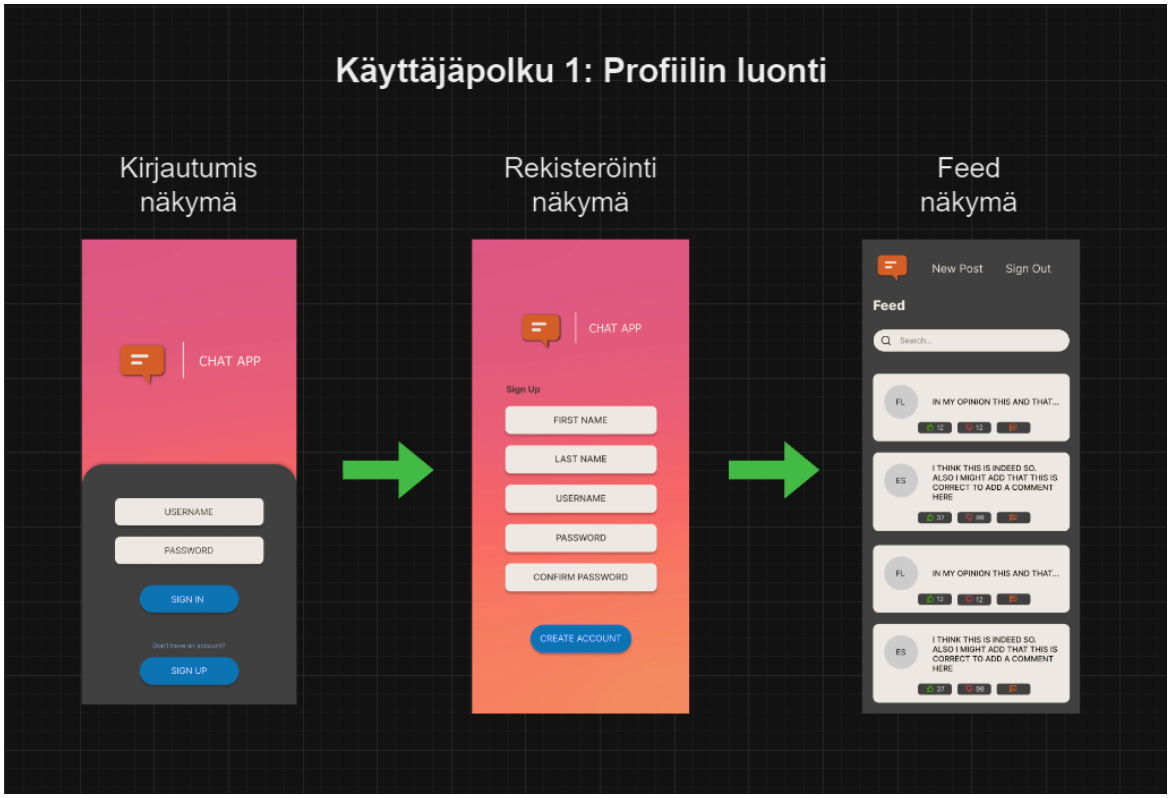
Käyttöliittymät perustuivat pääosin tummanharmaaseen taustaan. Harmaata taustaa vasten tuotiin kermanvalkoisella elementtejä, kuten tekstiä, syöttökenttiä ja muita laatikkomaisia objekteja. Kirjautumis- ja rekisteröintinäkymät pohjautuivat Adobe Illustrator -työpöytäsovelluksella luotuun taustateemaan, jossa sekotettuina olivat verenpunainen ja logosta saatu persimonin oranssi.

Laadukkaaseen käytettävyyteen kuuluu käyttöliittymästä erottuvien, helposti painettavien, sekä tarpeeksi suurien painikkeiden käyttö (Pasanen 2022, 9). Toiminnalisuuksia aktivoivia painike-elementtejä tuotiin sovelluksen näkymissä esille sinisellä taustavärillä, sekä kermanvalkoisella fontilla. Painikkeille pyrittiin antamaan suunnitteluvaiheessa riittävästi tilaa käyttöliittymissä. Lisäksi kaikki julkaisut listaava feed-näkymä sisälsi julkaisu-elementeissään nappeja, joita tuotiin esiin värein, sekä toimintaansa kuvaavin ikonein. Like-reaktiolle tarkoitettu painike sisälsi vihreää, dislike-reaktiolle tarkoitettu painike sisälsi punaista, sekä kommentointinäkymään johdattava kommenttipainike sisälsi persimonin oranssia.

Käyttöliittymät pyrittiin pitämään mallailuvaiheessa mahdollisimman selkeinä ja yksinkertaisina. Pasanen (2022) listaa käyttäjäkeskeisen sovellus-suunnittelun periaatteita seuraavasti. Sovellusta täytyy kehittää ennen kaikkea käyttäjän perspektiivistä. Käyttäjän pitäisi pystyä suorittamaan toiminnon, sekä löytämään haluamansa tieto mahdollisimman vähällä vaivalla. Lisäksi sisällön tulisi pyrkiä selkeään luettavuuteen ja ymmärrettävään, ytimekkääseen kieleen. (Pasanen 2022, 11.) Suunnitteluvaiheen viisi näkymää syntyivät sitten näiden periaatteiden pohjalta.

Kirjautumis- ja rekisteröintinäkymät sisälsivät logon, sekä toimintoja ajavat lomake-komponentit. Käyttäjän saatavilla olevat toimenpiteet kirjautumisnäkymässä olivat sisäänkirjatuminen lomakkeella, tai rekisteröintinäkymään siirtyminen alareunassa sijaitsevan painikkeen avulla. Toimintoja tekevät napit sisälsivät ytimekkäät, toimintoja kuvaavat tekstit.

Rekisteröintinäkymässä keskiöön syntyi rekisteröintilomake, jonka kirjoituskenttiä kuvattiin ytimekkäillä, paikkamerkkinä toimivilla teksteillä. Lomakkeen alareunaan lisättiin rekisteröinnin ajava painike, selkeällä toimintoa kuvaavalla tekstillä. Kuviossa 10 on nähtävissä kirjautumis-, sekä rekisteröintinäkymän mockup-versiot kuvattuna profiilin luonnin polutuksessa.



Kuvio 10 Profiilin luontia kuvaava käyttäjäpolku

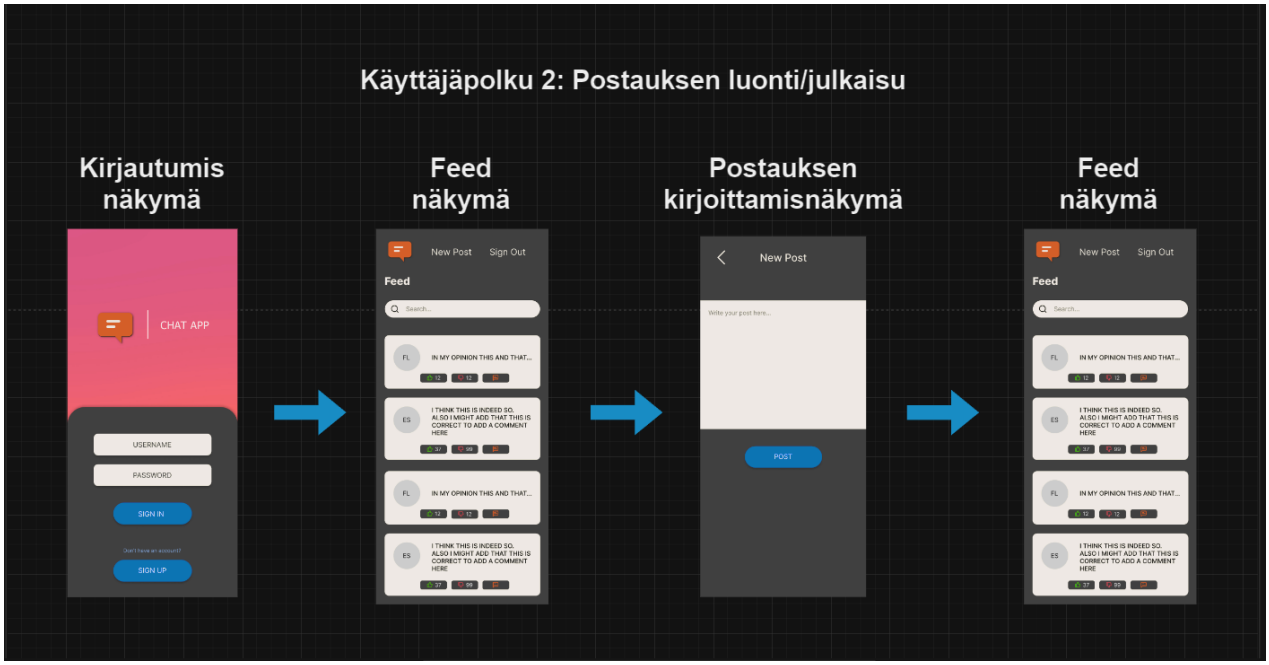
Onnistunutta kirjautumista seurasi aina siirtyminen feed-näkymään. Näkymän yläreunaan sijoittui sovelluksen navigointiin tarkoitettu navigaatiopalkki. Käyttäjän tiedetään yleensä olettavan, että navigaatio tapahtuu näkymän yläreunasta käsin. Lisäksi sovelluksen logon on tapana olla navigaatiopalkin ensimmäinen objekti vasemmalla. (Pasanen 2022, 12.) Navigaatiopalkki sisälsi vasemmalta oikealle mentäessä logopainikkeen, uuden julkaisun painikkeen, sekä

uloskirjautumiseen tarkoitetun painikkeen. Painikkeet sijoiteltiin vaakasuunnassa näkymään tasaisesti siten, että yksittäisten painikkeiden, sekä koko navigaatiopalkin ulkoreunoille jäi tasaisesti tilaa. Kaksi jälkimmäistä painiketta sisälsivät kermanvalkoiset, selkeällä otsikkomaisella fontilla varustetut tekstit.

Navigaatio-osan alle vasemmalle sijoittui näkymää kuvaava paksu otsikko. Otsikon alle sijoitettiin kermanvalkoinen hakupalkki, joka kattoi leveyssuunnassa lähes koko näkymän. Leveys pidettiin leveänä, että käyttäjä mahtuisi kirjoittamaan sen sisään mahdollisimman paljon näkyvästi. Lisäksi hakupalkin vasempaan reunaan asetettiin toimintoaan käyttäjälle kuvaava tiimalasi-ikoni, jonka viereen asettui teksti paikkamerkkinä.

Feed-näkymässä suurin osa annettiin lista-komponentille, joka koostui lukuisista julkaisukomponenteista. Yksittäinen listan julkaisukomponentti rakentui kermanvalkoiselle pohjalle kahdesta vaakarivistä. Ylimmälle riville sijoittui vasemmalta oikealle pyöreä profiilikuva, jonka jälkeen julkaisun kirjoitus tummana tekstinä. Alemmalle riville sijoitettiin tasaisesti keskelle asettuvat, toiminnallisuuksia ajavat kolme painiketta.

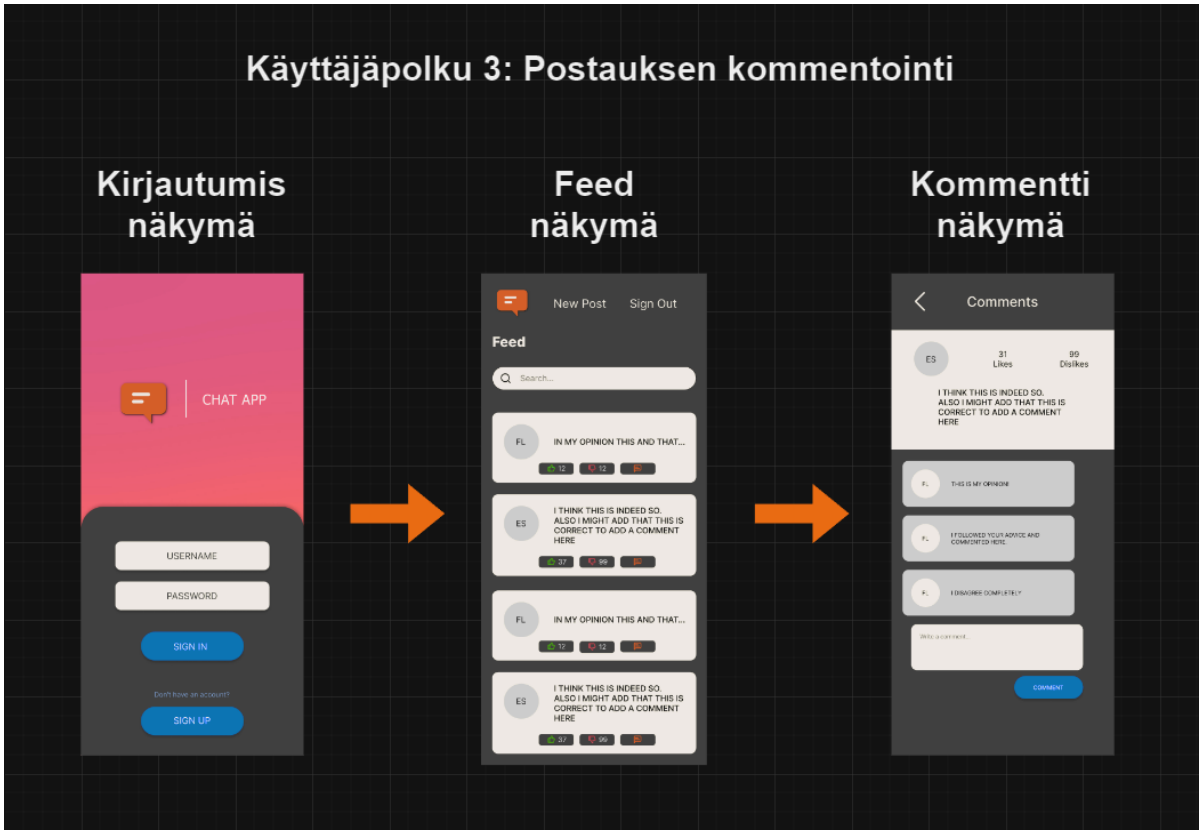
Julkaisun kirjoittamiselle tarkoitettuun näkymään navigointi suunniteltiin feed-näkymän navigaatiopalkissa sijaitsevan uudelle julkaisulle tarkoitetun painikkeen kautta. Näkymän yläreunaan sijoitettiin otsikkona kermanvalkoinen teksti, jonka viereen vasemmella tuli samanvärinen, näkymästä takaisin feediin navigoiva nuoli-ikoni. Julkaisun kirjoittamisnäkymän keskiöön sijoittui suuri, monelle tekstiriville tarkoitettu syöttökenttä. Syöttökentän vasempaan yläreunaan annettiin paikkateksti, jonka tarkoituksena oli jälleen ohjata käyttäjää toimimaan kentän kanssa. Syöttökentän alapuolelle sijoitettiin vielä toimintopainike kirjoituksen julkaisemiseksi. Feed-, sekä julkaisun kirjoittamisnäkymän mockup-versiot osana julkaisun luomisen käyttäjäpolkua ovat nähtävissä kuviossa 11.



Kuvio 11 Julkaisun tekemistä kuvaava käyttäjäpolku

Kommenttinäkymään siirtyminen suunniteltiin feed-näkymän julkaisuelementin kommenttipainikkeen kautta. Kommenttinäkymän yläreuna mukautettiin täysin julkaisun kirjoittamisnäkymään pohjautuvasti, jonka alla näytettiin laatikkoimainen, kermanvalkoisella pohjalla tyylitelty julkaisu-elementti. Julkaisun tietojen asetelma rakentui elementissä jälleen kahteen vaakariviin. Ylärivillä oli tasaisin välityksin julkaisun tekijän profiilikuva, tykkäysten lukumäärä, sekä dislikejen lukumäärä. Keskelle alariviä asetettiin julkaisun tekstiosio, jonka reunoille jätettiin riittävästi tyhjää tilaa. Näin pyrittiin pitämään sisältö selkeänä ja tekstiosio helposti luettavissa.

Julkaisutietojen alle sijoitettiin lista-komponenttina kommentit. Lista koostui julkaistuista kommenttielementeistä, sekä alimmaiseksi asetetusta uuden kommentin kirjoittamiseen tarkoitetusta syöttökentästä. Julkaistut kommentit pyrittiin erottamaan syöttökentästä taustavärillä, sekä vaakasuuntaisella asettelulla. Listan julkaistut kommenttielementit rakentuivat tekijän profiilikuvasta, sekä kommenttitekstistä. Kommentin kirjoittamisen syöttökenttä sisälsi paikkatekstin vasemmassa yläreunassa, jonka tarkoituksena jälleen ohjata käyttäjää. Kommentin julkaisutoimintoa palveli syöttökentän alapuolella vasemmalla oleva painike. Kuvio 12 sisältää kommenttinäkymän mockup-version, osana julkaisun kommentoinnin käyttäjäpolkua.



Kuvio 12 Julkaisun kommentointia kuvaava käyttäjäpolku

3.1.3 Palvelinpuolen mallintaminen

Sovelluksen suunnitteluvaiheessa nähtiin parhaaksi, että sen alla tapahtuva palvelinlogiikka rakennettaisiin mahdollisimman yksinkertaiseksi ja helppolukuiseksi, mutta kuitenkin toiminnaltaan tehokkaaksi. Yksinkertaisuus siitä syystä, että painopiste tutkimustyöllä oli käyttäjälle näkyvässä päässä.

Sovellukseen suunnitellut toiminnallisuudet määriteltiin seuraavasti. Palvelimelta vaadittiin ainakin datan hakemista, tallentamista, sekä uuden datan luontia. Datan varastoimiseen tarvittiin tietokanta, jonka vaatimuksena nähtiin dataobjektien tallentaminen ainakin käyttäjätiedoille, julkaisuille, sekä kommenteille.

Palvelimen tuli pystyä suorittamaan sovelluksen toiminnalisuuksiin vaadittuja kyselyjä, eli kommunikoimaan tietokannan ja käyttäjäpään välillä. Tarvittaviksi kyselyiksi nähtiin suunnitteluvaiheessa käyttäjätietojen hakeminen, uusien käyttäjätietojen luominen,

sisäänkirjautuminen, kaiken julkaisudatan hakeminen, uusien julkaisujen luominen, tietyn julkaisun hakeminen, julkaisukohtaisten kommenttien hakeminen, kommentien luominen, tykkäyksien luominen, sekä dislikejen luominen.

3.2 Sovelluksen kehittäminen

3.2.1 Natiivisovellusprojektin alustus

Varsinaisen sovelluksen kehittämisen toteutus alkoi käyttöliittymien ohjelmoinnilla. Projekti alustettiin paikallisesti Node.js-ajoympäristössä, johtuen sen tarjoamista työkaluista, sekä mukautuvuudesta mobiilisovelluskehitykseen. Lisäksi Node.js mahdollisti myös palvelinpuolen alustuksen samalla teknologialla, mikä nähtiin projektin yhteneväisyyden kannalta tärkeänä.

Sovelluskehityksen natiivialustaksi valikoitui React Native -kirjasto, yhdistettynä Expo-komentoliittymällä. React Nativen ollessa johtava viitekehys mobiilisovelluskehityksessä, nähtiin sen tarjoavan parhaat työkalut suunnitteluvaiheessa määriteltyjen käyttöliittymien tehokkaaseen rakentamiseen. Expo CLI valittiin komentoliittymäksi, koska se mahdollisti projektin käynnistämisen kehitysvaiheessa yhdellä komennolla, jonka jälkeen sovelluksen sai avattua suoraan mobiililaitteella, Expon tarjoaman Expo Go -sovelluksen kautta. Android Studio -työpöytäsovellus tarjosi myös Android-emulaattorillaan vaihtoehtoisen tien sovelluksen avaamiseen mobiilikoossa.

Useasti kutsuttavia tyylimäärittelyjä, kuten värejä ja fontteja varten luotiin oma JavaScript-tiedosto. Sovelluksessa esiintyvät fontit määriteltiin käyttöjärjestelmäkohtaisesti React Native -kirjaston tarjoamalla Platform-työkalulla. Androidille fontiksi asennettiin Roboto, kun taas IOS:ille Verdana. Muissa tapauksissa oletusfonttina toimi System. Sovelluksessa esiintyvät fonttikoot jaettiin arvoillaan sisältötekstiin, alaotsikkoon ja otsikkoon. Lisäksi tekstien paksuudet saivat arvot normaaliin ja paksuun.

React Native -kirjaston Text-komponentti räätälöitiin sovelluksen tekstikomponentteja varten omassa JSX-tiedostossa, tyylitiedostosta kutsutuilla väreillä ja fonttiko'oilla. Näin mahdollistettiin usein käytetyn teksti-komponentin nopea määrittely näkymissä, kun tyylit oli ennalta määrätty.

Sovelluksen listakomponentteja varten luotiin mock dataa, joka sijoitettiin omaan JavaScript-tiedostoonsa. Mock data koostui kovakoodatuista julkaisuista ja kommenteista. Tällä tavoin dataa voitiin tulostaa listakomponenteissa käyttöliittymien kehitysvaiheessa, ennen tietokannan ja palvelinlogiikan yhdistämistä sovelluskokonaisuuteen.

3.2.2 Käyttöliittymien ohjelmointi

Tässä luvussa käsitellään natiivisovelluksen käyttöliittymien ohjelmointivaihetta. React Native -sovelluksen komponenttien tiedostorakenne pohjustettiin pääkomponentilla, jonka sisältö rakentui päänäkymien, sekä reitityksen komponenteista.

Päänäkymien React-komponenttien ohjelmointi alkoi feed-näkymästä. Feed-komponentti rakentui monesta alakomponentista, seuraten mockup-versiota. Alakomponentit olivat NavBar-navigaatiopalkille, otsikolle, hakupalkille, sekä listakomponentille. Feed-komponentin hierarkia on nähtävissä kuviossa 13.

```
const Feed = ({ setToken }) => {
    return (
        <View style={styles.container}>
            <NavBar setToken={setToken} />
            <Header />
            <SearchBar />
            <ChatList />
        </View>
    );
};
```

Kuvio 13 Alakomponentit sisältävä Feed-komponentti

Navigaatiopalkki rakentui täysin mockup-version pohjalta. Otsikon komponenttiin lisättiin mockup-versiosta poiketen kirjautuneen käyttäjän profiilikuva, sekä käyttäjänimi. Lisäyksen uskottiin selkeyttävän viestintää sisäänkirjautumisesta käyttäjälle, kun kirjautuneen profiilin tuntomerkit ovat nähtävissä näkymässä. Vaihtoehtoisesti profiilikuvan puuttuessa näytettiin käyttäjää kuvaava

ikoni. Sama määrittely toteutettiin sovelluksen jokaisessa vastaavassa tapauksessa, kun käyttäjän profiilikuvaa pyrittiin tulostamaan. Listakomponentin julkaisukomponentti sai mockup-versiosta poiketen lisäyksenä kirjoittajan käyttäjänimen julkaisutekstin yläpuolelle. Lisäksi julkaisukomponentin toimintopainikkeiden väritys muutettiin käänteiseksi, jotta väritys olisi selkeämpi.

Julkaisun kirjoittamisnäkymä rakentui pääkomponentista, joka sisälsi alakomponenttina julkaisun kirjoittamislomakkeen, sekä otsikkopalkin. Otsikkopalkille luotiin uudelleen käytettävä komponentti, joka kutsuessaan otti text-arvona otsikolle tekstin. Mockupista poiketen otsikkokomponenttiin sisällytettiin kirjautuneen käyttäjän profiilikuva näkymän oikeaan yläreunaan. Julkaisun kirjoittamisnäkymän lomakekomponentissa sijaitseva painike muutettiin mockup-version sinisestä persimmonin oranssiin. Muutoksen nähtiin tuovan näkymään teemaan yhdistävää vaikutusta, kun siihen lisättiin teeman pääväriä.

Kommenttinäkymän pääkomponentti rakentui otsikko-, julkaisu-, lomake-, sekä listakomponentista. Kirjoittamisnäkymän tapaan otsikkokomponenttina käytettiin valmista komponenttia. Näkymä rakentui mockup-versiota mukaillen, muutamia poikkeuksia lukuunottamatta. Julkaisutiedot tulostava julkaisukomponentti sai mockupista poiketen lisäyksenä kirjoittajan käyttäjänimen, julkaisutekstin yläpuolelle. Kommenttilomake siirrettiin mockupista poiketen kommenttilistan yläpuolelle. Lisäksi lomakkeen painike muutettiin persimonnin oranssiksi, sekä sisällöksi annettiin tekstin sijaan toimintoa kuvaava puhekuplaikoni.

Kirjautumisnäkymä rakentui täysin mockup-version pohjalta, kun ei nähty syytä tehdä muutoksia. Sen pääkomponentti rakentui kahdesta alakomponentista. Ensimmäisenä logokomponentti, joka tulosti sovelluksen logon. Toisena alakomponenttina lisättiin kirjautumislomake, joka sisälsi näkymän syöttökentät ja painikkeet. Lomakkeen viestinnässä otettiin käyttöön React Native -kirjaston Alert-komponentti, joka mahdollisti lomakkeen lähettämisen yhteydessä ilmoituksen tulostuksen virhetilanteissa. Alert-komponenttia hyödynnettiin myös muissa sovelluksen lomakekomponenteissa.

Rekisteröintinäkymän rakenteeseen tehtiin ohjelmointivaiheessa vain pieniä muutoksia. Päänäkymä sisälsi mockup-versiota mukaillen alakomponentteinaan logon, lomakkeen otsikoivan

tekstin, ja lomakkeen. Lomakkeeseen lisättiin ohjelmointivaiheessa yksi syöttökenttä profiilikuvan lisäämistä varten. Toinen lisäys oli lomakekomponentin alle sijoitettu painike, jonka kautta käyttäjä pystyi halutessaan navigoimaan takaisin kirjautumisnäkymään.

3.2.3 Reitityksen rakenne

Käyttöliittymien väliseen navigointiin asennettiin React Router Native -kirjasto, johtuen sen yhteensopivuudesta React Native -sovelluksen kanssa. Sovelluksen reititys rakentui lopulta kahteen haarautuvaan reitityspuuhun, jota hallittiin useState-tilafunktiolla. Tilafunktion tarkoitus oli jakaa sovelluksen reititys kahteen tilanteeseen: Käyttäjä on kirjatunut sisään tai käyttäjä ei ole kirjautunut sisään.

Sisäänkirjautumattoman tilan reititys sisälsi Route-komponentit kirjautumis-, sekä rekisteröintinäkymän React-komponentteihin. Molempien reittien Route-objektit saivat path-arvokseen ”/”-reitin, eli ne toimivat reitityksensä oletusnäkyminä. Oletusnäkymien tilaa hallittiin jälleen omalla tilafunktiollaan.

Sovelluksen reitityksen sisäänkirjautunut tila piti sisällään reitit jäljellä oleviin pääkomponentteihin. Route-komponentit sisälsivät feed-näkymän oletuskomponenttina, kun taas vaihtoehtoiset reittikomponentit sisälsivät uuden julkaisun ja yksittäisen julkaisun näkymien komponentit URL-arvoineen.

3.2.4 Palvelimen alustaminen

Sovelluksen palvelinkokonaisuus toteutettiin GraphQL-kyselykielellä Apollo Server -palvelinkirjaston kanssa. Ajoympäristönä palvelimelle toimi React Native -sovelluksen tapaan Node.js. GraphQL-kyselykielen valittiin sen ollessa joustava ja yhteensopiva muita sovelluksen teknologioita kohtaan. Apollo Server valikoitui palvelimen pohjaksi, koska se tarjosi suoraviivaisen tavan rakentaa GraphQL-kyselykielellä toteutetun palvelimen, ollessaan samalla täysin yhteensopiva Node.js-ajoympäristön kanssa.

Palvelimen hakemistorakenteessa pyrittiin yksikertaisuuteen ja selkeyteen. Palvelimen kehitysympäristön ajon käynnistys Apollo Server -kirjastolla rakennettiin yksittäisen JavaScript-

tiedoston sisään. GraphQL-kyselyt jaettiin kahteen tiedostoon palvelinprojektin juureen. Ensimmäisessä tiedostossa määriteltiin palvelimen schema, jonka sisällä muotoiltiin sovelluksessa käytettävä data. Toinen GraphQL-tiedosto oli resolveri-funktioille, jotka sisälsivät datan käsittelemisen ja yhdistivät tietokannan palvelimeen. Kuviossa 17 on nähtävissä Apollo Serverin käynnistävä funktionaalisuus.

```
const server = new ApolloServer({
  typeDefs,
  resolvers,
});


startStandaloneServer(server, {
  listen: { port: 4000 },
  context: async ({ req, res }) => {
    const auth = req ? req.headers.authorization : null
    if (auth && auth.startsWith('Bearer ')) {
      const decodedToken = jwt.verify(
        auth.substring(7), process.env.JWT_SECRET
      )
      const currentUser = await User.findById(decodedToken.id)
      return { currentUser }
    }
  },
}).then(({ url }) => {
  console.log(`Server ready at ${url}/graphql`)
});
```

Kuvio 14 Apollo Serverin käynnistävä funktionaalisuus palvelimessa

Scheman sisältämä data muotoiltiin type-objektien avulla. Objektit luotiin pohjaksi käyttäjälle, autentikaatioon tarvittavalle tunnisteelle, julkaisulle, kommentille, like-reaktiolle, dislike-reaktiolle, sekä toiminnalisuuksia varten tarvittaville kyselyille ja mutaatioille.

Palvelimen kyselyillä haettiin dataa kolmessa muodossa. Ensimmäisenä kaiken julkaisudatan hakeminen, toisena kirjautuneen käyttäjän hakeminen, kolmantena yksittäisen julkaisun hakeminen annetun id-arvon perusteella. Mutaatioita palvelin tarjosi yhteensä kuusi kappaletta. Mutaatiot käsittelivät sisäänkirjautumisen annetuilla käyttäjätiedoilla, käyttäjän luomisen annetuilla käyttäjätiedoilla, julkaisun luomisen annetulla tekstitiedolla, kommentin luomisen

julkaisun id-arvolla ja annetulla tekstitiedolla, like-reaktion luomisen julkaisun id-arvolla, sekä dislike-reaktion luomisen julkaisun id-arvolla.

React Native -sovelluksen puolella hyödynnettiin Apollo Client -kirjastoa. Apollo Client mahdollisti yhteyden luomisen React Native -sovelluksen ja palvelimen välille. Alustus tapahtui asentamalla tarvittavat riippuvuudet Apollo Clientille ja GraphQL-kyselyille. Apollo Client alustettiin omassa tiedostossaan, jonka jälkeen se yhdistettiin ApolloProvider-komponenttina natiivisovelluksen pääkomponentin ympärille. Natiivisovelluksen projektihakemistoon lisättiin JavaScript-tiedostot alustamaan palvelimelta peilattavia kyselyjä, sekä mutaatioita.

3.2.5 Datan tallentaminen

Datan tallentaminen toteutettiin palvelimen kanssa dokumenttipohjaisesti MongoDB-tietokannassa. MongoDB valikoitui, koska se tarjosi helposti luettavan ja joustavan tavan tallentaa sovelluksessa käsiteltävää dataa. Riippuvuus tietokannan ja palvelinprojektin välille asennettiin Node.js-ajoympäristöön sopivan Mongoose-kirjaston avulla. Mongoose-kirjasto mahdollisti GraphQL-kielellä rakennettujen type-objektien helpon ja selkeän peilaamisen palvelimessa tietokantaan tallentamisen yhteydessä.

Yhteyden muodostamiseen vaadittiin myös uuden projektin alustaminen MongoDB Atlas -selainsovelluksen avulla, josta saatiin palvelimeen tarvittava URI-koodi. Koodin suojaamista varten palvelinprojektiin asennettiin Node.js-ajoympäristön tarjoama Dontev-paketti, jonka jälkeen URI-koodi luettiin projektin juuressa sijaitsevasta .env-tiedostosta. URI-koodi luettiin Mongoose-kirjaston valmiilla connect-funktiolla, jonka jälkeen yhteys tietokantaan oli valmis. MongoDB Atlas tarjosi myös mahdollisuuden dokumenttidatan tarkasteluun kehitystyön lomassa, jota toteutuksessa usein hyödynnettiin.

Palvelinhakemistoon lisättiin JavaScript-tiedostot tietokannan dokumentteja edustavia model-objekteja varten. Model-objektit luotiin palvelindataa peilaten käyttäjälle, julkaisulle, kommentille, like-reaktiolle, sekä dislike-reaktiolle. Model-tiedostot sisälsivät omat schema-objektinsa, joissa määriteltiin dokumenttien rakenteet vastaamaan palvelimen kyselyissä määriteltyjä type-objekteja.

Schema-objektien määrittelyssä hyödynnettiin myös Mongoosen unique validator -työkalua datan validoinnissa. Validoinnin avulla oli helppoa varmistaa, että tietokantaan tallennettava data pysyi määritellyssä muodossa. Model-tiedostojen lisäyksen jälkeen niitä hyödynnettiin GraphQL-palvelimen toiminnallisuusmäärittelyissä, eli resolverifunktioissa. Havainnollistavana esimerkkinä kaiken julkaisudatan, sekä kaiken käyttäjädatan hakevat resolverifunktiot ovat nähtävissä kuviossa 15.

```
Query: {
  // QUERY 1
  allPosts: async () => {
    return Post.find({})
  },
  // QUERY 2
  allUsers: async ()  =>  {
    return User.find({})
  },
```

Kuvio 15 Kyselyresolverit datan hakemiseen tietokannasta

3.2.6 Toiminnallisuudet sovelluksessa

Sovelluksen toiminnallisuuksien rakentaminen palvelimen ja tietokannan kanssa aloitettiin käyttäjänhallintaa käsittelevistä toiminnallisuuksista. Palvelimen puolella käyttäjänhallintaa varten täytyi asentaa avuksi muutamia NPM-paketteja. Ensimmäisenä JSON Web Token -paketti, joka tunnetaan myös lyhenteellä JWT. JWT-paketti mahdollisti käyttäjän varmistamisen toteuttamisen sovelluksessa. Toiseksi käyttäjänhallinassa hyödynnettiin Bcrypt-pakettia, joka mahdollisti salasana-arvon salaamisen rekisteröinnin, sekä kirjautumisen yhteydessä. Natiivisovellukseen asennettiin React Native Async Storage -työkalu, jonka avulla mahdollistettiin autentikaatiotunnisteen tallentaminen käyttäjähallinnassa.

Sisäänkirjautumisen ja rekisteröinnin funktionaalisuudet aktivoitiin käyttöliittymiensä lomakekomponenteissa. Lomakkeet rakentuivat React Native -kirjaston NativeTextInput-komponenteista, jotka vastaanottivat syöttökenttään tarvittavia arvoja. Valmis NativeTextInput-komponentti mahdollisti esimerkiksi automaattisen paikkamerkin kirjoittamisen, sekä salasanakentän salaamisen siihen kirjoittamisen yhteydessä käyttöliittymässä.

Sovelluksen kaikkia käyttäjältä tietoa vastaanottavia syöttökenttien arvoja hallittiin React-kirjaston useState-funktiolla. Kyseisen funktion avulla syöttökenttien arvojen tilaa pystyttiin päivittämään reaaliaikaisesti, käyttäjän kirjoittaessa lomakekomponenttiin.

Kirjautumislomakkeen painike-komponentti asennettiin aktivoimaan kirjautumisen käsittelty painamisen yhteydessä, lomakkeeseen annetuilla arvoilla. Kirjautumista käsittelevä funktio lähetti annetut arvot palvelimen puolelle kirjatumista käsittelevälle resolverifunktiolle, määritteli sitten autentikaatiotunnisteen muistiin Async Storage -työkalulla, pyyhki tarvittaessa muistista vanhan datan, sekä päivitti käyttöliittymän reitityksen tilan, jotta funktio pystyi viimeiseksi ohjata käyttäjän feed-näkymään.

Navigaatioiden aktivoinnissa käytettiin React Router Native -kirjaston tarjoamaa useNavigate-funktiota, joka käyttää hyödyksi reitityksen path-arvoja. Kuviossa 16 nähdään React-komponentissa tapahtuva kirjautumisen käsittely.

```
const handleSignIn = async () => {
    try {
      // LOG IN USER
      const token = await login(username, password);
      setToken(token);
      // NAVIGATE TO FEED
      navigate('/');
    } catch (error) {
      Alert.alert('Login failed', 'Invalid username or password');
    }
};
```

Kuvio 16 Kirjautumista käsittelevä funktio lomakkeen lähettämisen yhteydessä

Kirjautumislomakkeen tapaan rekisteröintilomake vastaanotti NativeTextInput-komponenteissaan tilankäsittelyllä hallittuja arvoja, jotka lähetettiin lomakkeen painikkeen aktivoimana palvelimelle käyttäjän luontia varten. Onnistuneen käyttäjän luomisen jälkeen ajettiin kirjautumisen funktionaalisuus, joka haettiin samasta JavaScript-tiedostosta kuin kirjautumislomakkeenkin yhteydessä. Rekisteröinnin toiminnallisuus lopetettiin automaattisen kirjautumisen päätteeksi feed-näkymään.

Seuraavaksi käsitellään sovelluksen pääsivun, eli Feed-näkymän ominaisuuksia. Feed-näkymä rakennettiin sisällyttämään useita toiminnallisuuksia, kuten uloskirjatumisen, kirjautuneen käyttäjän tietojen tulostamisen, sekä kaikkien julkaisujen tulostamisen.

Uloskirjautuminen sijoitettiin käyttöliittymän yläosassa sijaitsevaan navigointipalkkikomponenttiin. Navigaatiopalkin komponentti koostui tab-komponenteista, joista oikeanpuoleisin aktivoi uloskirjautumisen funktionaalisuuden. Uloskirjautuminen käsiteltiin funktiossaan nullaamalla ensiksi reitityksen tila natiivisovelluksessa, jotta mahdollistettiin jälleen kirjautumisnäkymään pääsy. Tämän jälkeen autentikaatiotunnus tyhjennettiin Async Storage -työkalun, sekä Apollo Client -työkalun muistista. Viimeisenä funktio varmisti vielä navigaation kirjautumisnäkymään.

Käyttäjätietojen tulostaminen kirjautuneena puuttui suunnitteluvaiheen mockup-versiosta, mutta se toteutettiin ohjelmointivaiheessa feed-näkymään. Tulostus tapahtui feed-komponentin otsikkokomponentissa, joka sisälsi nyt otsikkotekstin lisäksi oikeaan reunaan asetettuna kirjautuneen käyttäjän nimen, sekä profiilikuvan. Toiminnallisuudessa hyödynnettiin palvelimen kirjautuneen käyttäjän hakevaa kyselyä, jonka avulla tarvittavat arvot saatiin yhdistettyä komponentteihin.

Feed-näkymän keskeisinpänä ominaisuutena sisällytettiin kaikkien julkaisujen tulostaminen listakomponenttina. Listakomponentti luotiin React Native -kirjaston FlatList-komponentilla, joka vastaanotti data-arvona palvelimen kaikki julkaisut hakevan kyselyn palautuksen. FlatList-komponentti osasi tämän jälkeen id-arvojen perusteella erotella julkaisut omiksi komponenteikseen ja luoda listauksen käyttöliittymään. Tärkeänä ominaisuutena listakomponentille annettiin mahdollisuus sen skrollaamiseen ilman, että se vaikutti muiden komponenttien sijaintiin näkymässä. Listakomponentissa pystyttiin huomioimaan myös tilanne, jossa julkaisudataa ei löydy. Tyhjän komponentin sijasta voitiin tulostaa tekstikomponentti viestimään käyttäjälle, että julkaisuja ei ole saatavilla.

Julkaisulistan yksittäiseen julkaisukomponenttiin sisällytettiin julkaisuun kohdennettuja toimintoja ajavia painikekomponentteja. Julkaisun like-reaktion, sekä dislike-reaktion toiminnoissa tapahtuvat funktionaalisuudet olivat lähes täysin identtiset. Painikkeen painamisesta laukaistuna palvelimen resolveri-funktiolle lähetettiin toiminnon kohteena olleen julkaisun id-arvo, jonka

seurauksena uusi objekti luotiin tietokantaan. Lisäksi uusi like-reaktio, tai dislike-reaktio liitettiin juuri luodun objektin id-arvon avulla myös kohteena olleen julkaisuobjektin vastaavaan kenttään tietokannassa. Reaktioiden luomisen yhteydessä otettiin huomioon myös tilanteet, jossa sama käyttäjä yrittäisi tykätä samasta julkaisusta useammin kuin kerran. Palvelimen resolveri-funktiot rakennettiin huomaamaan tälläiset tilanteet, ja seurauksena käyttäjälle tulostettiin aina virheilmoitus Alert-komponenttina React Native -sovelluksen puolella.

Julkaisun luonti sisällytettiin sovelluksen julkaisun kirjoittamisnäkymään. Julkaisemisen toiminnallisuuden käsittely tapahtui julkaisulomake-komponentissa, joka vastaanotti NativeTextInput-komponenttiinsa käyttäjän syötteen. Julkaisemisen tapahtuman käsittely laukaistiin painikkeella, jonka seurauksena palvelimen mutaatio vastaanotti lomakkeen tekstiarvot ja julkaisu tallennettiin tietokantaan. Lomakekomponentin tapahtumakäsittelijä ohjasi vielä käyttäjän automaattisesti onnistuneen julkaisemisen päätteeksi takaisin feed-näkymään, jonka listakomponenttiin uusi julkaisu myös tulostettiin. Palvelimessa julkaisun luonnista vastaava resolveri-funktio huolehtii, että uuden julkaisu-objektin arvoina tykkäykset, disliket, sekä kommentit saavat oletuksena aina tyhjät arvot.

Kommenttinäkymään siirtymisen yhteydessä täytyi suorittaa palvelimeen yksittäisen julkaisun hakemisen kysely. Kysely laukaistiin feed-näkymän julkaisukomponentin kommenttipainikkeesta, jonka tapahtumakäsittelijän navigaatio-funktion sisältämään URL-arvoon sisällytettiin kohteena olevan julkaisun id-arvo. Tämän jälkeen kommentinäkymän pääkomponentti vastaanotti julkaisun id-arvon omasta URL-reitityksestään, React Router Native -kirjaston tarjoaman useParams-työkalun avulla.

Kohteena olevan julkaisun id-arvon avulla voitiin kommenttinäkymän pääkomponentissa ajaa yksittäisen julkaisun kysely palvelimelle, jonka seurauksena halutun julkaisun tiedot tulostettiin näkymän julkaisukomponenttiin. Yksittäisen julkaisun sisältämät kommentit voitiin myös hakea tulostamista varten. Kommentit tulostettiin tuttuun tapaan FlatList-komponenttina, id-arvoilla eroteltuina. Kommenttien haun palauttaessa tyhjää tulostettiin teksti-komponenttina kirjoitus viestimään käyttäjälle tilanteesta.

Kommenttinäkymän lomakekomponentti vastasi uuden kommentin luomisen toiminnallisuudesta. Lomakekomponentti vastaanotti kohdejulkaisun id-arvon, jota hyödynnettiin uuden kommentin luomisen mutaatiossa. Lomakkeen NativeTextInput-komponentti sisällytti käyttäjältä saaman tekstiarvon ja kommentin luonnin funktionaalisuus laukaistiin painikekomponentista. Sitten kommentin luomista vastaavaa mutaatiota kutsuttiin kommenttitekstin, sekä kohdejulkaisun id-arvoilla, jonka jälkeen kommentti luotiin palvelimessa ja tallennettiin tietokantaan.

Kommentin luomisen päätteeksi haluttiin pysyä yhä samassa näkymässä, joten lomakekomponentin syöttökenttä täytyi tyhjentää. Lisäksi oli tarpeen päivittää kommenttinäkymän listakomponentin tila uudella kommentilla. Uuden kommentin luomisen mutaatiota kutsuessa täytyi aina suorittaa kohdennetun yksittäisen julkaisun kysely sovelluksen välimuistista, jonka yhteydessä välimuisti päivitettiin uudella kommentilla. Lopputuloksena uusi kommentti päivittyi näkymään automaattisesti kommenttilomakkeen lähettämisen jälkeen ilman, että käyttäjän tarvitsi poistua sivulta tilan päivittämiseksi. Kuviossa 17 on nähtävissä funktionaalisuus, joka ajetaan kommentin luomisesta vastaavaa mutaatiota kutsuessa.

```
const [createComment] = useMutation(CREATE_COMMENT, {
    // UPDATE FUNCTION FOR THE CACHE AFTER NEW COMMENT ADDED
    update: (cache, { data: { createComment } }) => {
        // READ POST DATA FROM CACHE
        const cachedData = cache.readQuery({
            query: GET_SINGLE_POST,
            variables: { id: postId }
        });
        // UPDATE THE POST CACHE WITH NEW COMMENT
        cache.writeQuery({
            query: GET_SINGLE_POST,
            variables: { id: postId },
            data: {
                singlePost: {
                    ...cachedData.singlePost,
                    comments: [
                        ...cachedData.singlePost.comments,
                        createComment
                    ]
                }
            }
        });
    }
});
```

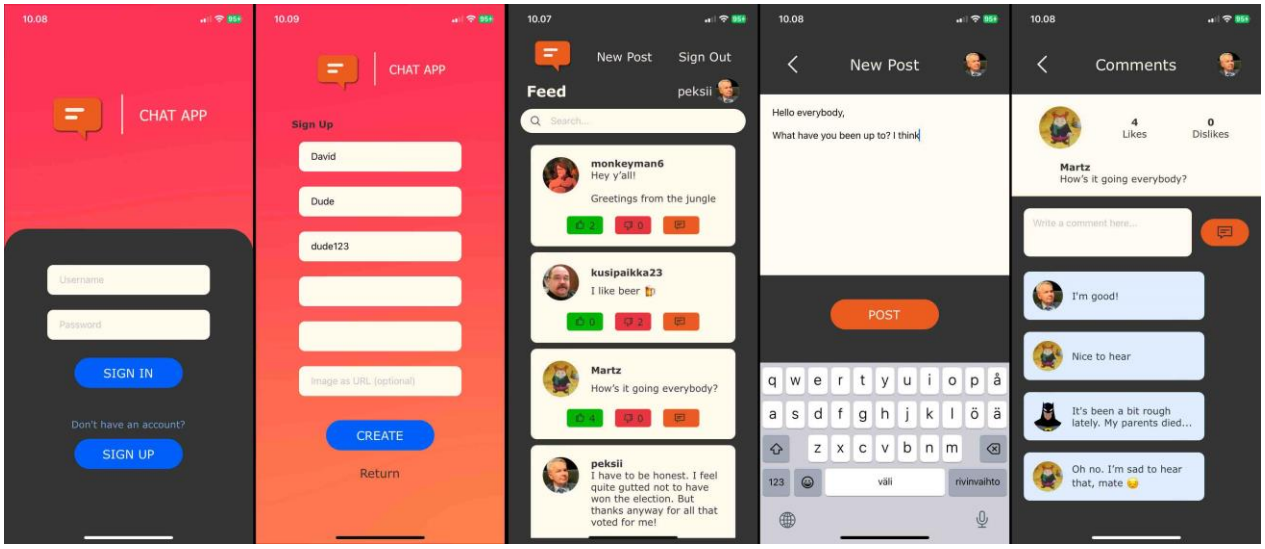
Kuvio 17 Välimuistin päivittävä funktionaalisuus kommentin luomisen yhteydessä

Käyttöliittymien kaikkiin painikekomponentteihin lisättiin toteutuksen loppuvaiheilla painikkeen painamisesta aktivoituvaa tyylinvaihtelua. Lisäys nähtiin perusteltuna, koska se toi lisää havainnollistavaa viestintää käyttöliittymiin, painikkeiden painamisen yhteydessä. Tyylittelyn vaihtelu perustui React-kirjaston useState-funktion tilalogiikkaan. Painamaisen ollessa aktiivinen, painike sai pienen muutoksen väritykseensä, luoden painamista viestivän animaatiomaisen efektin. Painamisen loppuessa tilamuuttuja muutti värityksen takaisin normaalin tilaansa.

4 Tutkimustyön tulokset

Tutkimustyön tavoitteena oli kehittää natiivimobiilisovellus, jonka tarkoituksena oli osoittaa laatua sen toiminnallisuudessa, tehokkuudessa, viestinnässä, sekä ulkoasussa. Käytettävyyden näkökulmasta johtuen kehitystyön painopisteen oli tarkoitettu olevan käyttöliittymien parissa. Mobiilisovelluksen muiden osapuolten, kuten palvelinlogiikan ja tietokannan toteutus päätettiin pitää yksinkertaisena, mutta kuitenkin toimivana, selkeänä, sekä lähtökohtaisesti sovelluskokonaisuutta tukevana.

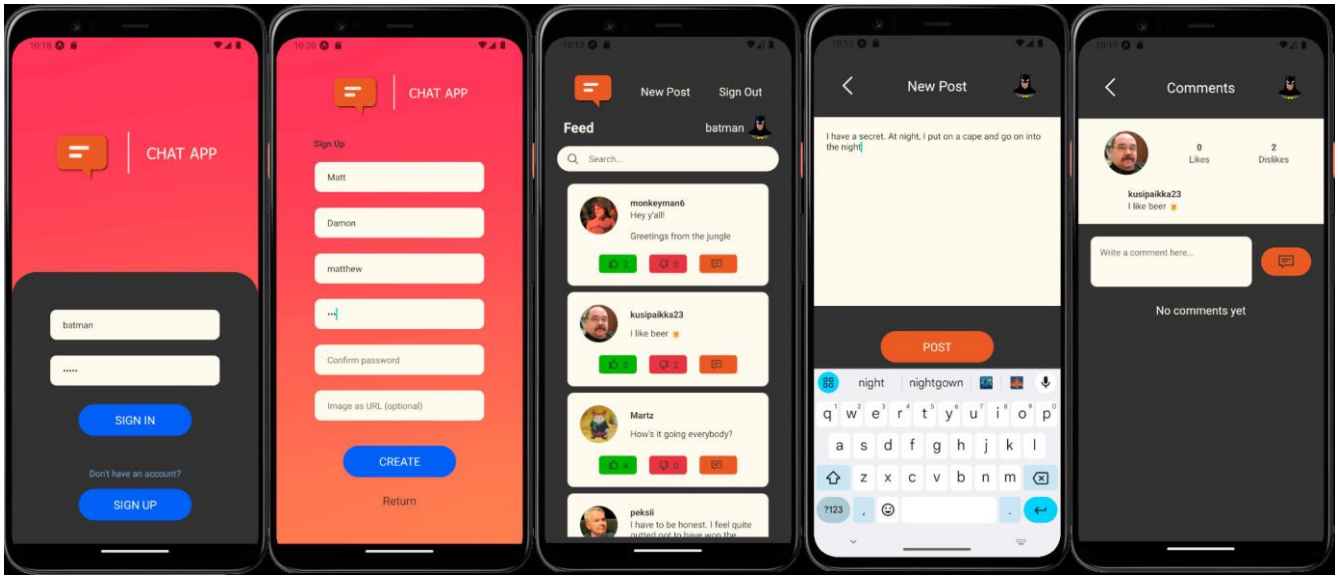
Mobiilisovelluksen toteutusta katsoessa voidaan todeta, että se oli laadukas ja tavoitteisiin nähden onnistunut. Painopiste pidettiin kehitysaikana hyvin pitkälti sovelluksen käyttöliittymien puolella ja sovelluskokonaisuus mukaili siihen nähden sille asetettuja tavoitteitaan. Käyttöliittymien lopulliset IOS-käyttöjärjestelmään räätälöidyt versiot ovat nähtävissä kuviossa 18.



Kuvio 18 Sovelluksen käyttöliittymät IOS-käyttöjärjestelmässä

Käytettävyyttä osoitettiin käyttöliittymissä selkeällä ja yksinkertaisella sisällöllä. Ulkoasu rakennettiin yhdistävällä väriteemalla, sekä käyttäjää ohjaavilla teksteillä, ikoneilla ja painikeanimaatioilla. Toiminnallisuudet tuotiin sovelluksessa esille selkeästi ja saatavuudeltaan helpoiksi. Toimintoja ajavat painikkeet sijoitettiin käyttöliittymissä selkeästi näkyviksi ja vahvoilla teemaan sopivilla väreillä erottuviksi. Käyttäjää ohjaavat tekstit tuotiin komponenteissaan esiin

vahvalla fontilla ja ytimekkäällä sisällöllä. Käyttöliittymien lopulliset Android-käyttöjärjestelmään räätälöidyt versiot ovat nähtävissä kuviossa 19.



Kuvio 19 Sovelluksen käyttöliittymät Android-käyttöjärjestelmässä

Sovelluksen toiminnallisuudet osoittivat käytettävyyttä toimimalla tehokkaasti johdonmukaisen palvelinlogiikan kanssa. Liikkuminen näkymien välillä oli helppoa ja nopeaa. Jokainen suunnitteluvaiheessa määritelty päätoiminto oli toteutettavissa noin kolmen painalluksen päässä, jos ei oteta huomioon syöttökenttiin kirjoittamisesta johtuvia painalluksia. Laadukasta käytettävyyttä osoitettiin myös sovelluksen viestinnässä. Toiminnallisuuksia aktivoivia painikkeita oli ehostettu animaatiomaisella tyylivaihteluilla. Virhetilanteissa oli toteutettu selkeät ilmoitukset toimintojen suorittamisen yhteydessä. Käytettävyyden osoituksena voidaan myös nähdä natiivisovelluksen saatavuus ja mukautuvuus IOS-, sekä Android käyttöjärjestelmille.

Tämän tutkimustyön tuloksista voitiin todeta, että mobiilisovelluksen luomisen yhteydessä laadukkaaseen käytettävyyteen pyrkiminen vaati kokonaisuutena monivaiheisen prosessin. Tämän tutkimuksen aikana prosessi koostui suunnittelun eri vaiheista, sekä sovelluksen käytännön toteutuksen eri vaiheista.

Käytettävän sovelluksen pohjaa rakentaessa oli tärkeää hahmottaa sovelluksen toiminnallisuudet heti suunnitteluvaiheessa. Tärkeinä työkaluina käytössä olivat käyttäjäpolut, joiden avulla oli

mahdollista hahmottaa keskeiset toiminnot käyttäjän näkökulmasta, keskustelusovellus-tyyppisessä kokonaisuudessa. Käyttäjäpolkujen perusteella hahmottui sovellukseen tarvittavat käyttöliittymät ja navigointi niiden välillä, joiden pohjalta oli mahdollista suunnittella käyttöliittymät mockup-versioina. Mockup-versioiden mallintamisen aikana selvennettiin käyttöliittymien sisältö, kuvitus, värimaailma ja asettelu.

Tehokkaan, suoraviivaisen, sekä helppolukuisen toimivuuden tiedettiin olevan osa hyvää käytettävyyttä, joten sovelluksen toimintojen pohjaksi rakennettiin yksinkertainen ja tehokas palvelinlogiikka. Palvelinkokonaisuus rakennettiin GraphQL-kyselykielellä yhdessä MongoDB-tietokannan kanssa. Kyseiset teknologiat valittiin, koska ne osoittivat helppokäyttöisyyttä ja yhteensopivuutta tämän sovelluskokonaisuuden kanssa.

Käyttöliittymät ohjelmoitiin React Native -kirjaston teknologiaa käyttäen yhteistyössä Expo-komentoliittymän kanssa. React Native -kirjasto tarjosi mahdollisuuden natiivimobiilisovelluksen rakentamiseen riippumatta käyttöjärjestelmästä. Lisäksi React Native -kirjasto lisäosineen tarjosi lukuisia valmiita komponentteja ja työkaluja käyttöliittymien rakentamiseen, joita tässä tutkimustyössä paljon hyödynnettiin.

Expo-komentoliittymä valittiin natiivisovelluksen kehitysalustaksi, koska se mahdollisti komentoliittymien avaamisen kehitysaikana puhelimessa ja tietokoneella Android-emulaattorissa. Sovelluksen renderöinti tapahtui puhelimessa ja emulaattorissa Expo Go -puhelinsovelluksen kautta, näyttäen kehitysaikana tehdyt muutokset näytöllä reaaliaikaisesti. Käyttöliittymien reaaliaikainen seuraaminen oikean puhelimen näytöllä todettiin sovelluksen kehityksen kannalta elintärkeäksi, koska se mahdollisti toimintojen ja yleistuntuman testaamisen käyttöliittymissä kehitysaikana.

Toimintojen suorittamisen yhteydessä panostettiin myös viestintään käyttäjän kanssa, koska käytettävyyden näkökulmasta ymmärrettiin tärkeäksi sisällyttää käyttöliittymiin virheilmoituksia ja muita ohjaavia tekstikomponentteja mahdollisissa poikkeustilanteissa.

5 Pohdinta

5.1 Johtopäätökset

Opinnäytetyön tarkoituksena oli tutkia natiivimobiilisovelluksen kehittämistä käytettävyyden näkökulmasta React Native -teknologialla. Tutkimuskysymyksiä asetettiin yhteensä kolme kappaletta, ja tutkimustyön päätteeksi kaikkiin osattiin vastata. Osattiin tunnistaa käytettävän mobiilisovelluksen rakentamisen eri tekijät, sekä vaiheet. Lisäksi syvennyttiin React Native -teknologiaan käyttöliittymien ohjelmoinnissa ja osattiin kertoa, että miksi se on hyvä työkalu käytettävän mobiilisovelluksen luomiseen.

Yhteenvetona saaduista tuloksista voitiin todeta, että käytettävyydeltään laadukas mobiilisovellus rakentuu monesta tekijästä. Ollakseen käytettävä, sovelluksen on oltava toiminnallisuudeltaan helppolukuinen ja tehokas. Käyttäjälle tarkoitetut toiminnot on oltava helposti saavutettavissa, sekä nopeasti toteutettavissa.

Helppolukuisuutta ja saavutettavuutta varmistetaan sovelluksessa perusteellisella suunnittelulla. Suunnittelu toteutetaan sisällön asettelun, väriteeman, ohjaavien tekstien, sekä kuvioiden kautta. Sisältö on hyvä perustaa yleisesti käyttöliittymissä toistuviin normeihin, joihin käyttäjä on jo entuudestaan tottunut. Käyttäjää on hyvä ohjata käyttöliittymissä sovelluksen toimintoihin selkeästi, esimerkiksi hyvin erottuvien painikkeiden ja kuvakkeiden avulla. Sovelluksen on myös hyvä automatisoida toimintaa mahdollisimman paljon, esimerkiksi toiseen näkymään navigoimalla toiminnallisuuden suorittamisen yhteydessä. Samalla viestintä käyttäjän ja sovelluksen välillä on oltava selkeää ja oikea-aikaista, esimerkiksi toimintojen virhetilanteissa.

React Native -kirjaston teknologia osoittautui hyväksi alustaksi nimenomaan nativiimobiilisovelluksen kehittämiseen. Kirjasto tarjosi laajan dokumentaation, sekä valmiiden komponenttien ja työkalujen kautta melko vaivattoman tavan kehittää mobiilisovellusta, kun painopiste oli käytettävyydessä. Suunnitteluvaiheen mockup-versiot oli melko helppoa toteuttaa React Native -kirjastoa käyttäen, mikä säästi paljon resursseja, kuten aikaa. Lisäksi suunnitteluvaiheen käyttäjäpolutuksen tarkka navigointi oli tehokasta ja yksinkertaista toteuttaa React Native -sovelluksessa.

Ajoittain oli haasteita kahden käyttöjärjestelmän samanaikaisessa kehittämisessä, kun kaikki React Native -kirjaston tarjoamat työkalut eivät aina olleet tuettuina molempiin. React Native todisti kuitenkin joustavuutensa, koska kaikki kohdatut ongelmat olivat ratkaistavissa vaihtoehtoisilla työkaluilla. Lisäksi palvelimen yhdistäminen Expo-komentoliittymällä luotuun React Native -sovellukseen tarjosi aluksi haasteita, kun Apollo Client -kirjaston asentaminen vaati jonkin verran sille kohdennettua konfiguraatiota.

5.2 Luotettavuus

Idea tutkimukselle tuli hyvin pitkälti kehittäjän omasta mielenkiinnosta käytettävyyteen, mobiilisovelluksiin, sekä työssä käytettyihin teknologioihin. Teknologioiden valintaan vaikutti paljon niiden suosio ja kehittäjän aikaisempi tuntemus, mutta myös yksilön motiivi syventyä ja kehittyä niiden parissa. Tutkimuksen ansiosta kehittäjän ymmärrys ja osaaminen käytettävän mobiilisovelluksen kehittämisestä, sekä sovelluskokonaisuuksien hahmottamisesta kasvoi huomattavasti.

Tietoperustaa kerättiin opinnäytetyötä varten runsaasti sitä varten käytettyjen teknologioiden virallisista dokumentaatioista, sekä muista aiheesta kirjoitetuista tietolähteistä. Tietolähteiden valinnassa pyrittiin varmistamaan parhaan mukaan tiedon autenttisuus ja luotettavuus. Käytettyjä lähteitä oli määrällisesti paljon, koska haluttiin varmistaa tietojen yhteneväisyys. Lähteet olivat muodoltaan suurimmaksi osaksi verkkolähteitä, koska aiheeseen liittyvä tekninen dokumentaatio painottui pääosin verkkosivuille. Ulkopuolisen tietoperustan lisäksi tutkimuksessa nojattiin myös osittain kehittäjän omakohtaiseen kokemukseen perustuvaan tietoon.

On hyvä huomioida, että tämän soveltavan tutkimuksen tulokset todistivat yhden tavan luoda käytettävä mobiilisovellus. Tämä ei tietenkään tarkoita sitä, etteikö samaan tulokseen pääsisi muitakin reittejä pitkin. Lisäksi on hyvä tiedostaa, että tulokset perustuvat kehittäjän omiin havaintoihin ja kokemuksiin. Kehittäjällä oli kuitenkin ennen tutkimustyötä suurimmaksi osaksi melko hyvä tuntemus siinä käytettyihin teknologioihin ja kehitysmenetelmiin. Tuloksia voidaan hyödyntää jatkossa ainakin React Native -kehyksellä toteutetussa sovelluskehityksessä, GraphQL-kyselykielellä toteutetussa palvelinkehityksessä, sekä myös yleisesti käyttöliittymä- ja käytettävyyssuunnittelussa.

5.3 Jatkokehitys

Käytettävyyden näkökulmasta jatkokehityksenä sovellukselle olisi hyvä toteuttaa käyttäjätestaus, joka sisältäisi hyvin suunnitellut käytännön testit, sekä perusteellisen raportoinnin. Testauksesta saatu data antaisi varmasti hyödyllistä ja suuntaa näyttävää tietoa sovelluksen käytettävyydestä, sekä myös mielenkiintoista palautetta oikeiden käyttäjien suunnalta. Testauksia ei tämän tutkimustyön aikana toteutettu, koska työn suuruus haluttiin pitää maltillisissa mitoissa rajallisista aikatauluista johtuen. Muutamia kevytmielisiä kokeiluja tutuilla ihmisillä suoritettiin, mutta ei mitään virallisesti raportoitavaa.

Mobiilisovelluksen teknistä jatkokehitystä ja ylläpitoa olisi tarpeen jatkaa, jos sen haluaisi pitää ajan tasalla ja toimintakykyisenä. Sen toiminnassa käytetyt työkalut tarvitsevat esimerkiksi ajoittaista päivittämistä. Sisällöltään sovellus oli lopulta melko yksinkertainen ja suppea, kun se rajattiin toiminnallisuuksiltaan pelkästään suunniteltujen käyttäjäpolkujen ympärille. Sovellusta voisi täydentää toiminnoilla, kuten vaikka profiilin muokkauksella, sekä julkaisun ja kommentin muokkaamisella ja poistamisella.

Palvelinarkkitehtuuri jätettiin myös suhteellisen yksinkertaiseksi. Palvelimen toimintaa olisi hyvä esimerkiksi laajentaa, tehostaa, sekä testata. React Native -sovelluksen puolella olisi myös paikallaan suorittaa toiminnallisuudet tarkistavat testaukset, mitä kautta olisi mahdollista paikantaa mahdolliset ohjelmointivirheet jatkokehityksessä. Tietoturvaan ei keskitytty kehityksen aikana lainkaan, mitä tulisi myös kehittää sovelluksessa, jos sen haluaisi joskus viedä julkaisuun.

Lähteet

Abbas, A. 2023. Building a GraphQL API with Node.js, Apollo Server. Artikkelijulkaisu Mediumin verkkosivuilla. 18.9.2023. Viitattu 12.4.2024.
https://medium.com/@aqeelabbas3972/building-a-graphql-api-with-node-js-apollo-server-da7213237310

Acharya, V. 2023. Mastering Asynchronous Programming in Node.js. Tutoriaalijulkaisu Mediumin verkkosivuilla. 2.7.2023. Viitattu 8.4.2023.
https://medium.com/@vishwasacharya/mastering-asynchronous-programming-in-node-js-174cb75272a0

Alam-Naylor, S. & Fateh, D. 2021. What is GraphQL? What a GraphQL API is and how to use it. Blogikirjoitus Contentfulin verkkosivuilla. 13.12.2021. Päivitetty 8.2.2024. Viitattu 9.4.2024.
https://www.contentful.com/blog/what-is-graphql/

Bashir, A. N.d. Static vs dynamic languages. Dokumentaatio eduactiven verkkosivuilla. Viitattu 28.2.2024.
https://www.educative.io/answers/static-vs-dynamic-languages

Borozenets, M. 2022. React Native Init vs Expo 2022: What Are the Differences? Artikkelikirjoitus Fulcrum yrityksen verkkosivuilla. 22.4.2022. Viitattu 4.3.2024.
https://fulcrum.rocks/blog/react-native-init-vs-expo

Botelho, B. & Gillis, A. N.d. Artikkelijulkaisu TechTargetin verkkosivuilla. Päivitetty maaliskuussa 2023. Viitattu 12.4.2024.
https://www.techtarget.com/searchdatamanagement/definition/MongoDB

Budzinski, M. N.d. What Is React Native? Complex Guide for 2023. Blogikirjoitus Netgurun verkkosivuilla. Viitattu 4.3.2024.
https://www.netguru.com/glossary/react-native

Bulat, R. 2020. Apollo Server: An Introduction to the Leading GraphQL Server. Artikkelijulkaisu Mediumin verkkosivuilla. 11.5.2020. Viitattu 12.4.2024.
https://rossbulat.medium.com/apollo-server-an-introduction-to-the-leading-graphql-server-35097b626dfb

Desktop vs Mobile vs Tablet Market Share Worldwide. 2024. Taulukko StatCounterin verkkosivuilla. Viitattu 27.2.2024.
https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet

Dziuba, A. 2024. When and Why to Use Node.js: An In-Depth Guide for 2024. Blogikirjoitus Relevantin verkkosivuilla. 6.8.2023. Päivitetty 12.1.2024. Viitattu 9.4.2024.
https://relevant.software/blog/why-and-when-to-use-node-js/

Gallo, K. 2023. What Is I/O (Input/Output)? Artikkelijulkaisu Built Inin verkkosivuilla. 2.3.2023. Viitattu 9.4.2024.
https://builtin.com/hardware/i-o-input-output

GraphQL-palvelin. N.d. Helsingin avoimen yliopiston oppimateriaali Full Stack open-verkkosivuilla. Päivitetty 24.9.2023. Viitattu 12.4.2023.
https://fullstackopen.com/osa8/graph_ql_palvelin

Hall, J. 2022. Getting Started with MongoDB & Mongoose. Artikkelijulkaisu MongoDB Developerin verkkosivuilla. 7.4.2022. Päivitetty 19.5.2022. Viitattu 16.4.2024.
https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/

Hanna, K. T. & Wigmore, I. N.d. What is a mobile app (mobile application)? Blogikirjoitus. TechTarget. Päivitetty helmikuussa 2023. Viitattu 27.2.2024.
https://www.techtarget.com/whatis/definition/mobile-app

How Node.js overcome the problem of blocking of I/O operations? N.d. Artikkelijulkaisu GeeksForGeeksin verkkosivuilla. Päivitetty 16.2.2022. Viitattu 9.4.2024.
https://www.geeksforgeeks.org/how-node-js-overcome-the-problem-of-blocking-of-i-o-operations/

Hufford, B. 2022. What is a Mockup? (+How to Create a Mockup in 2022). Artikkelijulkaisu Clique Studiosin verkkosivuilla. 5.5.2022. Viitattu 16.4.2024.
https://cliquestudios.com/mockups/

Introducing JSX. Why JSX? N.d. Dokumentaatio legacy.react.js verkkosivuilla. Viitattu 4.3.2024.
https://legacy.reactjs.org/docs/introducing-jsx.html

Introduction to Apollo Client. N.d. Dokumentaatio ApolloGraphQL:n verkkosivuilla. Viitattu 12.4.
https://www.apollographql.com/docs/react/

Introduction to Apollo Server. N.d. Dokumentaatio ApolloGraphQL:n verkkosivuilla. Viitattu 12.4.2024.
https://www.apollographql.com/docs/apollo-server/

Introduction to GraphQL. N.d. Dokumentaatio GraphQLn verkkosivuilla. Päivitetty 31.3.2024. Viitattu 9.4.2024.
https://graphql.org/learn/

Introduction to Node.js. N.d. Dokumentaatio nodejs:n verkkosivuilla. Viitattu 8.4.2024.
https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

Introduction to React Native. N.d. Helsingin avoimen yliopiston oppimateriaali Full Stack open-verkkosivuilla. Päivitetty 26.2.2024. Viitattu 4.3.2024.
https://fullstackopen.com/en/part10/introduction_to_react_native

JavaScript. N.d. Dokumentaatio MDN Web Docsin verkkosivuilla. Päivitetty syyskuussa 2023. Viitattu 28.2.2024.
https://developer.mozilla.org/en-US/docs/Web/JavaScript

Jordana, A. 2024. What Is JavaScript: A Beginner's Guide to the Basics of JS. Dokumentaatio Hostinger Tutorialsin verkkosivuilla. 26.2.2024. Viitattu 28.2.2024. https://www.hostinger.com/tutorials/what-is-javascript

JSON and BSON. N.d. Dokumentaatio MongoDB:n verkkosivuilla. Viitattu 12.4.2024. https://www.mongodb.com/json-and-bson

Kukic, A. & Vlaeva, S. 2021. MongoDB and Mongoose: Compatibility and Comparison. Artikkelijulkaisu MongoDB Developerin verkkosivuilla. 25.11.2021. Päivitetty 2.4.2024. Viitattu 16.4.2024. https://www.mongodb.com/developer/languages/javascript/mongoose-versus-nodejs-driver/

Käytettävyys ohjelmistokehityksessä: Miksi ja miten sitä tutkitaan? 2023. Blogikirjoitus Haltun verkkosivuilla. 18.10.2023. Viitattu 27.2.2024. https://www.haltu.fi/blogi/kaytettavyys

Long, M. 2023. Why JavaScript is Single Threaded? Blogikirjoitus Groove Technologyn verkkosivuilla. 21.7.2023. Viitattu 28.2.2024. https://groovetechnology.com/blog/why-javascript-is-single-threaded/

Michael, N. 2023. The 4 best React Native routing libraries. Blogikirjoitus LogRokcketin verkkosivuilla. 21.3.2023. Viitattu 4.4.2024. https://blog.logrocket.com/best-react-native-routing-libraries/

MongoDB Atlas Tutorial. N.d. Dokumentaatio MongoDB:n verkkosivuilla. Viitattu 12.4.2024. https://www.mongodb.com/basics/mongodb-atlas-tutorial

Nasir, H. N.d. What is the difference between React Router DOM & Native version? Dokumentaatio eduactiven verkkosivuilla. Viitattu 4.4.2024. https://www.educative.io/answers/what-is-the-difference-between-react-router-dom-native-version

Nkemchor, E. 2023. Mockups in UX: Definition and Best Practises. Artikkelijulkaisu UXtweakin verkkosivuilla. 17.4.2023. Viitattu 16.4.2024. https://blog.uxtweak.com/mockup-ux/

Node.js – NPM. N.d. Dokumentaatio Tutorials Pointin verkkosivuilla. Viitattu 9.4.2024. https://www.tutorialspoint.com/nodejs/nodejs_npm.htm

Object Data Model. N.d. Dokumentaatio Capterran verkkosivuilla. Viitattu 16.4.2024. https://www.capterra.com/glossary/object-data-model/

Overview of React.js. N.d. Dokumentaatio Patternsin verkkosivuilla. Viitattu 29.2.2024. https://www.patterns.dev/react/

Pasanen, A. 2022. Verkkosivujen Käyttökokemuksen parantaminen. Opinnäytetyö, AMK. Kaakkois-Suomen ammattikorkeakoulu, liiketalouden tutkinto-ohjelma. Viitattu 17.4.2024. https://www.theseus.fi/bitstream/handle/10024/747770/aku_pasanen.pdf;jsessionid=B2CD5EC744E2CCC1D77C368CC3B7D710?sequence=2

Patrun, L. 2023. Setup React Native project with Expo. Tutoriaalijulkaisu Mediumin verkkosivuilla. 2.10.2023. Viitattu 4.3.2024.
https://medium.com/@luka.patrun/setup-react-native-project-with-expo-8997e1d6f12f

Prototype-based programming. N.d. Dokumentaatio MDN Web Docsin verkkosivuilla. Päivitetty kesäkuussa 2023. Viitattu 28.2.2024.
https://developer.mozilla.org/en-US/docs/Glossary/Prototype-based_programming

Raji, A. 2022. Difference between GraphQL and Apollo-GraphQL. Blogikirjoitus Devin verkkosivuilla. 19.12.2022. Viitattu 12.4.2024.
https://dev.to/yemiklein/difference-between-graphql-and-apollo-graphql-3p08

Rajput, A. N.d. 6 Issues That User Path Analysis Can Help Uncover. Blogijulkaisu Moengagen verkkosivuilla. Päivitetty 17.7.2023. Viitattu 17.4.2024.
https://www.moengage.com/blog/issues-user-path-analysis-can-uncover/

React Introduction. N.d. Dokumentaatio GeeksforGeeksin verkkosivuilla. Päivitetty 25.1.2024. Viitattu 29.2.2024.
https://www.geeksforgeeks.org/reactjs-introduction/

React Router. N.d. Dokumentaatio GeeksforGeeksin verkkosivuilla. Päivitetty 15.3.2024. Viitattu 4.4.2024.
https://www.geeksforgeeks.org/reactjs-router/

Semah, B. 2022. What Exactly is Node.js? Explained for Beginners. Tutoriaalijulkaisu freeCodeCampin verkkosivuilla. 5.12.2022. Viitattu 8.4.2024.
https://www.freecodecamp.org/news/what-is-node-js/

Schemas and Types. N.d. Dokumentaatio GraphQLn verkkosivuilla. Päivitetty 31.3.2024. Viitattu 12.4.2024.
https://graphql.org/learn/schema/

Shaban, L. 2023. React Router: A step-by-step quide. Tutoriaalijulkaisu Mediumin verkkosivuilla. 21.10.2023. Viitattu 4.4.2024.
https://luqmanshaban.medium.com/react-router-a-step-by-step-guide-4c5ec964d2e9

Smith, A. 2017. Usability First – Why Usability Design Matters to UI/UX Designers. Artikkelijulkaisu UX Planetin verkkosivuilla. 7.12.2017. Viitattu 27.2.2024.
https://uxplanet.org/usability-first-why-usability-design-matters-to-ui-ux-designers-9dfb5580116a

State: A Component's Memory. Meet your first Hook. N.d. Dokumentaatio React.dev verkkosivuilla. Viitattu 29.2.2024.
https://react.dev/learn/state-a-components-memory

Tietojen tallettaminen MongoDB-tietokantaan. N.d. Helsingin avoimen yliopiston oppimateriaali Full Stack open-verkkosivuilla. Päivitetty 9.2.2024. Viitattu 12.4.2024.
https://fullstackopen.com/osa3/tietojen_tallettaminen_mongo_db_tietokantaan

What is GraphQL? 2019. Artikkelikirjoitus Red Hatin verkkosivuilla. 8.1.2019. Viitattu 9.4.2024. https://www.redhat.com/en/topics/api/what-is-graphql

What Is MongoDB? N.d. Dokumentaatio MongoDB:n verkkosivuilla. Viitattu 12.4.2024. https://www.mongodb.com/company/what-is-mongodb

What Is Node.js and Why You Should Use It. N.d. Artikkelijulkaisu Kinstan verkkosivuilla. Päivitetty 7.11.2023. Viitattu 9.4.2024. https://kinsta.com/knowledgebase/what-is-node-js/

What's the Difference Between Web Apps, Native Apps, and Hybrid Apps? N.d. Dokumentaatio AWS:n verkkosivuilla. Viitattu 27.2.2024. https://aws.amazon.com/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/

Why Should You Use a Router in React.j? N.d. Artikkelijulkaisu Simplilearnin verkkosivuilla. Päivitetty 20.2.2023. Viitattu 4.4.2024. https://www.simplilearn.com/tutorials/reactjs-tutorial/routing-in-reactjs

Yusufu, E. 2024. React Native Navigation: Tutorial with examples. Blogikirjoitus LogRocketin verkkosivuilla. 18.1.2024. Viitattu 4.4.2024. https://blog.logrocket.com/react-native-navigation-tutorial/