LAB University of Applied Sciences

Mobiilisovellus metsässä liikkujille

Mobiilisovellus luonnonpaikkojen kartoitukseen

LAB-ammattikorkeakoulu
Insinööri (AMK)
2025
Tomi Heikkinen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Tomi Heikkinen	Opinnäytetyö, AMK	2025
	Sivumäärä 37	

Työn nimi

Mobiilisovellus metsässä liikkujille

Mobiilisovellus luonnonpaikkojen kartoitukseen

Tutkinto ja koulutusala

Insinööri (AMK), tieto- ja viestintätekniikka

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

Tiivistelmä

Tässä opinnäytetyössä kehitettiin hybridimobiilisovellus. Sovellus tarjosi käyttäjille mahdollisuuden kartoittaa omia mielenkiinnon kohteita, sekä sienestys- ja marjastuspaikkoja luonnosta. Opinnäytetyön tarkoituksena oli tuottaa asennettava julkaisu Android alustalle.

Opinnäytetyö tarkasteli React Nativen ja Reduxin käyttöä hybridisovelluskehityksessä, keskittyen erityisesti funktionaaliseen kehitykseen. Työssä tutkittiin myös tekoälyn mahdollisuuksia ja haasteita parantamaan käyttäjäkokemusta ja monipuolistamaan palvelua.

Opinnäytetyön tuloksena on MVP (Minimum viable product) -sovellus Androidille, joka soveltuu käyttäjätestaukseen, sekä kysynnän ja jatkokehitysmahdollisuuksien selvittämiseen.

Asiasanat

React Native, Redux, hybridikehitys, sovelluskehitys

Abstract

Author(s)	Type of Publication	Published
Tomi Heikkinen	Thesis, UAS	2025
	Number of Pages 37	

Title of Publication

A Mobile Application for Nature Enthusiasts

A Mobile Application for Mapping Nature Spots

Degree, Field of Study

Engineer (UAS), Information and Communication Technology

Organisation of the client (if the thesis work is commissioned by another party)

Abstract

In this thesis, a hybrid mobile application was developed. The application enables users to map their personal interests as well as mushroom and berry picking locations in nature. The aim of the thesis was to produce an installable release for the Android platform.

The study examined the use of React Native and Redux in hybrid application development, with a particular focus on functional programming approaches. The potential and challenges of artificial intelligence for enhancing user experience and diversifying services were also explored.

The result of the thesis is an MVP (Minimum Viable Product) application for Android, which is suitable for user testing, and for assessing demand and opportunities for further development.

Keywords

React Native, Redux, Hybrid development, Software development

Sisällys

1 Johdanto

Opinnäytetyössä kehitetään hybridi mobiilisovellus Android-käännöksellä. Sovelluksella tallennetaan luonnonpaikkoja karttaan, otetaan valokuvia kohteista sekä lisätään merkintöihin muistiinpanoja. Lokaalin merkintöjen tallennuksen lisäksi sovelluksessa käytetään rajapintayhteyttä karttojen hakuun puhelimen paikannuksen avulla.

Markkinoilla on muutamia sovelluksia ja palveluja kuten Satokausi, Mushroom Spot ja iNaturalist. Jokainen näistä toimii hieman poikkeavin tavoin. Satokausi tarjoaa julkista dataa ja suunnistuskarttoja, mistä mahdollisesti löytyy marjoja ja sieniä. Mushroom Spot on moderni tyylikäs mobiilisovellus, jossa on julkisia sienipaikkoja sekä mahdollisuus tallentaa omia paikkoja ja tutkia sienien tietoja. iNaturalist on kansainvälinen palvelu, joka keskittyy erilaisten löytöjen kuvaamiseen ja tieteellisen tiedon hakuun.

Opinnäytetyön sovelluksen on tarkoitus laajentaa tarjontaa niin, että se yhdistää käytännössä Satokauden sieni-, mustikka- ja puolukkakarttapalvelun idean Mushroom Spotin modernimpaan käytettävyyteen. Opinnäytetyön sovelluksen tavoitteena ei ole tarjota julkista dataa, vaan kerääjät saavat omat löytöpaikat itselleen talteen.

Opinnäytetyön teoriaosuudessa tutkitaan valittua toteutusteknologiapinoa, sekä tekoälyn mahdollisuuksia ja toteutusosassa keskitytään sovelluksen suunnittelun ja toteutuksen kuvaamiseen. Työ toteutetaan tekijän omasta toimeksiannosta. Jatkotavoitteena on kehittää sovellus viimeisteltynä Android- ja iOS-markkinoille, lisäominaisuuksineen implementoituina.

2 Käytetyt teknologiat

2.1 React ja React Native

React kehitettiin Metan (entinen Facebook) sisällä vuonna 2011 parantamaan Facebook Ads -sovelluksen koodin huollettavuutta. Jordan Walke loi Reactin prototyypin, ja se julkaistiin avoimen lähdekoodin projektina vuonna 2013. (RisingStack Engineering 2025.) React on deklaratiivinen JavaScript-kirjasto, joka mahdollistaa modulaaristen käyttöliittymien rakentamisen virtuaalisen DOM:in avulla. Virtuaalinen DOM on nopea ja optimoi käyttöliittymän päivitykset tehokkaasti. (Green 2023.)

React Native julkaistiin virallisesti Facebookin F8-konferenssissa maaliskuussa 2015. Ensimmäisen julkaisun aikaan React Native tuki vain iOS-käyttöjärjestelmää, mutta syyskuussa 2015 julkaistiin myös Android-tuki. (RisingStack Engineering 2025.)

React Native, kuten React käytti alun perin JSX (JavaScript XML) JavaScript-syntaksin laajennusta (The Codest 2025). JSX:llä kirjoitetaan HTML-tyyppistä koodia, joilla voidaan määritellä omia React-komponentteja. JSX mahdollistaa selkeän HTML-tyylisen koodin JavaScriptin lisäksi samassa tiedostossa.

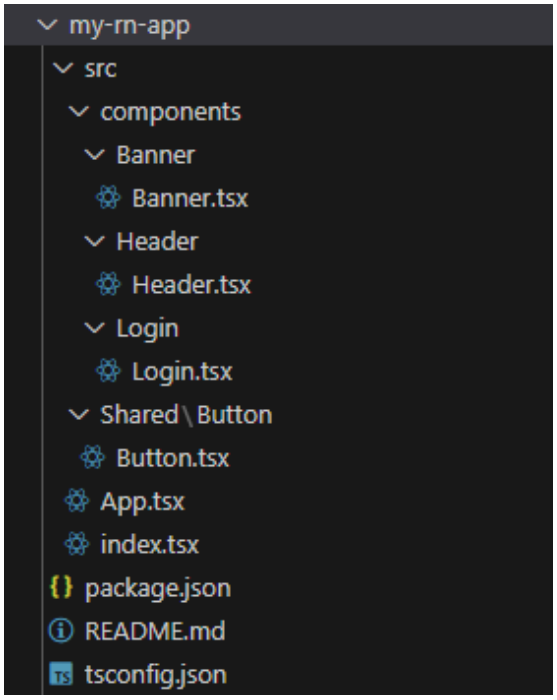
React Native on siirtynyt käyttämään vakiona TypeScript kieltä (React Native 2025a). TypeScript on Microsoftin, Anders Hejlsbergin johdolla kehittämä JavaScript-kielen ylijoukko. TypeScript tyypittää JavaScriptin ja tämä mahdollistaa ajonaikaisten virheiden minimoimisen normaaliin JavaScriptin dynaamiseen tyypitykseen nähden. (Austin 2025.) Kaikki TypeScript-koodi kuitenkin käännetään JavaScriptiksi julkaisuvaiheessa. TypeScript-laajennuksen myötä JSX-tiedostot ovat TSXiä.

React ja React Native ovat jatkaneet kehitystään aktiivisesti vuosien varrella. Viimeisin merkittävä julkaisu, React 19, tapahtui joulukuussa 2024. Tämä versio toi mukanaan useita uusia ominaisuuksia, kuten Actions-toiminnallisuuden, joka yksinkertaistaa tilan päivitysprosessia asynkronisissa funktioissa. Lisäksi React 19 paransi staattista sivugenerointia ja toi tuen palvelinkomponenteille. (React v19 2024.)

2.1.1 Komponenttipohjainen kehitys

Komponenttipohjainen kehitys on keskeinen periaate React Native -sovellusten kehityksessä. Tämä lähestymistapa mahdollistaa modulaarisen ja uudelleenkäytettävän koodin, joka parantaa kehitystyön tehokkuutta ja sovelluksen ylläpidettävyyttä. React Native -komponentit voivat olla joko luokka- tai funktionaalisia komponentteja, joista jälkimmäiset ovat suosituimpia niiden suorituskyvyn ja yksinkertaisuuden vuoksi. Komponentit kirjoitetaan TSX:llä ja ulkoasun tyylit määritellään Flexboxilla.

Pitkäsen (2021, 10) mukaan komponentit toteutetaan modulaarisesti omiin tiedostoihinsa. Modulaarinen ja hierarkkinen tiedostorakenne mahdollistaa koodin paremman luettavuuden ja selkeän vianetsinnän. Pääkomponentissa kuten ”Header” voi olla modulaarisesti kaksi lapsikomponenttia ”Banner” ja ”Login”. Hierarkkisesti nämä sijaitsevat samassa tiedostopuussa sisäkkäin omina kansioinaan ja kansiot sisältävät komponentin tarvitsemat TSX-tiedostot (Kuva 1).



Kuva 1. Esimerkki modulaarisesta tiedostorakenteesta React Native -projektissa

Toisin kuin perinteisessä MVC (Model-View-Controller) arkkitehtuurissa, React Nativen komponenttipohjaisessa lähestymistavassa TSX-tiedostot sisältävät yleensä kaiken tarvitsemansa. Niin näkymän, kuin logiikan sekä rajapintakutsut. Laajemmissa sovellusprojekteissa rajapintakyselyt kuitenkin eristetään omaan erilliseen palvelukerrokseen (services), johon komponenttien kyselyt viittaavat. Puristisessa modulaarisessa ajattelussa painotetaan tyylien sisältyvän komponenttiin, mutta käytännössä käytetään myös jaettuja tyylitiedostoja helpomman keskitetyn teemoituksen aikaansaamiseksi.

Yleisen käytännön mukaan projekteissa käytetään myös niin kutsuttuja jaettuja komponentteja (shared components). Nämä ovat esimerkiksi painikkeita, syöttökenttiä tai muita näkymien yleisiä komponentteja, joita voidaan yhdistää mihin tahansa projektin osaan. Jaetun komponentin ”Button” voi esimerkiksi tehdä ja tyylitellä yhdistämällä React Nativen peruskomponentit: ”TouchtableOpacity” ja ”Text”. Näin jaettua ”Button” komponenttia voi käyttää kaikkialla projektissa, joka sisältää perinteisen painikkeen toiminnallisuuden (Kuva 2).

my-rn-app > src > components > Shared > Button.tsx > ...

```tsx
import React from "react";
import { TouchableOpacity, Text, StyleSheet } from "react-native";

interface ButtonProps {
  label: string;
  onPress: () => void;
}

const Button: React.FC<ButtonProps> = ({ label, onPress }) => {
  return (
    <TouchableOpacity style={styles.button} onPress={onPress}>
      <Text style={styles.buttonText}>{label}</Text>
    </TouchableOpacity>
  );
};

const styles = StyleSheet.create({
  button: {
    paddingVertical: 10,
    paddingHorizontal: 20,
    borderRadius: 5,
    backgroundColor: "#61dafb",
    marginTop: 10,
  },
  buttonText: {
    fontSize: 16,
    fontWeight: "bold",
    color: "white",
    textAlign: "center",
  },
});

export default Button;
```

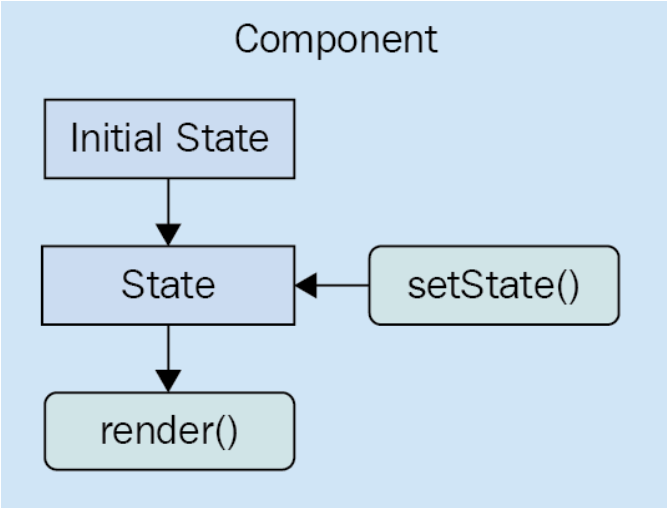
Kuva 2. Esimerkki jaetusta tilattomasta Button-komponentista

Esimerkissä nähdään TypeScriptillä toteutettu modulaarinen jaettu komponentti. Komponentille on määritelty oma tyyppimääritys (props), erillinen komponenttirakenne sekä tyylit StyleSheet.create -funktiolla, jolloin komponentti sisältää kaiken toiminnallisuuteen ja ulkoasuun liittyvät määritykset.

2.1.2 Tilalliset ja tilattomat komponentit

Kettusen (2023, 3) mukaan, komponentit voidaan jaotella kahteen päätyyppiin tilallisiin (stateful) ja tilattomiin (stateless). Tilalliset komponentit voivat olla joko luokkia tai funktionaalisia komponentteja, jotka käyttävät React Hookeja.

Tilalla (state) tarkoitetaan React-komponentin sisäistä tilaa, joka voi muuttua sovelluksen suorituksen aikana. Komponenttien muutokset käynnistävät automaattisesti komponentin uudelleenrenderöinnin. Kuvio 1 osoittaa miten tila voidaan muuttaa joko komponentin sisällä tai ulkopuolelta. (Boduch & Derks 2020, 84.)



Kuvio 1. Tilan asetus luokkakomponentissa (Boduch & Derks 2020)

Komponentille alustetaan alkuperäinen tila, jonka arvoa voidaan muuttaa setState-metodin avulla. Tämä metodi sekä päivittää tilan, että käynnistää komponentin uudelleenrenderöinnin Reactin virtuaali-DOMissa, jolloin näkymä päivittyy vastaamaan uutta tilaa.

Tilattomat komponentit eivät nimensä mukaan sisällä sisäistä tilaa. Tilattomat komponentit ovat puhtaita funktioita, jotka vastaanottavat vain props-parametreja ja palauttavat TSX:ää. (Kettunen, 2023.) Rakenteellisesti ne vähentävät toistuvan koodin määrää, mikä tekee sovelluksesta siistimmän ja helpommin luettavan (Kuva 3).

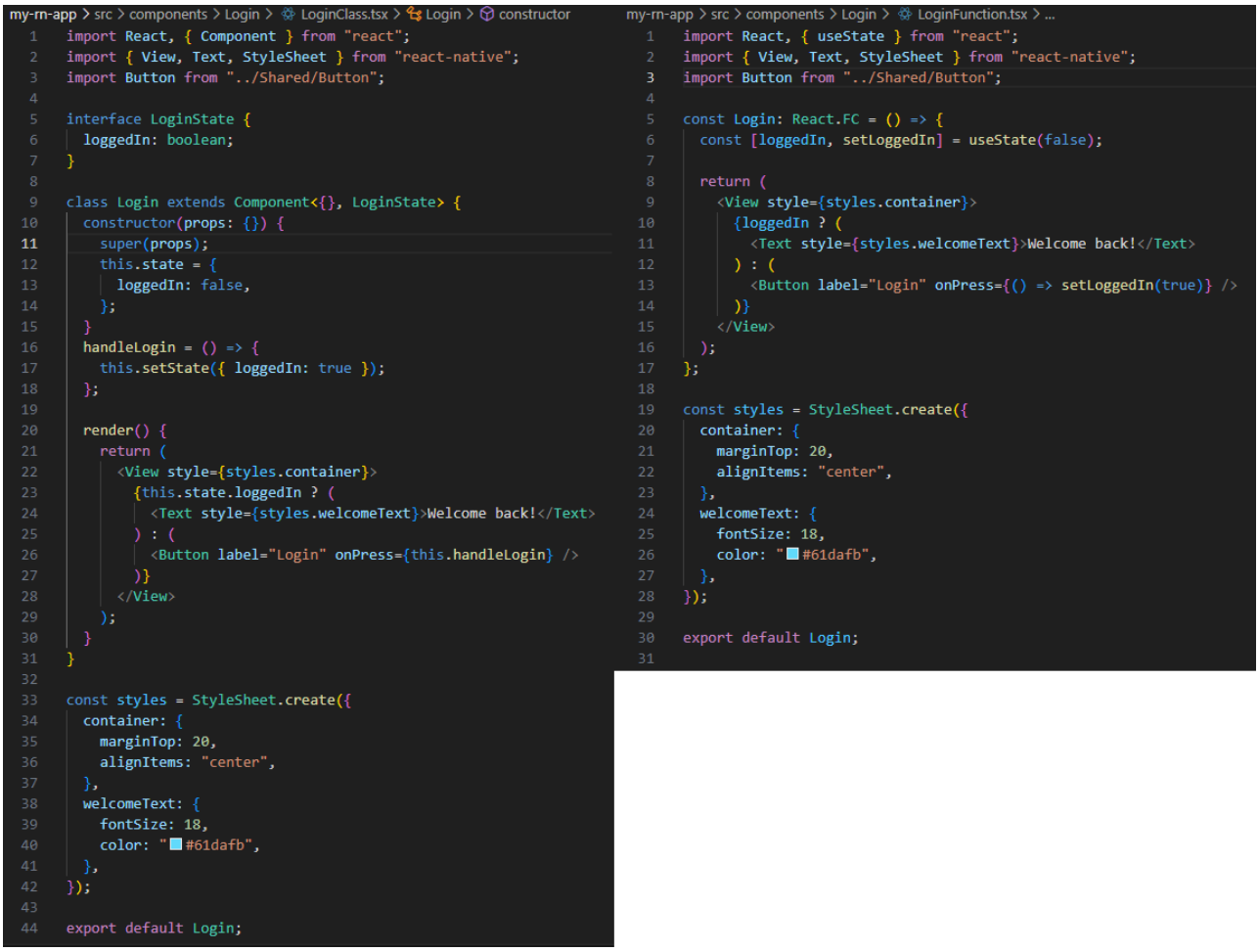
my-rn-app > src > components > Banner > Banner.tsx > ...

```tsx
import React from "react";
import { Text, StyleSheet } from "react-native";

interface BannerProps {
  message: string;
}

const Banner: React.FC<BannerProps> = ({ message }) => {
  return <Text style={styles.bannerText}>{message}</Text>;
};

const styles = StyleSheet.create({
  bannerText: {
    fontSize: 24,
    fontWeight: "bold",
    textAlign: "center",
    color: "#61dafb",
  },
});

export default Banner;
```

Kuva 3. Tilaton Banner-lapsikomponentti

Koodirivillä 4 komponentille määritellään ensin TypeScript-rajapinta, jonka avulla varmistetaan, että komponentti vastaanottaa aina oikeantyyppisen viestin. Itse komponentti on kirjoitettu funktionaalisena komponenttina, joka palauttaa tyylitellyn Text-elementin. Toteutus osoittaa, kuinka yksinkertaisimmillaan tilaton komponentti voi vastaanottaa datan ulkopuolelta, esittää sen ruudulla ja pysyä samalla rakenteeltaan hyvin selkeänä.

Tilalliset komponentit hallinnoivat omaa tilaansa. Luokkakomponenteissa tila hallinnoidaan this.state-objektin ja setState-metodin avulla, kun taas funktionaalisissa komponenteissa käytetään useState-hookkia (Kuva 4). Tilalliset komponentit reagoivat elinkaaren tapahtumiin joko luokan elinkaaren metodeilla tai useEffect-hookilla.



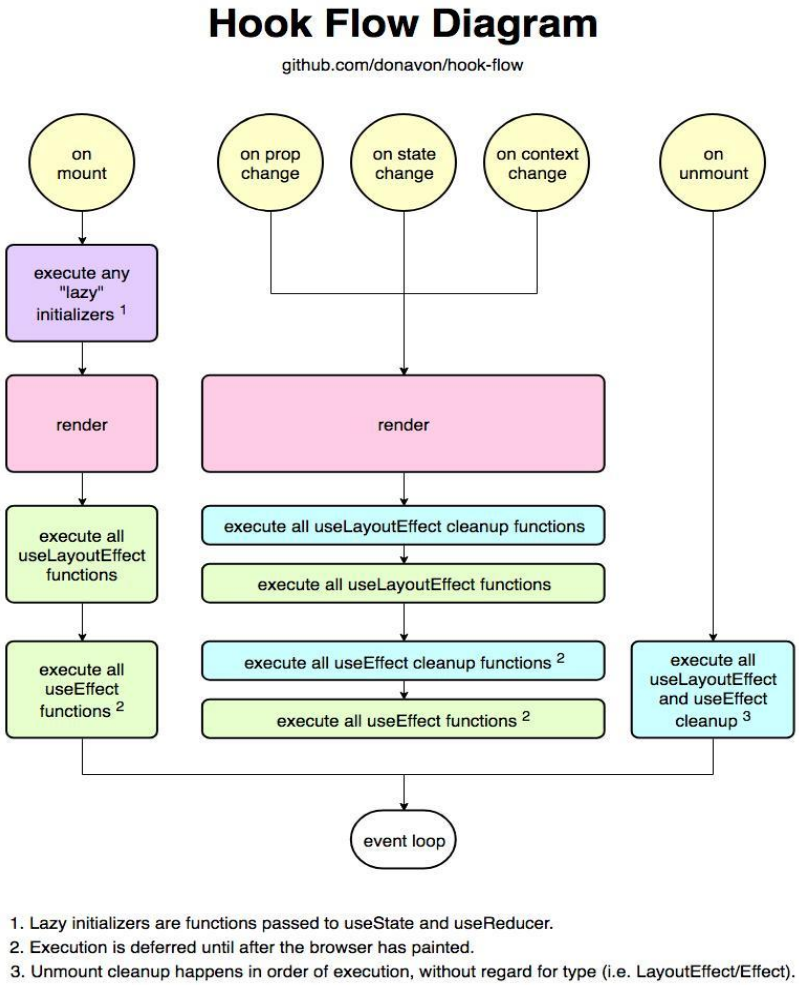
Kuva 4. Tilallinen luokkapohjainen, sekä funktionaalinen Login-lapsikomponentti

Kuvasta voidaan havaita myös eroja rakenteen ja syntaksin selkeydessä. Luokkakomponentti edellyttää erillisen render-metodin koodirivillä 19, jonka kautta JSX palautetaan, kun taas funktionaalinen komponentti voi palauttaa rakenteen suoraan return-lauseessa rivillä 8. Lisäksi luokkakomponentti vaatii metodien sitomista komponentin kontekstiin, kun taas funktionaalinen lähestymistapa mahdollistaa tilan ja logiikan käsittelyn ilman this-

avainsanaa. Näiden rakenteellisten erojen seurauksena funktionaalinen komponentti on tiiviimpi ja paremmin yhteensopiva Reactin nykyisen kehityssuunnan kanssa.

2.1.3 React-koukut

React-koukut (hooks) ovat olennainen osa modernia React-kehitystä mahdollistaen komponenttien tilan ja elinkaarien hallinnan funktionaalisesti. Ne tarjoavat monipuolisen joukon työkaluja ilman luokkapohjaisia komponentteja. (React 2025a.) Kuvion 2 prosessikaavio havainnollistaa koukkujen elinkaaren.



Kuvio 2. React-koukkujen elinkaari (Devopedia 2022a)

Diagrammi havainnollistaa kuinka React-koukkujen elinkaari etenee vaiheittain riippuen komponentin elinkaareen kuuluvista tapahtumista. Kun komponentti luodaan (mount-vaihe), suoritetaan mahdolliset lazy-initialisaattorit, minkä jälkeen tapahtuu renderöinti. Renderöinnin jälkeen aktivoituvat ensin useLayoutEffect-hookit ja niiden jälkeen useEffect-hookit.

Muutokset komponenteissa, kuten propsien, tilan tai kontekstin vaihtuminen, johtavat uuteen renderöintiin. Tämän jälkeen suoritetaan aina ensin useLayoutEffect cleanup -funktiot, sitten varsinainen useLayoutEffect sekä lopuksi useEffect cleanup ja useEffect-funktiot. useEffect ja sen cleanup-funktiot suoritetaan vasta, kun selain on päivittänyt ja piirtänyt käyttöliittymän näkyviin. Komponentin poistuessa (unmount) suoritetaan kaikki useLayoutEffect- ja useEffect-cleanup-funktiot siinä järjestyksessä, kuin ne on määritelty, riippumatta siitä kumpaan hook-tyyppiin ne liittyvät.

Sisäänrakennetut Koukut voidaan jakaa useisiin luokkiin käyttötarkoituksen mukaan. Tila-Koukut (useState, useReducer) hallitsevat komponenttien sisäistä tilaa. Konteksti-Koukut (useContext) mahdollistavat tiedon välittämisen komponenttipuussa ilman props-ketjutusta. Ref-Koukut (useRef, useImperativeHandle) tarjoavat DOM-manipulaatiota ja komponenttien välistä vuorovaikutusta. Efekti-Koukut (useEffect, useLayoutEffect, useInsertionEffect) hallitsevat sivuvaikutuksia ja synkronointia ulkoisten järjestelmien kanssa. Suorituskyky-koukut (useMemo, useCallback, useTransition, useDeferredValue) optimoivat renderöintiä ja parantavat responsiivisuutta. Muut koukut (useDebugValue, useId, useSyncExternalStore) palvelevat spesifisiä kehitystarpeita. Lisäksi kehittäjät voivat luoda omia mukautettuja Koukkuja. (React 2025a.)

2.1.4 Natiivit komponentit

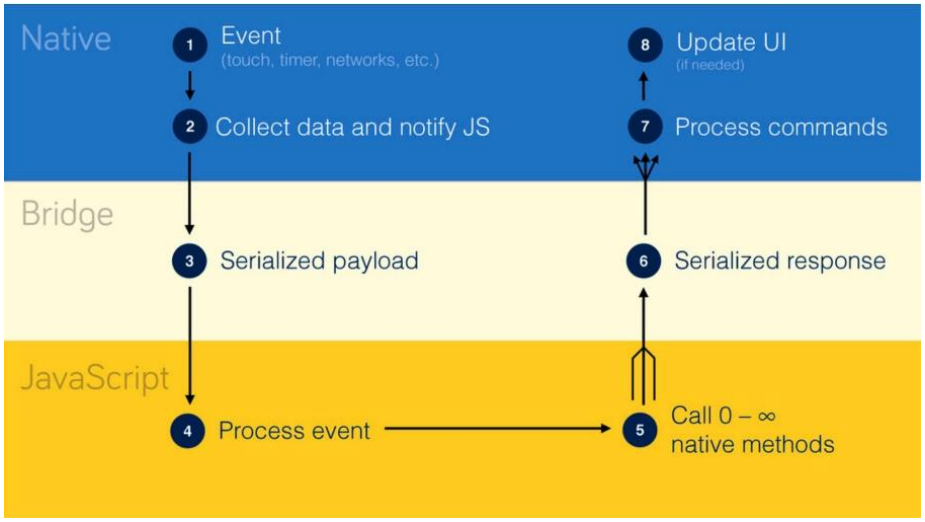
React Nativen natiivit komponentit ovat valmiita käyttöliittymäelementtejä, jotka on suunniteltu toimimaan suoraan mobiililaitteiden natiiveilla alustoilla. Nämä komponentit muodostavat React Nativen arkkitehtuurin perustan mahdollistaen natiivien mobiilisovellusten kehityksen. Taulukko 1 havainnollistaa React Native UI -komponenttien vastaavuuksia Android, IOS sekä WWW-näkymässä.

REACT NATIVE UI -KOMPONENTTI	ANDROID-NÄKYMÄ	IOS-NÄKYMÄ	VERKKOVASTINE	KUVAUS
`<View>`	`<ViewGroup>`	`<UIView>`	`<div>`	Säiliö, joka tukee joustavaa asettelua (flexbox), tyylejä, joitakin kosketustoimintoja ja saavutettavuuden hallintaa.
`<Text>`	`<TextView>`	`<UITextView>`	`<p>`	Näyttää, muotoilee ja sisällyttää tekstijonoja sekä tukee myös kosketustapahtumia.
`<Image>`	`<ImageView>`	`<UIImageView>`	`<img>`	Näyttää erilaisia kuvatyyppejä.
`<ScrollView>`	`<ScrollView>`	`<UIScrollView>`	`<div>`	Yleinen vieritettävä säiliö, joka voi sisältää useita komponentteja ja näkymiä.
`<TextInput>`	`<EditText>`	`<UITextField>`	`<input type="text">`	Mahdollistaa käyttäjän syöttää tekstiä.

Taulukko 1. React Nativen ydinkomponenttien vertailu (mukailtu React Native 2025b)

Nämä natiivit komponentit eroavat tavallisista React-komponenteista siten, että ne käyttävät suoraan alustakohtaisia Android tai iOS natiiveja vastineita sen sijaan, että ne koostuisivat HTML-, CSS- ja JavaScript-koodista. Tämä mahdollistaa sovellusten suorituskyvyn optimoinnin ja natiivien ominaisuuksien hyödyntämisen eri mobiilialustoilla.

Kuten kuviossa 3 esitetään, React Native yhdistää React-komponentit Androidin ja iOS:n natiivikomponentteihin C++:lla toteutetun sillan avulla. Silta siirtää serialisoituja JSON-objekteja asynkronisesti JavaScript- ja natiivipuolen välillä. (Devopedia 2022b.) Tämä arkkitehtuuri mahdollistaa natiivien mobiilisovellusten kehittämisen JavaScript-kielellä hyödyntäen samalla alustakohtaisia ominaisuuksia (Robinson-Adams 2022).



Kuvio 3. Kommunikaatiovirta natiivikoodin ja JavaScript-koodin välillä sillan kautta (Devopedia 2022b)

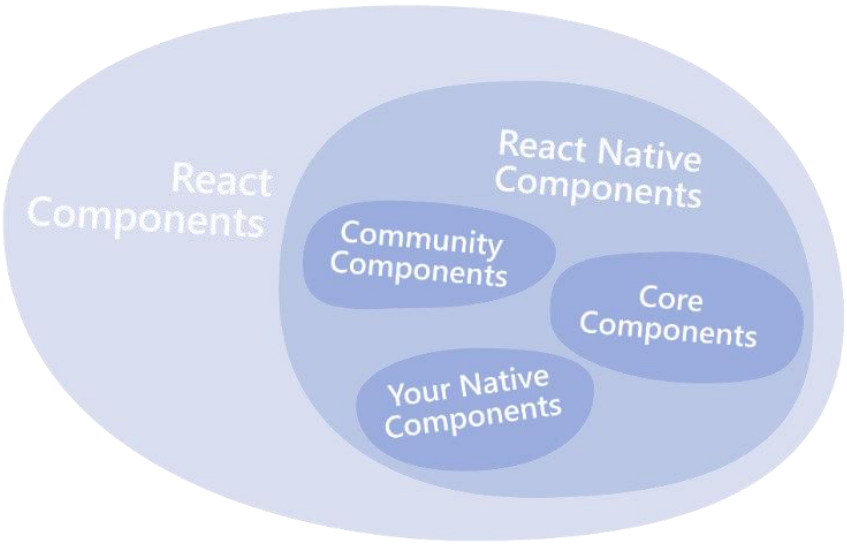
Sillan monivaiheinen rakenne on ollut keskeinen suorituskykyyn vaikuttava tekijä React Nativen arkkitehtuurissa. Kehitystyössä tähän on vastattu uudella arkkitehtuurilla, jossa JSI-rajapinnan ja Fabric-renderöintimoottorin avulla viestinvälitys tapahtuu suoremmin ilman raskasta serialisointia, mikä vähentää viiveitä ja parantaa käyttöliittymän reagointikykyä.

2.1.5 Integraatiot kolmannen osapuolen kirjastoihin

React Native -kehityksessä kolmannen osapuolen kirjastojen käyttö on yleistä ja tuo mukanaan sekä etuja että haasteita. Tyypillinen React Native -mobiilisovellus koostuu usein laajasta kokoelmasta kolmansien osapuolten kirjastoja, mikä mahdollistaa monipuolisten toiminnallisuuksien toteuttamisen tehokkaasti. Tämä open source -kirjastojen hyödyntäminen antaa kehittäjille mahdollisuuden valita juuri omiin tarpeisiin parhaiten sopivat komponentit ja toiminnallisuudet ilman, että niitä tarvitsee toteuttaa itse. (Punosmobile 2023.)

Kolmannen osapuolen kirjastojen käyttöön liittyy myös riskejä. Yksi merkittävimmistä haasteista on kirjastojen ylläpidon mahdollinen viivästyminen tai päivitysten puuttuminen. Tämä voi johtaa tilanteeseen, jossa sovellus ei pysty hyödyntämään uusimpia alustakohtaisia ominaisuuksia tai turvallisuuspäivityksiä yhtä nopeasti kuin natiivikehityksessä. Lisäksi, koska React Native on käyttöjärjestelmän ulkopuolinen kehys, uusien ominaisuuksien tuominen alustalle voi kestää kauemmin verrattuna natiiviohjelmointikieliin. (Punosmobile 2023.)

Yleisesti käytettyjä kirjastoja ovat esimerkiksi karttapalvelut, kompassi ja OAuth2-kirjautuminen. Karttapalveluiden integroimiseen sovelluksiin käytetään yleisesti react-native-maps-kirjastoa, joka mahdollistaa interaktiivisten karttojen lisäämisen käyttöliittymään (React Native Maps 2025). OAuth2-kirjautumisen toteuttamiseen React Native -sovelluksissa puolestaan soveltuu react-native-app-auth -kirjasto, joka helpottaa OAuth 2.0 ja OpenID Connect -protokollien implementointia (FormidableLabs 2025). Kompassitoiminnallisuuden toteuttamiseen voidaan käyttää react-native-compass-heading -kirjastoa, joka mahdollistaa kompassin suunnan vastaanottamisen sekä iOS- että Android-laitteilla (Firofame 2025). Kuvio 4 havainnollistaa kolmannen osapuolen komponenttien asemaa React Nativen komponenttiekosysteemissä.



Kuvio 4. React Native komponenttidiagrammi (React Native, 2025b)

React Nativen komponentit perustuvat Reactiin. Komponentit voidaan jaksaa kolmeen alajoukkoon: natiiveihin ydinkomponentteihin, kolmannen osapuolen kehittämiin jaettuihin community-komponentteihin sekä kehittäjien omiin natiivikomponentteihin.

2.1.6 Suorituskyky ja optimointi

Suorituskyvyn optimointi on keskeinen osa React Native -sovellusten kehitystä, sillä se vaikuttaa suoraan käyttäjäkokemukseen ja sovelluksen toimivuuteen. Suorituskykyongelmia voivat aiheuttaa erityisesti JavaScriptin ja natiivin koodin välinen vuorovaikutus, joka voi johtaa viivästyksiin ja suorituskyvyn heikkenemiseen erityisesti monimutkaisissa ja suurissa sovelluksissa. (Rafalski 2025.)

Yksi keskeinen optimointitekniikka on komponenttien tarpeettomien uudelleen renderöintien vähentäminen. React tarjoaa useita sisäänrakennettuja optimointimenetelmiä, kuten shouldComponentUpdate (luokkakomponenteille) ja React.memo (funktionaalisille komponenteille), jotka voivat parantaa sovelluksen nopeutta ja reaktiivisuutta (React 2025b). React Native -dokumentaatio tukee tätä lähestymistapaa ja suosittelee optimointitekniikoiden hyödyntämistä suorituskyvyn parantamiseksi.

Toinen tehokas tekniikka suorituskyvyn parantamiseksi on asynkroninen lataaminen ja komponenttien "laiska lataus" (lazy loading). Tämä menetelmä mahdollistaa sovelluksen osien lataamisen vain, kun niitä tarvitaan, mikä vähentää alkuperäistä latausaikaa ja parantaa käyttäjäkokemusta. Laiska lataus on erityisen hyödyllinen suurissa sovelluksissa, joissa kaikkia komponentteja ei tarvitse ladata heti sovelluksen käynnistyessä. (React 2025c.)

React Compiler on Reactin uusi käännösaikainen työkalu, joka automaattisesti optimoi sovelluksen komponentit ja koukut parantaen niiden suorituskykyä. Tämä vähentää turhia uudelleenrenderöintejä. React Compileria käytettäessä kehittäjän ei tarvitse enää käyttää useMemoa, useCallbackia tai React.memoa manuaalisesti. React Compilerin käyttöönotto on suositeltu, vaikka se on tällä hetkellä vapaaehtoinen lisä React-sovelluksiin, tulevaisuudessa jotkin ominaisuudet voivat edellyttää Compilerin käyttöä toimiakseen täysin. (React 2025d.)

Suorituskyvyn optimointiin kuuluu myös muistin ja muiden resurssien tehokas hallinta. Kuvien optimointi ja muistivuotojen estäminen voivat estää sovellusta kaatumasta tai hidastumasta pitkäaikaisessa käytössä. Kehityksessä on hyödyllistä käyttää React Native DevTools:ia, joka auttaa virheenkorjauksessa ja suorituskyvyn profiloinnissa. (React Native 2025c.)

Jos suorituskyvyn ongelmat eivät ratkea pelkästään JavaScriptin optimoinnilla, voidaan harkita natiivien moduulien käyttöä. Natiivikoodin kirjoittaminen (Swift/Objective-C tai Kotlin/Java) ja käyttö voi olla erityisen hyödyllistä sovelluksissa, joissa käsitellään raskaita laskutoimituksia tai monimutkaista grafiikkaa. Yksittäisten natiivikomponenttien kirjoittaminen

antaa kehittäjille mahdollisuuden optimoida kriittisiä osia sovelluksesta ilman, että koko sovellus pitäisi kehittää natiivisti. (React Native 2025d.)

2.2 Tilanhallinta React Nativessa

Tilanhallinta React Nativessa on keskeinen osa sovelluskehitystä, joka vaikuttaa merkittävästi mobiilisovellusten toimintaan, suorituskykyyn ja skaalautuvuuteen. Tilanhallinta voidaan jakaa kahteen pääkategoriaan: komponenttien paikalliseen tilaan ja globaaliin tilaan.

Komponenttien paikallinen tila soveltuu hyvin käyttöliittymälogiikan ja renderöinnin hallintaan tietyssä sovelluksen osassa. Globaalin tilan hallintaan puolestaan käytetään usein Redux-kirjastoa, joka tarjoaa keskitetyn tietovaraston ja parantaa tiedonkulkua erityisesti laajoissa sovelluksissa. (Hafkenschiel 2023.)

Tilanhallinnan hallitseminen on olennainen taito React Native -kehittäjille. Noudattamalla parhaita käytäntöjä ja valitsemalla sopivat työkalut voidaan luoda suorituskykyisiä, luotettavia ja käyttäjäystävällisiä sovelluksia. (Hafkenschiel 2023.)

Tilanhallintatyökalun valinnassa on huomioitava sovelluksen vaatimukset:

- Redux sopii laajoihin, monimutkaista tilanhallintaa vaativiin sovelluksiin.
- Context API on kevyempi vaihtoehto yksinkertaisempiin tarpeisiin.
- MobX tarjoaa reaktiivisen tilanhallinnan vaihtoehtona Reduxille.
- Komponenttien paikallinen tilanhallinta Reactin omalla setState-metodilla tai useState-koukulla.

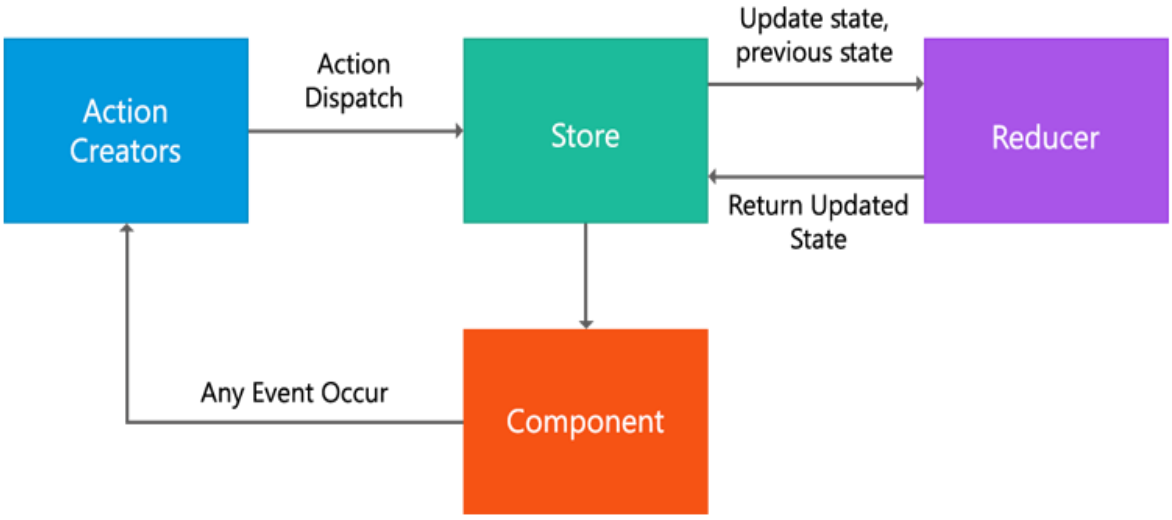
2.3 Redux-tilanhallinta

Redux on JavaScript-kirjasto, joka tarjoaa ennustettavan ja keskitetyn tavan hallita sovellusten tilaa. Reduxia on yleisesti käytetty osana Reactia, mutta se toimii myös muissa JavaScript-kehiyksissä kuten Vue ja Angular. (Flatirons 2024.)

Yksi keskeisistä Reduxin toiminnallisuuksista on globaalin tilan hallinta. Redux ylläpitää sovelluksen tilaa keskusvarastossa (store), joka toimii yhtenäisenä tiedonlähteenä kaikille komponenteille. Tämä eliminoi tarpeen välittää tilaa komponenttien välillä propseina (prop drilling) ja helpottaa tilan seurantaa monimutkaisissa sovelluksissa. Globaali tila on erityisen hyödyllinen, kun useat komponentit tarvitsevat samaa dataa kuten esimerkiksi käyttäjän kirjautumistilaa. (Flatirons 2024.)

2.3.1 Redux toimintaperiaatteet ja arkkitehtuuri

Reduxin kolmen keskeistä toimintaperiaatetta voidaan kuvata seuraavasti: Yksi totuuden lähde (Single source of truth). Koko sovelluksen tila säilytetään yhdessä puurakenteisessa objektissa. Tila on vain lukumuodossa (immutable data). Tilan muutokset tehdään aina toimintojen (actions) kautta, ei suoralla manipuloinnilla. Muutokset toteutetaan puhtailla funktioilla (pure functions), jolloin reducerit käsittelevät toiminnot synkronisesti ilman sivuvaikutuksia. (Flatirons 2024.) Kuvio 5 havainnollistaa prosessin keskeiset vaiheet.



Kuvio 5. Reduxin toimintakaavio (ScolarHat 2025)

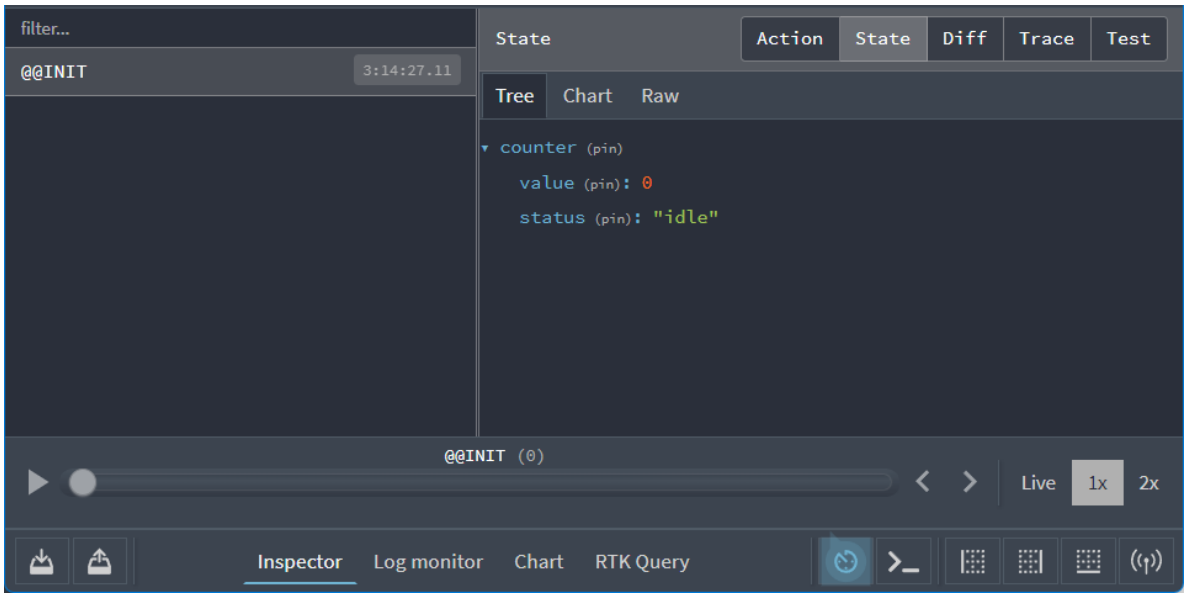
Reduxin arkkitehtuuri perustuu yksisuuntaiseen tietovirtaukseen, jossa data kulkee yksisuuntaisesti ja ennustettavasti läpi sovelluksen, alkaen käyttöliittymästä lähetetystä toiminnosta (action), joka sitten prosessoidaan reducerissa. Reducer, toimien puhtaana funktiona, tuottaa uuden tilan annetun toiminnon ja edellisen tilan perusteella. (Flatirons 2024.)

Tämän jälkeen store, joka toimii sovelluksen keskitettynä tilahallinnan yksikkönä, päivittää globaalin tilan. Lopuksi käyttöliittymä renderöidään uudelleen heijastamaan päivitettyä tilaa, mikä takaa sovelluksen johdonmukaisen ja ennustettavan käyttäytymisen. Tämä yksisuuntainen tietovirtamalli tekee myös virheenjäljityksestä hallittavaa ja selkeää.

2.3.2 Reduxin debuggaus

Redux DevTools on suosittu työkalu Redux-pohjaisten sovellusten kehittämiseen ja virheiden jäljittämiseen. Se tarjoaa kehittäjille mahdollisuuden tarkastella ja manipuloida sovelluksen tilaa reaaliajassa, mikä helpottaa monimutkaisten tilanhallinnan ongelmien ratkaisemista. Tämä työkalu sisältää useita ominaisuuksia, jotka tekevät siitä hyödyllisen apuvälineen Redux-sovellusten kehittäjille. (Obregon 2023.)

Redux DevTools sisältää myös hyödyllisen komponenttipuun tarkastelijan. Tämä ominaisuus auttaa visualisoimaan komponenttien välisiä suhteita ja mahdollistaa yksittäisten komponenttien propsien ja tilan tarkastelun. Tämä on erityisen hyödyllistä monimutkaisten komponenttihierarkioiden ymmärtämisessä ja debuggauksessa. Nämä auttavat kehittäjiä tunnistamaan ja analysoimaan sovelluksessa esiintyviä virheitä nopeasti ja tehokkaasti, mikä nopeuttaa ongelmien ratkaisua ja parantaa sovelluksen vakautta. (Obregon 2023.) Kuva 5 havainnollistaa Redux DevTools:in käyttöliittymää.



Kuva 5. Redux DevTools käyttöliittymä (Redux 2025)

Yksi Redux DevToolsin keskeisistä ominaisuuksista on aikamatkustus-debuggaus. Tämä toiminto mahdollistaa kehittäjien liikkumisen sovelluksen tilahistoriassa eteen- ja taaksepäin, mikä auttaa ongelmien tarkassa paikantamisessa ja sovelluksen käyttäytymisen ymmärtämisessä eri ajanhetkillä. Lisäksi työkalu tarjoaa yksityiskohtaisen näkymän sovelluksen toiminnoista ja tilasta, mukaan lukien nykyisen tilan ja aiemmat tilat. (Obregon 2023.)

2.4 Tekoäly

Tekoäly on nykyaikaisen mobiilisovelluskehityksen keskeinen osa-alue, joka mahdollistaa luonnonpaikkojen automaattisen tunnistuksen ja analysoinnin. Erityisesti luonnossa liikkuvien käyttäjien tarpeisiin tekoäly yhdistää koneoppimisen ja konenäön menetelmät, jotka pystyvät analysoimaan mobiililaitteen kameralla otettuja kuvia, tunnistamaan niissä esiintyviä lajeja ja tarjoamaan käyttäjälle älykästä palautetta. Tämän automaation avulla käyttäjän ei tarvitse erikseen syöttää tietoja, vaan tunnistus tapahtuu reaaliajassa ja tarjoaa välittömän vastauksen. (Wäldchen ym. 2018.)

Keskeinen tekninen lähtökohta on syväkonvoluutioneuroverkkojen (CNN) käyttö, jotka ovat osoittaneet olevansa erityisen tehokkaita kuvantunnistustehtävissä. Krizhevskyn, Sutskeverin ja Hintonin 2012 lanseeraama CNN-arkkitehtuuri on edelleen yksi peruskivistä, joka on mahdollistanut modernin kuvantunnistuksen sovellukset kaikilla aloilla. (Krizhevsky ym. 2012.) Syväoppimisen kirjastot kuten fastai ja PyTorch puolestaan helpottavat näiden mallien tunnetuksi tekemistä ja räätälöintiä käytännön sovelluksiin (Howard & Gugger 2017).

2.4.1 Reaaliaikainen sienten tunnistus

Wäldchen ym. (2018) mukaan sienten reaaliaikainen tunnistus on tekoälypohjaisen luontosovelluksen ydinominaisuuksia. Käyttämällä laajasti kerättyjä kuvia sekä neuroverkkoja, sovellus pystyy analysoimaan luonnossa otetut kuvat ja tunnistamaan lajin paikan päällä. Tämä teknologia perustuu CNN-malleihin, joiden tehokas arkkitehtuuri mahdollistaa monimutkaisten kuvioiden ja piirteiden tunnistamisen monenlaisissa olosuhteissa.

Paikallinen arviointi mobiililaitteissa mahdollistaa sen, että käyttäjä saa välittömän palautteen ilman tarvetta lähettää suuria kuvadatajoukkoja pilveen, mikä ylläpitää sovelluksen käytettävyyttä ja tietoturvaa. LiteRT (entinen Tensorflow) mahdollistaa tätä käyttämällä optimoituja malleja, jotka hyödyntävät mobiililaitteiden rajallisia laskentaresursseja tehokkaasti ja energiaa säästävästi. (LiteRT 2025.) Tämä on erityisesti tärkeää ulkoilmassa, missä verkkoyhteys voi olla puutteellinen tai poissa.

Vaikka mallin tarkkuus riippuu paljon koulutusdatasta, Krizhevskyn ym. (2012) artikkeli osoittaa, että CNN-arkkitehtuurit ovat osoittautuneet erittäin kykeneviksi kuvantunnistustehtävissä, ja ne ovat nykyisin standardi myös luonnontunnistuksessa. Fastai- ja PyTorch-kirjastot taas tarjoavat helppokäyttöiset työkalut mallien kehittämiseen tukien jatkuvaa parantamista ja soveltamista mobiilisovelluksiin (Howard & Gugger 2017, luku 4).

2.4.2 Kohteiden tunnistus maastosta

Luonnon kohteiden tunnistus maastossa ei rajoitu pelkästään sieniin, vaan laajennusmahdollisuus koskee myös kasveja, marjoja ja muita luonnon ilmiöitä. Moderneissa sovelluksissa yhdistetään kuvan analyysi ja kartta-aineisto, jolloin käyttäjät voivat tallentaa ja lokalisoida havaintojaan karttanäkymään reaaliaikaisesti. Tämä tuo lisäarvoa, sillä käyttöliittymä ei vain näytä kohdetta vaan liittää sen paikkaan, jolloin tiedonkeruu ja analyysi tehostuvat.

Pilvipalvelut, kuten Google Cloud Vision API, mahdollistavat laajoja luokitus- ja tunnistusominaisuuksia sekä jatkuvan mallien päivityksen ilman, että sovelluksen käyttäjän tarvitsee huolehtia paikallisista päivityksistä. Tämä täydentää mobiililaitteiden paikallista laskentatehoa ja mahdollistaa käytännölliset hybridiratkaisut. (Google Cloud Vision API 2025.)

2.4.3 Tekoälyn mahdolliset riskit

Vaikka tekoäly tarjoaa merkittäviä etuja luonnon kohteiden tunnistuksessa, siihen liittyy myös riskejä. Esimerkiksi virheellinen tunnistus voi johtaa käyttäjiä harhaan, mikä on erityisen vakavaa myrkyllisten sienten kohdalla, joissa väärä tunnistus voi olla hengenvaarallinen. Käyttäjille tulee siksi tarjota kriittinen näkökulma tekoälyn antamiin tunnistuksiin ja mahdollisuus ihmisen tekemään tarkistukseen.

Toinen keskeinen seikka on yksityisyyden suojaaminen. Kuvien ja sijaintitietojen käsittelyssä on noudatettava GDPR:n vaatimuksia, jotka määrittävät muun muassa käyttäjän oikeuden hallita omia tietojaan ja velvollisuuden käsitellä tietoja turvallisesti (GDPR 2018). Tekoälyn opetusdatojen ja mallien tulee olla mahdollisimman puolueettomia ja oikein hallinnoituja, jotteivat ne johda syrjiviin tuloksiin tai skenaarioriskeihin (Howard & Gugger 2017).

Tekoälyratkaisujen jatkuva valvonta, päivitys ja ylläpito ovat tarpeen, jotta sovellukset pysyvät luotettavina muuttuvissa ympäristöissä ja liiketoimintatarpeissa. Tämä edellyttää suunnitelmallisuutta ja resursseja, mutta parantaa pitkällä aikavälillä sekä käyttäjäkokemusta että turvallisuutta.

2.4.4 Tekoälyn etiikka ja vastuullisuus

Tekoälyn etiikka luonnontieteellisissä sovelluksissa korostaa läpinäkyvyyden ja käyttäjien autonomian merkitystä. Luonnossa liikkuvien käyttäjien tulee ymmärtää, miten tekoälyjärjestelmä toimii, esimerkiksi miten varmoja sen tunnistukset ovat ja mitkä ovat sen rajoitukset. Sovelluksen tulisi tarjota mahdollisuus ihmisen tekemään tarkistukseen epävarmoissa tilanteissa, jotta käyttäjä voi säilyttää aktiivisen roolin päätöksenteossaan. Tämä vastaa tutkimusyhteisössä laajasti esitettyä vaatimusääntä avoimuudesta ja käyttäjälähtöisyydestä tekoälyn käytössä. (Helsingin yliopisto 2024.)

Sirviön (2022) mukaan vastuullinen tekoälyn käyttö ei ole vain tekninen haaste, vaan se liittyy myös yhteiskunnallisiin rooleihin ja vastuun jakautumiseen. Sovelluskehittäjien, tutkijoiden ja käyttäjien tulee yhdessä kantaa vastuu siitä, että tekoälysovellukset toimivat luotettavasti, oikeudenmukaisesti ja turvallisesti. Tämä eettinen ulottuvuus korostaa myös monialaista yhteistyötä sekä säännöllistä arviointia ja päivittämistä uuden tiedon ja muuttuvien olosuhteiden mukaan.

Tekoälyn vastuullinen käyttö voi toimia luonnonsuojelun tukemisessa ja ympäristötietoisuuden lisäämisessä. Vastuullisesti koulutetut mallit, jotka huomioivat biodiversiteetin monimuotoisuuden ilman vääristymiä, voivat edistää ekologista tasa-arvoa ja auttaa

paikallistason päätöksenteossa. Tällainen lähestymistapa varmistaa, että tekoälysovellukset tukevat kestävää kehitystä ja ihmiskeskeistä luonnon arvostusta ilman teknologista ylivaltaa. (LUT 2024.)

Lisäksi luotettavan tekoälyn rakentaminen vaatii käyttäjien ja ammattilaisten keskinäistä luottamusta. Käyttäjien kouluttaminen tekoälyn mahdollisuuksiin ja rajoituksiin sekä käytännön eettisten periaatteiden esilletuonti vaikuttaa sovellusten omaksumiseen ja vastuulliseen hyödyntämiseen (Euroopan komissio 2019). Näin varmistetaan, että tekoäly toimii apuvälineenä, ei korvaajana, ja että se auttaa luonnossa tekemään tietoisia ja vastuullisia päätöksiä.

3 Hybridi mobiilisovelluksen toteutus

3.1 Arkkitehtuuri

Mobiilisovelluksen toteutuksessa on valittu moderni teknologiapino, joka tukee tehokasta ja joustavaa kehitystä sekä skaalautuvuutta. Sovellus on rakennettu käyttäen TypeScriptiä, joka tarjoaa vahvan tyypityksen ja parantaa koodin ylläpidettävyyttä sekä virheiden hallintaa. Käyttöliittymän rakentamisessa hyödynnettiin React Nativea, joka mahdollistaa sovelluksen julkaisun sekä iOS- että Android-alustoille yhdellä yhteisellä koodipohjalla. Tämä nopeuttaa kehitysprosessia ja vähentää ylläpitokustannuksia.

Tilanhallintaan sovelluksessa on otettu käyttöön Redux, joka tarjoaa ennustettavan tilanhallinnan ja helpottaa sovelluksen tilan hallintaa erityisesti monimutkaisemmissa käyttöliittymissä. Paikalliseen tiedontallennukseen käytetään SQLite-tietokantaa, joka tarjoaa relaatiomalliin perustuvan ja suorituskykyisen ratkaisun datahallintaan myös offline-tilanteissa. Lisäksi AsyncStorage toimii välimuistina pienimuotoisemmalle datalle, kuten käyttäjäasetuksille, mahdollistaen nopean tallennuksen ja tiedon käsittelyn.

Sovelluksen keskiössä on karttapalvelu, joka toteutetaan hyödyntämällä Maanmittauslaitoksen tarjoamaa ohjelmistorajapintaa. Tämä API mahdollistaa ajantasaisen ja paikkatietopohjaisen karttatiedon hakemisen ja esittämisen sovelluksessa. Yhdistämällä paikallinen tietokanta ja ulkoinen API-data saadaan aikaan lähes reaaliaikainen ja käyttäjäystävällinen karttapalvelu.

Sovelluksen kansiohierarkia on selkeästi jäsennelty, mikä tukee koodin ylläpidettävyyttä ja modulaarisuutta. Src-juurihakemistossa sijaitsevat sovelluksen kaikki keskeiset lähdekooditiedostot ja -kansiot.

- Assets-kansio sisältää sovelluksen staattiset resurssit, kuten kuvat, joita käyttöliittymä käyttää.
- Components-kansiosta löytyy uudelleenkäytettäviä React-komponentteja, kuten kielenvalintakomponentti (LanguageSelector.tsx).
- Constants sisältää sovelluksen käyttöön määriteltyjä vakioarvoja, joita voidaan hyödyntää läpi sovelluksen.
- Database-kansio on varattu paikallisen tietokannan hallintaan. Siellä kansio "queries" sisältää tietokantakyselytiedostot, jotka liittyvät sijaintitietoihin sekä muistiinpanoihin. Lisäksi kansiossa on skeema- ja indeksointitiedostot, jotka määrittelevät tietokannan rakenteen ja käyttölogiikan.

- Features-kansio jakaa sovelluksen eri toiminnallisuusalueisiin, kameraan, karttaan, muistiinpanoihin ja asetuksiin. Tämä rakenne mahdollistaa kunkin ominaisuuden kehittämisen ja ylläpidon erillään muista.
- Hooks sisältää Reactin omat koukut, joita käytetään sovelluksen tilanhallinnassa ja logiikassa.
- Screens-kansio pitää sisällään sovelluksen näkymäkomponentit eli eri näkymät, joita käyttäjä sovelluksessa näkee.
- Services-kansio sisältää tiedoston, joka vastaa Maanmittauslaitoksen rajapinnan käsittelystä.
- Store sisältää Reduxin tilanhallinnan määrittelyt ja logiikan.
- Translations-kansio sisältää käännöstiedostot.

Näin rakentuva kansiorakenne tukee modulaarista kehitystä, parantaa selkeyttä ja helpottaa myös tiimityöskentelyä isommissa projekteissa. Se erottelee sovelluksen eri toiminnallisuudet ja tukee teknologiapinon (TypeScript, React Native, Redux, SQLite, AsyncStorage, API-yhteydet) tehokasta hyödyntämistä. Tämä rakenne tukee myös helppoa laajentamista ja ylläpitoa.

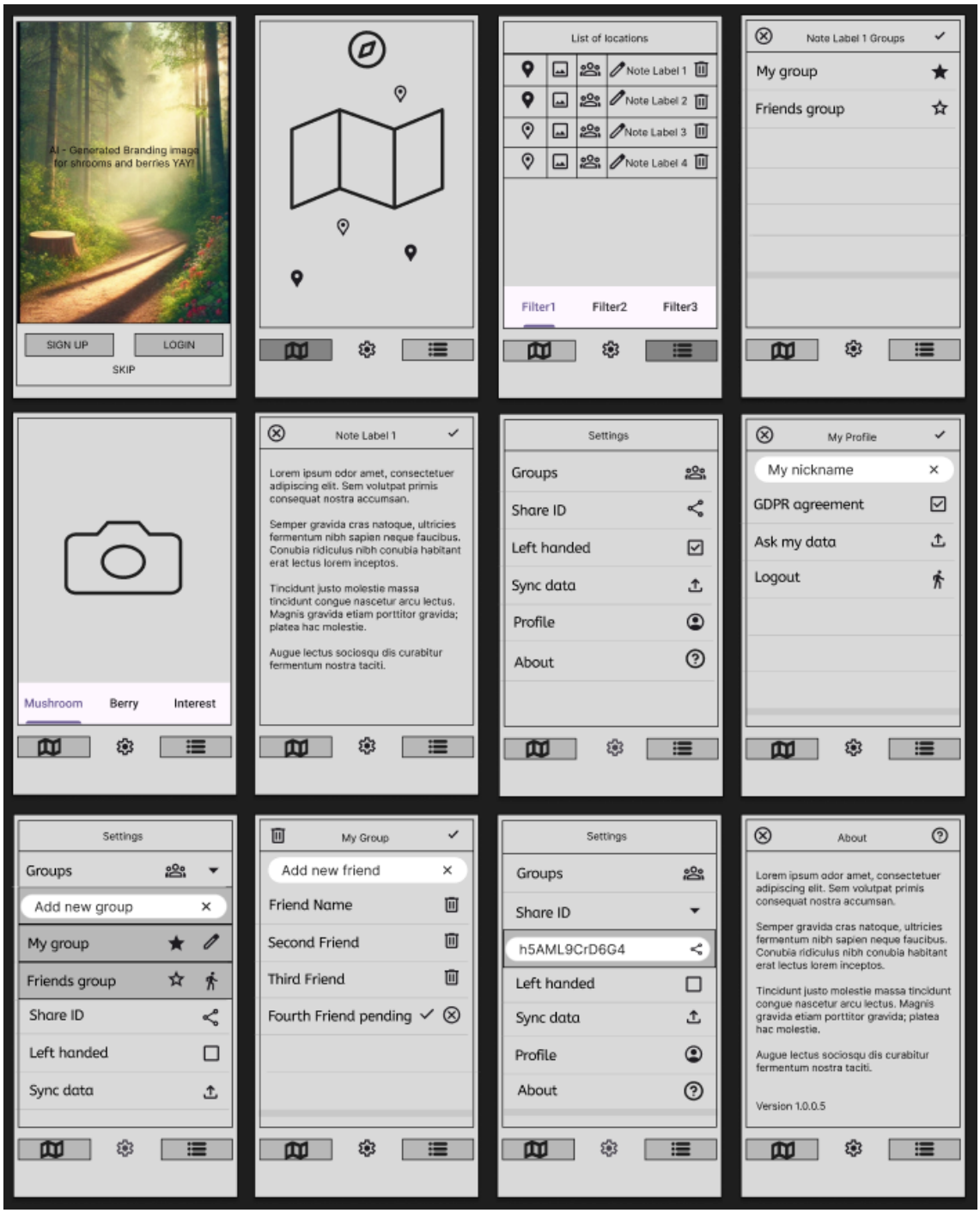
3.2 Käyttöliittymän suunnittelu

Käyttöliittymän suunnittelussa LuontoSpot-sovelluksessa on panostettu selkeyteen ja käytettävyyteen, jotta käyttäjäkokemus olisi mahdollisimman sujuva ja intuitiivinen. Sovelluksen visuaalisen ilmeen ja toimintojen suunnittelussa on kiinnitetty huomiota siihen, että käyttöliittymä toimii johdonmukaisesti eri mobiililaitteilla ja erilaisissa käyttötilanteissa. Mahdollisimman responsiiviset komponentit mukautuvat näytön kokoon ja käyttöjärjestelmään, mikä mahdollistaa helpon navigoinnin ja havainnollisen karttanäkymän, sekä listaukset.

Suunnittelun tavoitteena on tarjota käyttäjälle miellyttävä ja tehokas käyttökokemus, jossa luonnonhavaintojen tallentaminen on nopeaa ja visuaalisesti selkeää. Käyttöliittymän toiminnallisuudet on pyritty integroimaan saumattomasti osaksi sovelluksen kokonaisuutta, jolloin käyttäjä voi keskittyä omien havaintojen tallentamiseen.

3.2.1 Ulkoasu

Sovelluksen käyttöliittymää suunniteltiin aluksi Figma:lla. Sovelluksen käyttöliittymän suunnittelussa on panostettu selkeyteen ja käyttäjäystävällisyyteen, mikä näkyy visuaalisissa luonnoksissa. Ulkoasu rakentuu minimalistisesta ja toiminnallisesta tyylistä, jossa navigointi on helppoa ja intuitiivista. Suunnittelussa otettiin käyttöön mahdollisimman paljon ikoneita, jolloin säästytään suurelta käännösten määrältä (Kuva 6).



Kuva 6. Figman rautalankamallit

Etusivunäkymä tarjoaa käyttäjälle selkeät kirjautumis- ja rekisteröitymisvaihtoehdot tulevaisuuden tarpeisiin, ja sovelluksen visuaalinen ilme on luonnonläheinen, joka tukee sovelluksen teemaa.

Karttanäkymässä käyttäjä voi tarkastella eri sijainteja selkeällä ja yksinkertaisella karttatoteutuksella, jossa sijainnit on merkitty helposti tunnistettavilla kuvakkeilla. Näkymän alaosassa on navigaatiopainikkeet, joilla siirrytään nopeasti muihin keskeisiin osioihin, kuten asetuksiin ja listanäkymään.

Muistiinpanolista esittelee käyttäjän tekemät merkinnät selkeinä riveinä, joissa näkyvät sekä otsikot että niihin liittyvät visuaaliset merkit, kuten kuvat ja sijainnit. Käyttäjä voi suodattaa ja hallita muistiinpanojaan kätevästi. Kameranäkymä on suunniteltu todella suoraviivaiseksi keskittyen helppoon kuvan ottamiseen ja merkintäkategoriaksi valitsemiseen, mikä nopeuttaa havaintojen kirjaamista. Muistionäkymä tarjoaa laajemman mahdollisuuden muistiinpanojen lukemiseen ja muokkaamiseen, korostaen tekstin selkeyttä ja käytettävyyttä.

Asetusnäkymä demonstroi monipuolisia ja tulevaisuudessa laajennettavissa olevia toimintoja, kuten käyttäjäryhmien hallintaa, tiedon synkronointia, profiilin muokkausta sekä tietosuoja-asetusten hallintaa. Lisäksi näkymä sisältää mahdollisuuksia jakaa käyttäjätunnus ja mukauttaa sovelluksen käyttöä henkilökohtaisten mieltymysten mukaan. Ulkoasun suunnittelussa haluttiin korostaa helppokäyttöisyyttä ja laajennettavuutta, mikä toimii pohjana tulevalle kehitykselle ja käyttäjäkokemuksen parantamiselle.

3.2.2 Sovelluksen saavutettavuus

Sovelluksen käyttöliittymän suunnittelun alkuvaiheessa harkittiin muun muassa vasenkätisten tilan tarvetta. Testauksen ja käytettävyyden arvioinnin perusteella päädyttiin keskitettyihin painikkeisiin, jotka ovat mahdollisimman helppokäyttöiset yhdellä asettelulla.

Sovelluksessa käytetään vihreää päävärinä sekä muita luontoon viittaavia sävyjä, mikä tukee sovelluksen teemaa ja antaa käyttäjälle rauhallisen sekä selkeän käyttöliittymäkokemuksen. Tummassa tilassa värit vaihtuvat sopivammaksi kontrastin ansiosta, mikä parantaa luettavuutta ja käyttöergonomiaa hämärässä. Visuaalinen ilme on suunniteltu minimalistiseksi ja selkeäksi siten, että ikonit, painikkeet ja navigointielementit ovat helposti tunnistettavissa ja saavutettavissa.

Käytettävyyttä on parannettu listauksilla, joissa otsikot ja visuaaliset merkinnät ovat helposti hahmotettavissa, ja navigointipainikkeet sijoitettu nopean siirtymisen mahdollistamiseksi eri näkymiin. Muistiinpanolistauksessa merkinnät ovat selkeinä riveinä, ja muistiinpanojen lukemista sekä muokkaamista varten painopiste on laitettu tekstin selkeyteen ja käyttöliittymän yksinkertaisuuteen.

3.3 Sovelluksen kehitys

Sovelluksen kehitys aloitettiin alustamalla React Native -projekti käyttäen NPX-työkalua. Projekti nimettiin LuontoSpotiksi. Kehityksen alkuvaiheessa keskityttiin keskeisten toiminnallisuuksien ja teknologioiden integrointiin sekä niiden toimivuuden varmistamiseen.

Ensimmäisenä vaiheena sovellukseen asennettiin Redux-kirjasto tilanhallinnan toteuttamiseksi. Redux valittiin sen tarjoaman keskitetyn ja ennustettavan tilanhallintamallin vuoksi. Reduxin toiminnallisuuden testaamiseksi toteutettiin yksinkertainen laskurikomponentti, jolla varmistettiin sekä sovelluksen kääntyminen että Reduxin asianmukainen toiminta.

Seuraavana vaiheena sovellukseen integroitiin sqlite-storage-paketti paikallisen tietokannan toteuttamiseksi. Tietokannan toimivuuden testaamiseksi laajennettiin aiemmin luotua laskuria. Sovellukseen lisättiin toiminnallisuus, joka mahdollisti laskurin arvon tallentamisen SQLite-tietokantaan. Tämä laajennus palveli kahta tarkoitusta:

- Se demonstroi Reduxin ja SQLiten yhteistoimintaa sovelluksessa.
- Se mahdollisti SQL-implementaation toimivuuden testaamisen käytännössä.

Tilanhallinnan implementoinnin ja paikallisen tietokantaintegraation valmistuttua sovelluksen kehityksessä siirryttiin käyttöliittymän parantamiseen ja yhtenäistämiseen. Tässä vaiheessa otettiin käyttöön React Native Paper -komponenttikirjasto, joka tarjoaa valmiita, Material Design -ohjeistuksia noudattavia käyttöliittymäelementtejä. Lisäksi implementoitiin SafeAreaContext, joka varmistaa sovelluksen sisällön asianmukaisen sijoittumisen eri mobiililaitteiden näytöillä. Käyttöliittymän visuaalista ilmettä ja käytettävyyttä parannettiin edelleen lisäämällä projektiin Vector Icons-ikonipaketti.

Sovelluksen toiminnallisuuden laajentamiseksi ja käyttökokemuksen parantamiseksi integroitiin i18next-lokalisaatiopaketti, joka mahdollistaa sovelluksen kääntämisen useille kielille. Lisäksi implementoitiin tuki mobiililaitteiden ”gestureille”, mikä parantaa sovelluksen interaktiivisuutta ja käytettävyyttä. Kehitysprosessin aikana asennettiin myös muita tukikirjastoja, jotka helpottavat sovelluksen käyttöä ja tehostavat sen toiminnallisuutta.

Integraatioita ja kirjastoja testattiin aluksi kehitetyllä laskurikomponentilla, jotta voitiin varmistaa kaikkien ominaisuuksien toimivuus ja yhteensopivuus ennen sovelluksen yksilöllisiä komponentteja. Nämä kehitysvaiheet mahdollistivat siirtymisen varsinaisten sovelluskomponenttien suunnitteluun ja toteutukseen.

Sovelluksessa hyödynnettiin laajasti kolmansien osapuolien kirjastoja ja teknologioita, jotka lähtökohtaisesti nopeuttivat kehitystä ja käyttöliittymän monipuolisuutta. Keskeisiä kirjastoja olivat mm.:

- Async-storage: Käyttäjätietojen ja asetusten pysyvään säilömiseen.
- React-navigation: Monipuoliseen navigointiin (stack, drawer, native).
- Redux/toolkit: Tilanhallinnan ja skaalautuvuuden toteutukseen.
- Sqlite-storage: Paikallisen tietokannan toteutukseen.

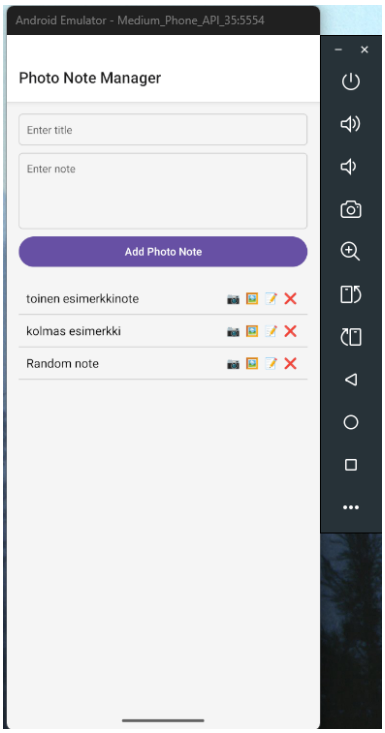
- i18next ja i18next-browser-languagedetector: Monikielisyyden ja automaattisen kielivalinnan toteutukseen.
- React-native-paper/reanimated/screens/maps/vector-icons/vision-camera: Integroimaan karttaominaisuudet, animaatiot, ikonit ja kameran käyttö osaksi sovellusta.

Kirjastojen valintaa perusteltiin niiden yhteensopivuudella React Native -ekosysteemin kanssa, aktiivisella kehityksellä sekä selkeillä dokumentaatioilla. Jokainen kirjasto asennettiin npm:llä, konfiguroitiin projektin tarpeen mukaan ja testattiin käyttöliittymän sekä toiminnallisuuksien kehityksen yhteydessä.

Node-pohjaiset riippuvuudet hallittiin package.json-tiedostossa. Kirjastojen versiot valittiin niin, että ne tukivat React Nativen ja siihen liittyvien tekniikoiden uusinta versiota ja mahdollistivat skaalautuvan sekä laajennettavan sovellusarkkitehtuurin. Pakettien päivittäminen oli osa jatkuvaa kehitystyötä, jolla varmistettiin sovelluksen toimivuus ja tietoturva.

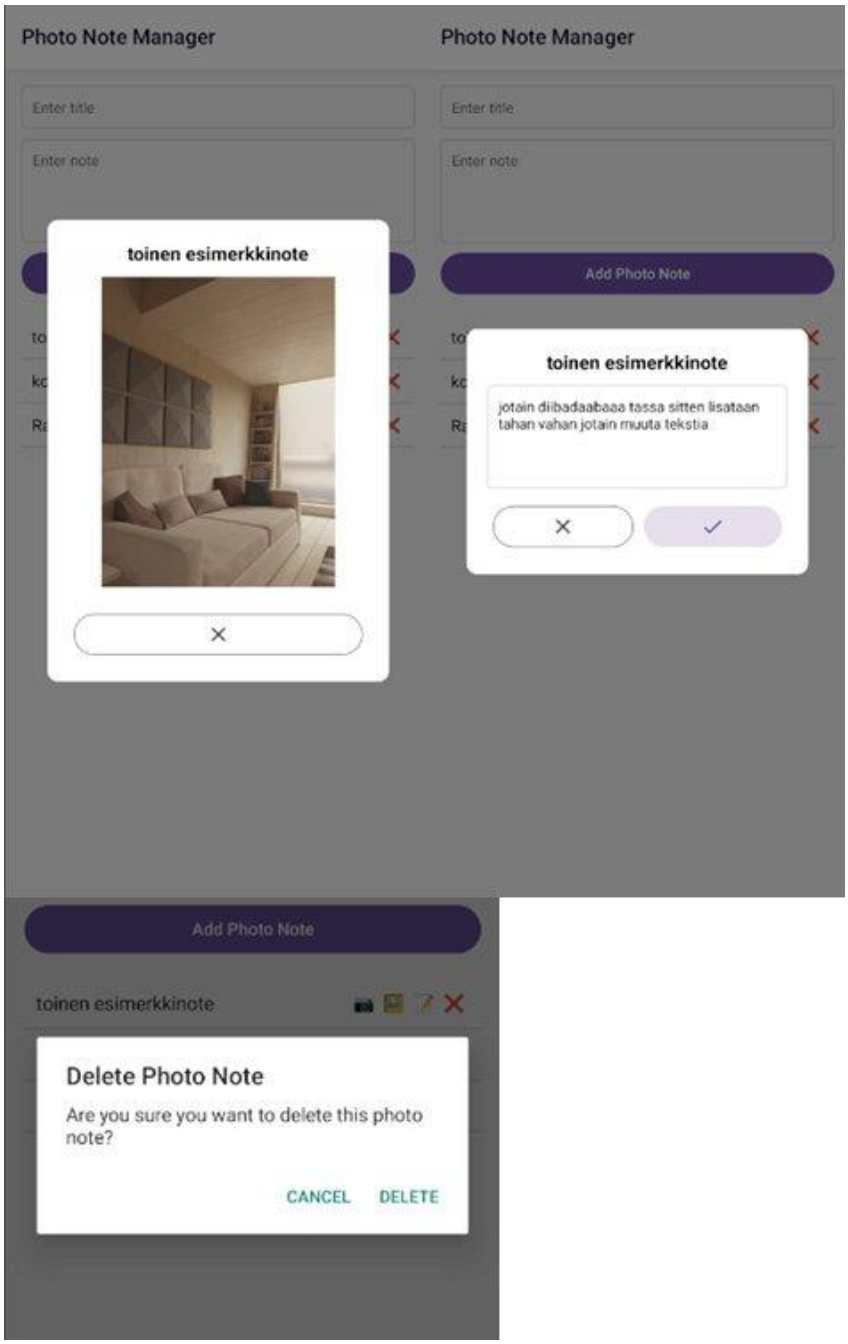
3.3.1 Listausnäkymä

Listausnäkymästä tuli sovelluksen ensimmäinen oma komponentti, joka toimii myös omana sovelluksena PhotoNoteManagerina. PhotoNoteManageriin lisättiin kaksi inputkenttää muistion otsikolle ja itse muistiinpanolle. Otsikkokenttä on pakollinen tieto, jolla tallennetaan tietokantaan uusi rivi. Tietokanta tallennuksen jälkeen uusi listaus muistioista haetaan Reduxilla ja päivitetään varastoon sekä uudeksi tilaksi komponenttiin, jota käytetään listaukseen. Kuvassa 7 näkyy, miten listaus tulostetaan lisäyspainikkeen alapuolelle.



Kuva 7. PhotoNoteManager-näkymä

Listaus on itsessään varsin yksinkertainen. Se tulostaa muistion otsikon ja ikoneilla kameran, valokuvan, tekstieditorin ja ruksin. Kuvassa 8 havainnollistetaan listauksen painikkeiden toiminnallisuudet.

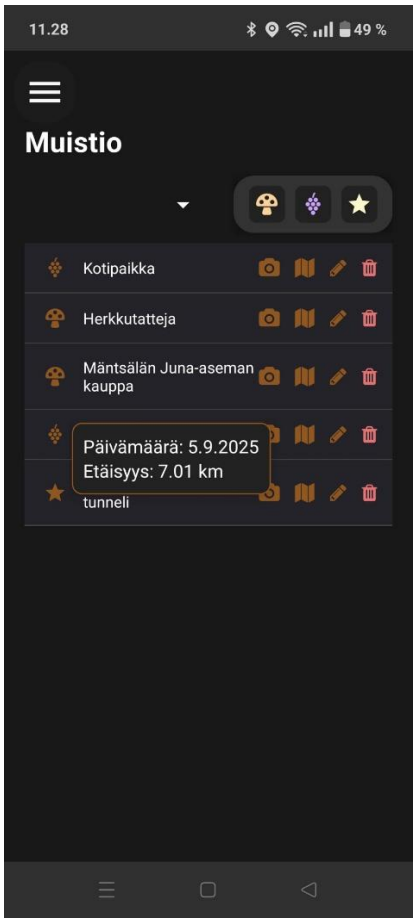


Kuva 8. Muistion valokuvamodaali

Kamera-ikonia painamalla aukeaa Camera-komponentti. Valokuva-ikoni näyttää nykyisen tallennetun kuvan modaalissa, tekstieditointi-ikoni avaa modaalin, jossa voi lukea koko muistiinpanon ja päivittää sen. Ruksi-ikonista avaa Alert-ikkunan, jonka vahvistuksen jälkeen poistetaan muistion rivi tietokannasta ja Redux-varastosta.

Lopuksi PhotoNoteManager-komponenttia refaktoroitiin vastaamaan paremmin LuontoSpot-sovelluksen tarpeita. Refaktoroinnin yhteydessä kielenvalintaominaisuus eriytettiin omaksi jaetuksi komponentikseen, jotta sitä voidaan tarvittaessa hyödyntää sovelluksessa useissa eri kohdissa. Lisäksi komponentin värimaailma mukautettiin LuontoSpotin visuaaliseen ilmeeseen ja sovelluksen teemaan sopivaksi, mikä paransi kokonaisuuden yhtenäisyyttä ja käytettävyyttä.

Komponentti päivitettiin niin, että sekä kuva ja teksti näkyvät samassa modaalissa. Näin Muistioriville saatiin tilaa lokaatio-ikonille, josta voi siirtyä kartalle. Lisäominaisuutena Muistion kuvaan lisättiin pinch-to-zoom ominaisuus, jolla kuvaa voi zoomata ja liikutella modaalissa. Näin kuvaa voi tarkastella sovelluksessa tarkemmin. Kuva 9 havainnollistaa julkaisuversion muistiolistauksen käyttöliittymän toiminnallisuudet.



Kuva 9. PhotoNoteMagerin lopullinen versio tummassa tilassa

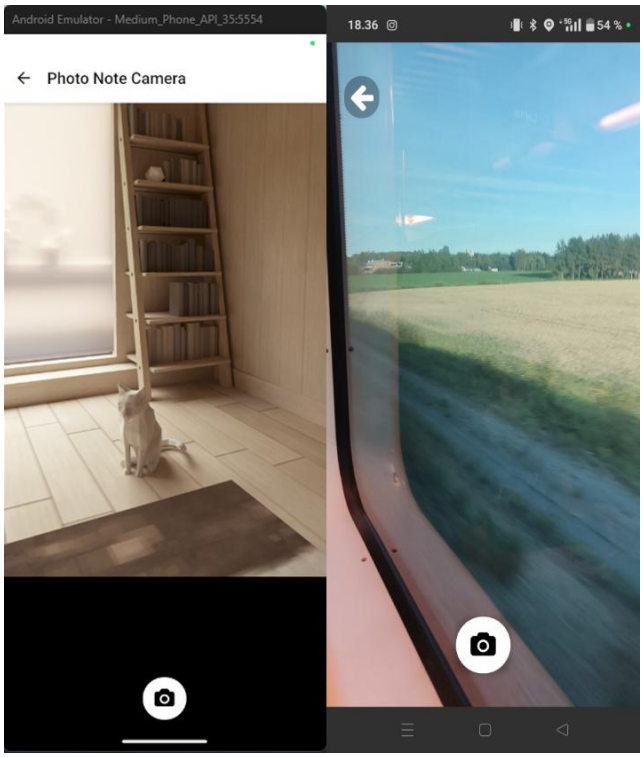
Lopuksi listausominaisuuteen päivitettiin merkinnän tyypin ikoni, jota klikkaamalla näkee muistiinpanon lisäyksen päivämäärän, sekä reaaliaikaisen etäisyyden kohteeseen. Listaukseen lisättiin suodatusominaisuus, jolla voi valita listattavat kohteet merkintöjen tyypeittäin. Suodatusta laajennettiin react-native-picker -komponentilla, jolla voi valita listauksen järjestyksen uusimman, vanhimman tai lähimmän kohteen perusteella.

3.3.2 Kamera

Camera-komponentti toteutettiin Vision Camera -kolmannen osapuolen kirjastolla. Vision Camera on varsin uusi ja vaatii sekä iOS 12:n tai Android-SDK 21:n tai uudemman toimiakseen. Vision Camera valittiin, koska se on edeltäjäänsä React Native Cameraa suorituskykyisempi ja sisältää monia toiminnallisuuksia, kuten kasvojentunnistuksen ja QR-koodin lukijan, jatkokehitystä ajatellen.

Vision Cameran asennus onnistui npm-komennolla kuten muidenkin liitännäisten, mutta paketin lisäksi piti Android-käännökseen käydä AndroidManifest.xml-tiedostoon lisäämässä oikeudet kameran ja vaihtoehtoisesti myös mikrofonin käyttöön. Näiden lupien kyselystä piti pitää huoli komponentissa, jotta kameraa voi käyttää.

Aluksi kameratoiminnallisuus toteutettiin yksinkertaisesti callback-funktion avulla, jossa kuva otettiin ja sen tiedostopolku palautettiin komponentille jatkokäsittelyä varten. Tämä ratkaisu kuitenkin sidottiin aluksi PhotoNoteManagerin logiikkaan. Toteutusta refaktoroitiin niin, että vastuu tietokantaan ja Redux-varastoon tallentamisesta siirrettiin kokonaan Camera-komponenttiin, jolloin siitä tuli itsenäinen ja uudelleenkäytettävä kokonaisuus. Refaktorointi mahdollisti kameran käytön myös Karttanäkymässä. Kuva 10 havainnollistaa kameran käyttöliittymän kehitysvaiheet.



Kuva 10. Alkuperäinen ja viimeistely kamerakomponentti

Alkuperäisessä käyttöliittymän suunnittelussa hyödynnettiin stack-navigaation vakioheaderia sekä alareunaan sijoitettua footeria, jossa oli toimintopainike. Lopullisessa toteutuksessa käyttöliittymästä pyrittiin tekemään mahdollisimman selkeä ja minimalistinen. Kameraan lisättiin erillinen paluupainike, ja footerin viemä tila poistettiin, jotta se ei rajoittaisi kameranäkymää. Ainoastaan kameran laukaisupainike säilytettiin. Lisäksi komponentti käärittiin SafeAreaContext-rakenteeseen, mikä estää sovellusta skaalautumasta Android-laitteiden järjestelmäpainikkeiden päälle. Tämä ero näkyy kuvassa oikealla, jossa lopullinen ratkaisu havainnollistuu.

3.3.3 Karttamerkinnät ja sijaintitiedot

Karttamerkinnät ja sijaintitiedot ovat LuontoSpot-sovelluksen keskeinen osa, sillä ne mahdollistavat käyttäjän omien havaintojen tallentamisen sekä niiden paikallistamisen kartalle. Sovellus hyödyntää puhelimen paikannusominaisuuksia ensin hakemalla käyttäjän nykyisen sijainnin ja sen jälkeen päivittämällä sitä jatkuvasti. Tämä mahdollistaa reaaliaikaisen liikkumisen seurannan kartalla ja varmistaa, että merkinnät sijoittuvat oikeaan paikkaan luonnossa kuljettaessa. Jatkuva paikannus toteutettiin mobiililaitteen sensoreihin pääsyn kautta, jolloin käyttäjän koordinaatit saadaan luotettavasti kartalle.

Suunnistusominaisuuksia sovelluksessa tukee kompassi, joka toteutettiin käyttäen kirjastoa react-native-compass-heading. Tämän avulla sovellus saa käyttöönsä puhelimen magnetometristä kerättyä dataa ja pystyy näyttämään käyttäjälle magneettisen pohjoisen suunnan. Toteutuksessa kompassin nuoli osoittaa aina pohjoiseen ja sen ympärillä oleva kehä on paikoillaan. Samaa headingiä käytetään myös kartalla oman katselusuunnan indikointiin, koska kartat tulevat APIn kautta aina vakiona kuva pohjoiseen päin.

Sovelluksen visuaalinen käyttö perustuu ulkoisen karttapalvelun käyttöön, josta ladataan taustakartat sovellukseen. Sovelluksessa hyödynnettiin Maanmittauslaitoksen karttapalvelua ja sen tekstikäyttöön tarkoitettua ilmaista rajapintaa. Rajapintaan hankittiin API-avain, joka tallennettiin vakiomuuttujaksi. API-avaimen tiedosto lisättiin .gitignore-tiedostoon, jotta se ei vuoda julkiseen tietoon versionhallinnan kautta. Rajapinnan avulla voidaan näyttää tarkat maastopohjat ja rakennetut ympäristöt. Kartalle voidaan lisäksi sijoittaa käyttäjän oma sijaintipiste, sekä käyttäjän omat merkinnät.

Merkinnät muodostavat sovelluksen käyttäjän näkökulmasta olennaisen toiminnallisuuden. Luonnosta tehdyt havainnot, paikat tai tapahtumat voidaan tallentaa kartalle visuaalisesti. Tällaisia merkintöjä on mahdollista luokitella tageilla, jolloin käyttäjä voi tarkemmin määrittää merkinnän sisällön. Tagit ovat Sieni, Marja tai Mielenkiinto. Näin käyttäjä voi merkata itselleen sienikarttoja, marjastuskarttoja, tai vaikkapa tehdä vaellusreitin merkkeineen ja

niihin liitettyine kuvineen. TAG-painikkeet lisättiin kartan alle, jota painamalla käyttäjän nykyinen sijainti tallennetaan tietokantaan, sekä Redux-varastoon, jonka jälkeen tagi ilmestyy kartalle. Kuva 11 näyttää lopullisen karttapalvelun käyttöliittymän.

Kuva 11. Karttapalvelun näkymän rakenne ja toiminnot

Karttanäkymä muodostaa laajemman käyttöliittymäkokonaisuuden. Sen yläreunassa sijaitsee erillinen drawer-komponentti, johon on toteutettu mukautettu navigaatioratkaisu. Navigoinnin yhteydessä on lisäksi integroitu kompassi. Kompassin vieressä sijaitsee automaattisen seurannan tilaa osoittava indikaattori. Sininen tähti ilmaisee aktiivista seurantaa ja harmaa tähti sen poiskytkentää. Lisäksi käyttöliittymässä on reaaliaikainen yhteysindikaattori, joka vaihtuu vihreäksi tai punaiseksi sen mukaan, onko kartta- ja sijaintitiedot päivittyneet viiden sekunnin sisällä.

Käyttöliittymän keskiosassa näkyy karttapalvelu, jossa esitetään käyttäjän sijainti sekä mahdolliset tallennetut kohteet. Kartan alalaidassa, osittain sen päällä, on painike, jolla käyttäjä voi kytkeä automaattisen seurannan päälle tai pois. Tähän liittyy myös suodatuslogiikka. Suodattimilla käyttäjä voi rajata, minkä tyyppiset tagimerkinnät kartalla näytetään. Alareunassa sijaitsee kolme erillistä tagipainiketta, joiden avulla käyttäjä voi lisätä uusia merkintöjä.

3.3.4 Sijaintien, kuvien ja muistiinpanojen liitokset

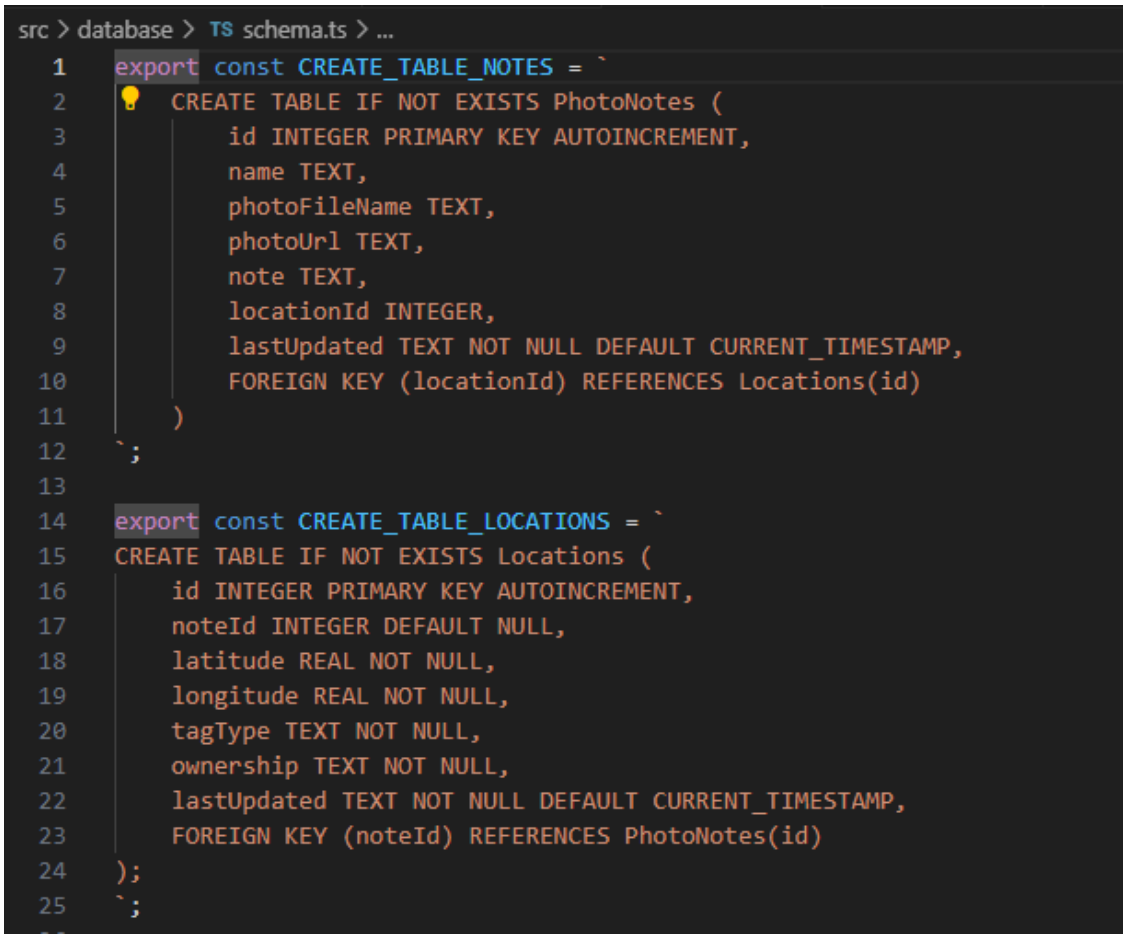
Komponenttien refaktoroinnissa sijainnin yhdistäminen muistiinpanoihin toteutettiin Reduxin ja SQLiten kanssa. PhotoNoteManager-komponentista irrotettiin uuden muistiinpanon lisäys, ja ominaisuus liitettiin kartan tagiin osaksi karttapalvelua. Kun tagi on lisätty sitä painamalla voi kyseisen kohteen poistaa, tai siihen voi lisätä muistiinpanon. Muistiinpanon lisäyksen jälkeen sitä voi muokata ja siihen voi ottaa valokuvan.

Redux varmistaa, että muistiinpanot ja niihin liittyvät sijaintitiedot pysyvät synkronoituna ja helposti hallittavana kokonaisuutena. Tarvittaessa käyttäjä voi poistaa tagin kartalta poistamalla vastaavan muistiinpanon myös PhotoNoteManagerin listauksesta, jolloin myös sijaintiin liittyvä tieto katoaa sovelluksen tilasta. Tämä yhtenäinen tilanhallinta helpottaa koodin ylläpitämistä ja varmistaa, että komponenttien välinen tiedonsiirto toimii luotettavasti ilman epäjohdonmukaisuuksia. Toteutusta optimoitiin siten, että kuvan päivityksen ja muistiinpanon poiston yhteydessä laitteelta poistetaan myös valokuvatiedosto, eikä vain tietokantarivin tietoa sen osoitteesta.

3.3.5 Lokaali tietojen tallennus

LuontoSpot-sovelluksen paikallinen tallennus pohjautuu kahden päätaulun käyttöön SQLite-tietokannassa: PhotoNotes ja Locations. PhotoNotes-taulu sisältää muistiinpanoihin liittyvät tiedot, kuten valokuvan tiedostonimen (photoFileName), kuvan URL-osoitteen (photoUrl), muistiinpanotekstin (note) sekä nimen (name). Tauluun tallennetaan myös viimeksi päivitetyn tiedon aikaleima (lastUpdated), joka saa oletuksena nykyisen ajan automaattisesti.

Toinen taulu, Locations, vastaa sijaintitiedoista ja sisältää sarakkeet paikan tunnisteelle (id), sijainnin nimen (title), koordinaateille (latitude ja longitude) sekä tyypitykselle (tagType). Lisäksi Location-taulussa on omistajuutta kuvaava kenttä (ownership) ja viimeksi päivitetyn ajan tallennus (lastUpdated). Taulujen välillä on viiteavainlinkitys (FOREIGN KEY), joka sitoo muistiinpanot ja sijainnit toisiinsa saumattomasti, mahdollistaen esimerkiksi muistiinpanoon liittyvän sijainnin haku sekä päinvastoin. Kuva 12 kuvaa sovelluksessa käytetyn tietomallin.



```
src > database > TS schema.ts > ...
export const CREATE_TABLE_NOTES = `
    CREATE TABLE IF NOT EXISTS PhotoNotes (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT,
        photoFileName TEXT,
        photoUrl TEXT,
        note TEXT,
        locationId INTEGER,
        lastUpdated TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (locationId) REFERENCES Locations(id)
    )
`;

export const CREATE_TABLE_LOCATIONS = `
CREATE TABLE IF NOT EXISTS Locations (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    noteId INTEGER DEFAULT NULL,
    latitude REAL NOT NULL,
    longitude REAL NOT NULL,
    tagType TEXT NOT NULL,
    ownership TEXT NOT NULL,
    lastUpdated TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (noteId) REFERENCES PhotoNotes(id)
);
`;
```

Kuva 12. Skeema tietomallista

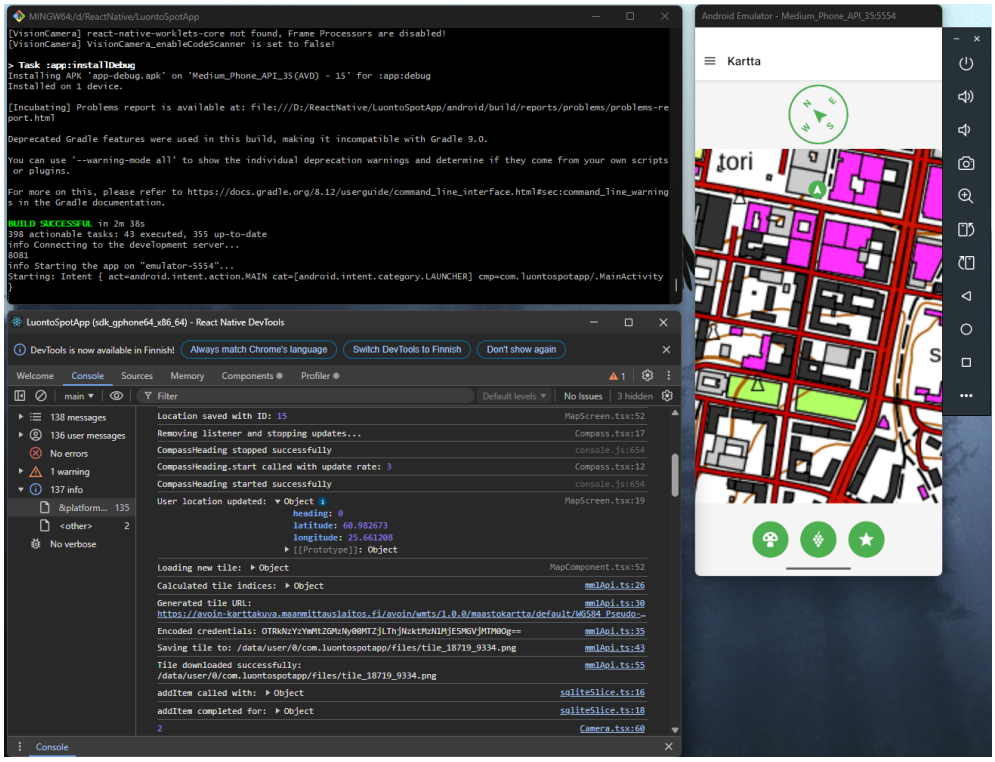
Tämä tietokantarakenne tukee sovelluksen tarvetta hallita sekä paikkatietoja että niihin liittyviä muistiinpanoja ja kuvia yhdistettynä. Skeema on suunniteltu myös laajennettavaksi, mikä mahdollistaa jatkokehityksen esimerkiksi ryhmäominaisuuksien tai lisämetatietojen lisäämisen osaksi tietomallia.

Lisäksi sovelluksessa hyödynnettiin AsyncStoragea erillisille asetustiedoille ja pienemmille sovelluksen käyttöparametreille. AsyncStorage on React Native -ympäristössä suosittu asynkroninen tallennusratkaisu, joka soveltuu kevyiden avain-arvo -parien säilyttämiseen. Tämä erottelu SQLite-tietokannasta mahdollistaa asetusten nopean käsittelyn ja päivityksen ilman raskaita tietokantaoperaatioita.

Kokonaisuutena paikallinen tallennusratkaisu yhdistää SQLite:n relaatiotietokannan monipuolisuuden ja AsyncStoragen keveyden. Nämä ratkaisut takaavat sekä luotettavan datan hallinnan että käyttäjäkokemuksen sujuvuuden offline-tilassa.

3.3.6 Sovelluksen testaus

Sovelluksen testaaminen aloitettiin emulaattoriympäristössä, joka mahdollisti kehityksen aikaisen käytön ja toiminnallisuuksien tarkistamisen nopeasti ilman erillisiä laitteita. Emulaattori tarjosi joustavan tavan simuloida eri näyttökokoja sekä käyttöjärjestelmäversioita, mikä helpotti erityisesti käyttöliittymän toimivuuden ja sovelluksen vakauden varmistamista. Kehitysvaiheessa tämä oli tehokas ratkaisu, sillä muutoksia voitiin kokeilla ja arvioida välittömästi. Kuva 13 havainnollistaa kehitystyötä.



Kuva 13. Sovelluksen testaaminen emulaattorilla

Kehitystyö tapahtui kahdella näytöllä. Toisessa näytössä oli Visual Studio Code ja toisella näytöllä oli Git Bash -komentorivityökalu. React Developer tools, sekä emulaattori. Git Bashissa ajettiin emulaattorin käynnistys ja konfigurointi komennot, Developer toolsilla seurattiin logituksia sekä virhe- ja varoitusilmoituksia. Emulaattori toimi Fast Refresh toiminnallisuudella Metro bundlerin välityksellä. Eli aina, kun lähdekoodia muokataan, kehityspalvelin Metro tunnistaa muutoksen, jolloin muutettu koodi päivittyy emulaattorille ilman, että sovellusta tarvitsee aina kääntää uudelleen. Fast Refresh on nykyään React Nativen oletus ominaisuus, joka yhdistää Hot Reloadingin sekä Live Reloadingin ominaisuudet.

Testauksen viimeisessä vaiheessa sovellusta suoritettiin myös fyysisellä puhelimella, jotta sen käytettävyys ja toiminta voitiin arvioida todellisissa käyttötilanteissa. Tätä varten sovelluksesta koottiin erillinen APK-paketti, joka asennettiin testilaitteelle. Puhelimella testaaminen oli erityisen tärkeää, sillä se mahdollisti sovelluksen käytön luonnossa, jossa

esimerkiksi kameran toiminta, GPS-paikannus ja käyttöliittymän selkeys saattoivat näyttäytyä eri tavalla kuin emulaattorissa. Näin varmistettiin, että sovellus tukee käyttäjien tarpeita myös todellisessa toimintaympäristössä.

3.3.7 Jatkokehitys

Jatkokehityksen näkökulmasta yksi merkittävä lisäominaisuus voisi olla puheentunnistuksen (speech-to-text) käyttöönotto, joka helpottaisi muistiinpanojen tekemistä erityisesti luonnossa liikkuessa. Näin käyttäjä voisi tallentaa havaintonsa nopeasti ilman tarvetta kirjoittaa niitä manuaalisesti mobiililaitteella, mikä parantaisi käytettävyyttä ja saavutettavuutta esimerkiksi liikkuvissa tilanteissa.

Sovelluksen arkkitehtuuria voitaisiin laajentaa hyödyntämällä ulkoista rajapintaa (API) datan säilyttämiseen. Tällä tavoin käyttäjien tiedot ja havainnot eivät olisi sidottuja yksittäiseen laitteeseen, vaan ne voitaisiin säilyttää keskitetysti ja hakea käyttöön useilta eri päätelaitteilta. Tämä toisi sovellukseen skaalautuvuutta ja mahdollistaisi suurempien tietomäärien hallinnan luotettavasti.

Käyttäjäkohtaiset ominaisuudet voisivat puolestaan rakentua autentikoinnin varaan, esimerkiksi hyödyntämällä Oauth2-standardeja. Tämä mahdollistaisi yksilöllisten profiilien luomisen, mikä avaisi mahdollisuuden tarkemmille personointimahdollisuuksille ja käyttäjien väliselle toiminnallisuudelle. Yhtenä jatkokehitysmahdollisuutena voidaan nähdä myös ryhmien luominen ja tietojen jakaminen käyttäjien kesken, mikä tukisi yhteisöllistä luontohavaintojen tallentamista ja vertaisoppimista.

Tekoälyominaisuuksia voitaisiin lisätä muun muassa reaaliaikainen kohteiden, kuten sienten tai marjojen tunnistus maastosta, joko otetuista kuvista tai kameran näkymästä. Toinen tekoälyominaisuus voisi olla sienten tai marjojen lajitunnistus, mutta tämä pitäisi suunnitella niin selkeäksi, että käyttäjä ymmärtää oman vastuunsa väärän tunnistuksen tullen, sekä ominaisuus kuvan jakoon ja ihmisen antamiin tunnistuksiin (vertaisoppiminen). Tällaisten tunnistettujen kuvien käyttö voisi taas vastaisuudessa toimia sieni- ja marjamallien jatkokoulutukseen.

Lisäksi jatkokehityksessä on tärkeää huomioida GDPR-lainsäädännön asettamat vaatimukset. Käyttäjien tuottamien tiedostojen säilyttäminen edellyttää tietoturvasta ja tietosuojasta huolehtimista, ja erityisesti mahdollinen datan jatkokäyttö tai jälleenmyynti tulee arvioida huolellisesti juridisesta ja eettisestä näkökulmasta. Tämän vuoksi jatkokehitysvaiheessa on suositeltavaa selvittää tarkemmin tietosuojaan liittyvät riskit sekä luoda selkeät käytännöt käyttäjien oikeuksien turvaamiseksi.

4 Yhteenveto ja pohdinta

Tässä opinnäytetyössä on kehitetty LuontoSpot-niminen hybridimobiilisovellus luonnonpaikkojen kartoitukseen, valokuvien, muistiinpanojen ja paikkatietojen yhdistämiseen React Native -kehystä ja Redux-tilanhallintaa hyödyntäen. Sovellus mahdollistaa käyttäjien tallentaa omia havaintojaan luonnosta, liittää niihin kuvia ja tekstiä sekä merkitä ne kartalle interaktiivisella ja intuitiivisella käyttöliittymällä. Sovelluksesta on käännetty APK-paketti.

Projektin toteutus on osoittanut React Nativen vahvuuden joustavana ja modulaarisena kehitysalustana, jolla voidaan rakentaa tehokkaita ja skaalautuvia mobiilisovelluksia yhdellä yhteisellä koodipohjalla Android- ja iOS-alustoille. Modulaarinen komponenttipohjainen arkkitehtuuri mahdollistaa koodin uudelleenkäytettävyyden, mikä nopeuttaa kehitystyötä ja helpottaa ylläpitoa ja tarjoaisi hyvät mahdollisuudet myös useamman kehittäjän samanaikaiselle tiimityöskentelylle. Redux tarjoaa yhtenäisen tavan hallita sovelluksen globaalia tilaa, mikä selkeyttää tilahallintalogiikkaa ja tukee vakaata käyttäjäkokemusta erityisesti monimutkaisissa toiminnallisuuksissa, kuten sijaintien, kuvien ja muistiinpanojen synkronoinnissa. AsyncStore toimi kevyenä ja käytännöllisenä tapana tallentaa yksinkertaisia asioita, kuten sovelluksen kielivalinnan.

Toteutuksen aikana esiin nousi haasteita etenkin kolmannen osapuolen natiivikomponenttien integroinnissa, kuten kameran ja karttapalveluiden kanssa. Vaikka näiden kirjastojen käyttö nopeutti kehitystä merkittävästi, ne toivat mukanaan myös riippuvuuksia ja ylläpidollisia haasteita, joita tulee huomioida jatkokehityksessä. React Developer Tools ja Redux DevTools olivat korvaamattomia apuvälineitä sovelluksen tilan seurannan, virheiden jäljityksen ja suorituskyvyn optimoinnin tukena.

Jatkokehityksen näkökulmasta LuontoSpotissa on potentiaalia monipuolistaa sovelluksen ominaisuuksia erityisesti tekoälyn avulla. Tekoälyominaisuuksien, kuten reaaliaikaisen sienten tunnistuksen suoraan maastossa otetuista kuvista ja puheentunnistuksen integroiminen muistiinpanojen nopeuttamiseksi, parantaisi käyttäjäkokemusta merkittävästi. Nämä älykkäät toiminnot helpottaisivat luonnonhavaintojen tekemistä ja lisäisivät sovelluksen käytännön hyödyllisyyttä.

Kokonaisuutena projekti vahvistaa React Nativen ja Reduxin asemaa tehokkaina työkaluina monipuolisissa mobiilisovellusprojekteissa. Vaikka hybridikehitykseen liittyy omia haasteitaan, sen tarjoama nopea kehityssykli ja alustariippumattomuus tekevät LuontoSpotista joustavan ja laajennettavan ratkaisun. Tulevaisuudessa on kiinnostavaa seurata, miten React Native ja sen ekosysteemi kehittyvät ja kuinka ne pärjäävät kilpaillussa mobiilikehityksen kentässä.

Lähteet

Austin, D. 2025. A Brief History of TypeScript: From Origin to Modern Adoption. Medium. Viitattu 10.2.2025. Saatavissa https://medium.com/totally-typescript/a-brief-history-of-typescript-from-origin-to-modern-adoption-791368ec4b91

Boduch, A. & Derks, R. 2020. React and React Native: A Complete Hands-on Guide to Modern Web and Mobile Development with React.js. 3. painos. Birmingham: Packt Publishing.

Devopedia. 2022a. React Hooks. Viitattu 17.2.2025. Saatavissa https://devopedia.org/react-hooks

Devopedia. 2022b. React Native. Viitattu 17.2.2025. Saatavissa https://devopedia.org/react-native

Euroopan komissio. 2019. Luotettavaa tekoälyä koskevat eettiset ohjeet. Viitattu 26.8.2025. Saatavissa https://www.europarl.europa.eu/meetdocs/2014_2019/plmrep/COMMITTEES/JURI/DV/2019/11-06/Ethics-guidelines-AI_FI.pdf

Euroopan parlamentin ja neuvoston asetus (EU) 2016/679. 2018. Yleinen tietosuoja-asetus (GDPR). Viitattu 26.8.2025. Saatavissa https://eur-lex.europa.eu/legal-content/FI/TXT/?uri=CELEX%3A32016R0679

Firofame. 2025. react-native-compass-heading. npm. Viitattu 17.2.2025. Saatavissa https://www.npmjs.com/package/react-native-compass-heading

Flatirons. 2024. Understanding the Fundamental Basics of Redux in State Management Viitattu 16.3.2025. Saatavissa https://flatirons.com/blog/what-is-redux/

FormidableLabs. 2025. react-native-app-auth. GitHub. Viitattu 15.2.2025. Saatavissa https://github.com/FormidableLabs/react-native-app-auth

Google Cloud Vision API. 2025. Google Cloud. Viitattu 26.8.2025. Saatavissa https://cloud.google.com/vision/docs

Green, O. 2023. Where Is the Virtual DOM Stored in React? Mobile App Circular. Viitattu 10.2.2025. Saatavissa https://mobileappcircular.com/where-is-the-virtual-dom-stored-in-react-41724bc09f4f

Hafkenschiel, I. 2023. Mastering State Management in React Native: A Comprehensive Guide. Viitattu 16.3.2025. Saatavissa https://blog.tigerbytestudio.com/mastering-state-management-in-react-native-a-comprehensive-guide/

Helsingin Yliopisto. 2024. Generatiivisen tekoälyn käyttö tutkimuksessa. Helsingin Yliopisto. Viitattu 26.8.2025. Saatavissa https://www.helsinki.fi/fi/tutkimus/vastuullinen-tiede/tekoalyn-kaytto-tutkimuksessa

Howard, J., Gugger, S. 2017. Deep Learning for Coders with fastai and PyTorch. O'Reilly Media. Viitattu 26.8.2025. Saatavissa https://course.fast.ai/Resources/book.html

Kettunen, A. (2023). Jaetut UI-komponentit. Diplomityö. Tampereen yliopisto. Viitattu 17.2.2025. Saatavissa https://trepo.tuni.fi/bitstream/10024/146261/2/KettunenAnnina.pdf

Krizhevsky, A., Sutskever, I., Hinton, G.E. 2012. ImageNet Classification with Deep Convolution1al Neural Networks. Institute of Electrical and Electronics Engineers. Viitattu 26.8.2025. Saatavissa https://proceedings.neurips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

LiteRT. 2025. Developer Guide. Google. Viitattu 26.8.2025. Saatavissa https://www.tensorflow.org/lite

LUT. 2024. Mitä on vastuullinen tekoäly? – LUT tutkii ja kehittää eettisiä ja tehokkaita ratkaisuja. LUT-yliopisto. Viitattu 26.8.2025. Saatavissa https://www.lut.fi/fi/artikkelit/mita-vastuullinen-tekoaly-lut-tutkii-ja-kehittaa-eettisia-ja-tehokkaita-ratkaisuja

Obregon, A. 2023. A Deep Dive into Redux DevTools: Debugging and Analyzing Your Application’s State. Medium. Viitattu 16.3.2025. Saatavissa https://medium.com/@AlexanderObregon/a-deep-dive-into-redux-devtools-debugging-and-analyzing-your-applications-state-b634ead3927b

Pitkänen, S. 2021. React-komponettien jakaminen sovellusten kesken. Viitattu 17.2.2025. Saatavissa https://www.theseus.fi/bitstream/10024/512664/3/Pitkanen_Sami.pdf

Punosmobile. 2023. Ohjelmointikielet Suomen mobiilikehityksessä 2020: React Native. Viitattu 15.2.2025. Saatavissa https://www.punosmobile.com/blog/ohjelmointikielet-suomen-mobiilikehityksessa-2020-osa-4-react-native/

Rafalski, K. 2025. Top Tips to Boost React Native Performance in 2025. Viitattu 17.2.2025. Saatavissa https://www.netguru.com/blog/react-native-performance

React. 2025a. React Hooks. Viitattu 17.2.2025. saatavissa https://react.dev/reference/react/hooks

React. 2025b. React memo. Viitattu 17.2.2025.
Saatavissa https://react.dev/reference/react/memo

React. 2025c. Lazy. Viitattu 7.9.2025. Saatavissa
https://react.dev/reference/react/lazy#suspense-for-code-splitting

React. 2025d. React Compiler. Viitattu 7.9.2025.
Saatavissa https://react.dev/learn/react-compiler/introduction

React Native. 2025a. Using Type Script. Viitattu 17.2.2025.
Saatavilla https://reactnative.dev/docs/typescript

React Native. 2025b. React Native Components Diagram. Viitattu 17.2.2025.
Saatavilla https://reactnative.dev/docs/intro-react-native-components.

React Native. 2025c. React Native DevTools. Viitattu 17.2.2025.
Saatavissa https://reactnative.dev/docs/react-native-devtools

React Native. 2025d. Native Modules Intro. Viitattu 17.2.2025.
Saatavissa https://reactnative.dev/docs/turbo-native-modules-introduction

React Native Maps. 2025. React Native Mapview component for iOS + Android. GitHub.
Viitattu 17.2.2025. Saatavissa https://github.com/react-native-maps/react-native-maps

React v19. 2024. React 19 is now stable. Viitattu 17.2.2025.
Saatavissa https://react.dev/blog/2024/12/05/react-19

Redux. 2025. Redux Essentials, Part 2: Redux Toolkit App Structure. Viitattu 16.3.2025.
Saatavissa https://redux.js.org/tutorials/essentials/part-2-app-structure

RisingStack Engineering. 2025. The History of React.js on a Timeline. Viitattu 10.2.2025.
Saatavissa https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/

Robinson-Adams, S. 2022. React Native's bridge, under the hood. Medium.
Viitattu 15.2.2025. Saatavissa https://medium.com/@samwhadams/react-native-under-the-hood-80677cf9bf96

ScolarHat. 2025. Geting Started With React Redux part 1. Viitattu 16.3.2025
Saatavissa https://www.scholarhat.com/tutorial/react/getting-started-with-react-redux

Sirviö, J. 2022. Kohti eettisesti toimivaa tekoälyn kehittämis-ympäristöä. YAMK opinnäytetyö. Viitattu 26.8.2025. Saatavissa https://www.theseus.fi/bitstream/handle/10024/752332/Sirvi%C3%B6_Jonna.pdf?sequence=2

The Codest. 2025. React Kehitys: Kaikki mitä sinun on tiedettävä. Viitattu 10.2.2025. Saatavissa https://thecodest.co/fi/blog/reagoida-kehittamiseen-kaikki-mita-sinun-on-tiedettava/

Wäldchen, J., Rzanny, M., Seeland, M., Mäder, P. 2018. Automated plant species identification—Trends and future directions. PLoS Computational Biology. Viitattu 26.8.2025. Saatavissa https://pmc.ncbi.nlm.nih.gov/articles/PMC5886388/