Google Maps -karttojen ja ohjelmointirajapintojen käyttö Android -mobiiliohjelmoinnissa

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus

Syksy 2024

Harri Saros

HAMK
Hämeen ammatti-
korkeakoulu

Tietojenkäsittelyn koulutus Tiivistelmä
Tekijä Harri Saros Vuosi 2024
Työn nimi Google Maps -karttojen ja ohjelmointirajapintojen käyttö Android -mobiiliohjelmoinnissa
Ohjaaja Pentti Ojaniemi

Opinnäytetyön tavoitteena oli perehtyä Google Mapsin ohjelmointirajapintoihin sekä selvittää, miten niiden avulla voidaan hakea kartta- ja mobiililaitteen sijaintitietoja ja miten nämä tiedot voidaan integroida mobiilisovellukseen. Lisäksi tarkoituksena oli kehittää opinnäytetyön tekijän osaamista mobiiliohjelmoinnista.

Opinnäytetyön teoreettisessa osuudessa käsiteltiin ohjelmointirajapintojen perusteita, niiden tarjoamia hyötyjä ja toimintaperiaatteita. Teoreettisessa osuudessa käsiteltiin myös Google Mapsin karttapalvelua ja sen tarjoamia ohjelmointirajapintoja.

Käytännön osuudessa käytiin läpi, kuinka Google Mapsin sekä mobiililaitteen ohjelmointirajapintoja hyödyntämällä voitiin hakea karttatietoja sekä käyttäjän sijaintietoja. Näiden tietojen pohjalta suunniteltiin, ohjelmoitiin ja otettiin käyttöön lenkkeilysovellus React Nativella Android -alustalle. Sovellus seuraa käyttäjän sijaintia, näyttää sen kartalla piirtäen kuljetun reitin ja tallentaa lenkkiin liittyvät tiedot puhelimen SQLite-tietokantaan.

Opinnäytetyö onnistui vastaamaan esitettyihin tutkimuskysymyksiin ja tulokset osoittavat, kuinka mobiilisovellus toteutettiin React Nativella, mitä sen toteutuksessa tuli ottaa huomioon ja miten ohjelmointirajapintoja käytettiin React Nativessa.

Avainsanat React Native, Google Maps -ohjelmointirajapinnat, SQLite, mobiiliohjelmointi
Sivut 57 sivua ja liitteitä 1 sivua

HAMK
Hämeen ammatti-
korkeakoulu

Degree Programme in Business Information Technology
Author Harri Saros
Subject Using Google Maps and application programming interfaces in Android mobile programming
Supervisors Pentti Ojaniemi

Abstract
Year 2024

The aim of the thesis was to become familiar with Google Maps application programming interfaces, to find out how they can be used to retrieve map and mobile device location information and how this information can be integrated into the into a mobile application. In addition, the aim was to develop the thesis author's skills in mobile programming.

The theoretical part of the thesis covered the basics of application programming interfaces, their benefits and principles of operation. The theoretical part of the thesis also covered the map service of Google Maps, and the programming interfaces it provides.

The practical part examined how to use Google Maps and the mobile device programming interfaces to retrieve map information and user location data. This information was used to design, program and deploy a jogging application on React Native for Android -platform. The application tracks the user's location, displays it on a map and drawing the traveled route and stores the information related to the run in the phone's SQLite database.

The thesis succeeded in answering the research questions posed and the results show how the mobile application was implemented using React Native, what had to be taken into account in its implementation and how the application programming interfaces were used in React Native.

Keywords React Native, Google Maps application programming interfaces, SQLite, mobile programming
Pages 57 pages and appendices 1 pages

Sanasto

API	Ohjelmointirajapinta (Application Programming Interface), jonka avulla mahdollistetaan sovellusten tai ohjelmien välinen kommunikointi keskenään. Mahdollistaa erilaisten sovellusten integroinnin ja yhteistoiminnan.
HTTP-protokolla	Kommunikointiprotokolla, joka mahdollistaa tietojen siirtämisen web-palvelimen ja selaimen välillä.
JavaScript	Verkkosivujen ja -sovellusten kehittämiseen käytetty ohjelmointikieli.
JSON	(JavaScript Object Notation) Tiedonvälitysmuoto, jota käytetään tietojen siirtämiseen ja tallentamiseen tietokannoissa, sovelluksissa ja verkkopalveluissa.
Kirjasto (ohjelmoinnissa)	Kokoelma valmiiksi kirjoitettuja ohjelmakoodin osia, jotka voivat sisältää funktioita, luokkia, menetelmiä ja muuttujia.
Komponentti	Itsenäinen ja uudelleenkäytettävä koodiyksikkö, joka sisältää sekä logiikan että käyttöliittymän osat tietyn toiminnallisuuden toteuttamiseksi.
Natiivisovellus	Sovellus, joka on kehitetty erityisesti tietty käyttöjärjestelmä tai laitteisto mielessä, ja joka toimii suoraan tämän alustan kanssa ilman välittävää ohjelmointikerrosta.
React Native	Avoimen lähdekoodin alusta, jonka avulla kehittäjät voivat rakentaa ristialustaisia mobiilisovelluksia käyttämällä JavaScriptiä ja React -kirjastoa.
REST	(Representational State Transfer) Arkkitehtuurimalli, joka tarjoaa periaatteet resurssien käytölle ja kommunikoinnille hajautetuissa järjestelmissä.

SDK	(Software Development Kit) Ohjelmistokehityksen työkalupaketti, jonka avulla voidaan luoda sovelluksia, ohjelmistoja ja kehyksiä nopeammin ja tehokkaammin. SDK:t tarjoavat valmiita komponentteja ja toiminnallisuuksia.
URI	(Uniform Resource Identifier) Merkkijono, joka identifioi resursseja yksilöllisesti maailmanlaajuisesti.
XML	(Extensive Markup Language) Merkintäkieli, jota käytetään tietojen tallentamiseen ja siirtämiseen tietokonejärjestelmien välillä.

Sisällys

Kuvat, komennot, ohjelmakoodit, taulukot ja kaavat

Liitteet

1 Johdanto

Monet sovellukset ja verkkosivustot, joita käytämme päivittäin, hyödyntävät ohjelmointirajapintoja. Nykyään kuka tahansa voi ohjelmointirajapintojen ansiosta esimerkiksi ostaa lentolippuja, tarkistaa sään mistä tahansa maailmasta tai tehdä ravintolavarauksia verkossa. Tässä opinnäytetyössä perehdytään ohjelmointirajapintoihin, tarkastellaan Google Mapsin tarjoamia ohjelmointirajapintoja ja tutkitaan, kuinka niiden avulla voidaan hakea karttatietoja sekä mobiililaitteen sijaintitietoja ja miten nämä rajapinnat saadaan integroitua mobiilisovellukseen. Opinnäytetyön tavoitteena on myös saada kokemusta sovelluskehityksestä ja kehittää omaa osaamista mobiiliohjelmoinnista.

Opinnäytetyön teoreettisessa osuudessa kerrotaan mitä ohjelmointirajapinnat ovat, mitä hyötyä niistä on, miten ne toimivat ja miten niitä voidaan hyödyntää etenkin sovelluskehityksessä. Teoreettisessa osuudessa käsitellään myös lyhyesti Google Mapsin karttapalvelua ja Google Mapsin tarjoamia ohjelmointirajapintoja.

Opinnäytetyön käytännön osuudessa tavoitteena on suunnitella, ohjelmoida ja ottaa käyttöön yksinkertainen hyvinvointisovelluksen prototyyppi. Sovelluksen tarkoituksena on näyttää käytännössä, miten puhelimen sijaintietoja saadaan haettua ja tuotua mobiilisovellukseen sekä miten Google Mapsin rajapintojen avulla saadaan haettua karttatietoja ja miten ne voidaan tallentaa ja yhdistää mobiilisovellukseen. Opinnäytetyön teoreettisen osuuden aikana pyritään selvittämään, mikä Googlen tarjoama rajapinta soveltuisi parhaiten sovelluksen tarpeisiin. Suunnitelmana on luoda lenkkeilysovellus, joka seuraa käyttäjän sijaintia, näyttää sen kartalla, piirtää karttaan käyttäjän kulkeman lenkin sekä tallentaa puhelimen tietokantaan lenkkiin liittyviä tietoja kuten lenkin pituuden, lenkkiin käytetyn ajan ja muuta käyttäjää kiinnostavia tietoja. Sovellus tallentaa nämä tiedot tietokantaan niin, että käyttäjä voi niitä tarkastella myöhemmin.

Opinnäytetyössä ohjelmoitava mobiilisovellus ohjelmoidaan käyttäen React Nativea. Ohjelmointiympäristönä käytetään Visual Studio Codea. Mobiilisovellukselle luotuun tietokantaan käytetään tietokantaratkaisuna SQLiteä. Mobiilisovelluksen testaukseen käytetään Android Studio -emulaattoria ja fyysisiä älypuhelimia.

Opinnäytetyöstä rajataan pois, kuinka ohjelmointirajapinta voidaan itse luoda, ja keskitytään siihen, miten ohjelmointirajapinnat toimivat ja miten valmis ohjelmointirajapinta voidaan integroida omaan sovellukseen sovelluskehittäjän näkökulmasta. Tämän opinnäytetyön käytännön osuus on toteutettu käyttämällä REST API-arkkitehtuuria.

Opinnäytetyössä pohditaan kysymyksiä:

- Kuinka Google Maps saadaan integroitua ohjelmointirajapintojen avulla Android -sovellukseen?
- Kuinka mobiilisovellus kehitetään React Nativella?
- Mitä mobiilisovelluksen toteuttamisessa tulee ottaa huomioon?

2 Ohjelmointirajapinnat

Ohjelmointirajapintojen eli APIen (Application Programming Interface) avulla eri sovellukset ja verkkosivustot voivat kommunikoida ja jakaa tietoja keskenään. Niitä käytetään päivittäin erilaisissa sovelluksissa ja internetsivuilla. Nykyään ohjelmointirajapintoja on laajalti käytössä, eikä loppukäyttäjä edes huomaa niiden olemassaoloa ohjelmistoa tai palvelua käyttäessään.

Ohjelmointirajapintojen toimintaa kuvataan usein esimerkillä ravintolan asiakkaalle tarjoamista palveluista. Ravintolan asiakkaana ollaan vuorovaikutuksessa tarjoilijan kanssa ravintolan tarjoamien palveluiden saamiseksi. Tarjoilijalta voidaan tilata ruokaa ja juomaa, kysyä ruokalistasta, pyytää lasku ja maksaa se sekä tiedustella muita ravintolapalveluita. Kun ruokaa tilataan, vie tarjoilija tilauksen keittiöön, jossa kokit valmistavat tilatun aterian ja tarjoilija tuo tilatun aterian pöytään. Tarjoilija tarjoaa siis tavan olla vuorovaikutuksessa ravintolan kanssa ilman tarvetta tuntea kaikkia monimutkaisia prosesseja, joita ravintolan kulissien takana tapahtuu. Asiakkaan ei siksi tarvitse huolehtia muun muassa ravintolan resepteistä, keittiön laitteista, kulujen laskemisesta, ruoan tarjoilusta tai muista ravintolan toimintaan liittyvistä asioista. Ravintolatasoisen ruoan valmistamisen taitoa ei myöskään tarvita, koska ravintolassa on nämä palvelut tehty valmiiksi, ja tarjoilijan kautta nämä palvelut saadaan käyttöön. Tarjoilija toimii siis rajapintana asiakkaan ja ravintolan palveluiden välillä (Mlytics, n.d.).

Ohjelmointirajapinnat toimivat kuten tarjoilija ravintolassa, yhdistäen eri palvelut ja ohjelmistot toisiinsa. Ne mahdollistavat järjestelmien ja sovellusten välisen vuorovaikutuksen ja tietojen jakamisen tehokkaasti ja joustavasti. Ohjelmointirajapinta koostuu ohjeista ja säännöistä, joita sekä asiakas että palvelin ymmärtävät, ja jotka määrittelevät, kuinka yksi ohjelmisto voi käyttää toisen ohjelmiston tarjoamia tietoja ja toimintoja. Ohjelmointirajapinta toimii verkkopalvelimen ja sovelluksen välissä, välittäen viestejä asiakkaan ja palvelimen välillä. Ohjelmointirajapinnan voidaan ajatella toimivan myös kääntäjänä, joka auttaa eri kieliä puhuvia ihmisiä ymmärtämään toisiaan. Samoin, kun kaksi sovellusta haluaa kommunikoida keskenään, ohjelmointirajapinta auttaa niitä ymmärtämään toistensa pyyntöjä ja vastauksia.

2.1 Ohjelmointirajapintojen historiaa

Ajatus ohjelmointirajapinnasta syntyi jo 1950-luvulla. Termi mainittiin ensimmäisen kerran Maurice Wilkesin ja David Wheelerin vuonna 1951 kirjoittamassa kirjassa The Preparation of Programs for an Electronic Digital Computer (Mikula, 2023). Kirjassa esiteltiin useita keskeisiä tietojenkäsittelyn termejä, mukaan lukien ohjelmointirajapintojen varhainen versio.

Ohjelmointirajapinnat rajoittuivat tässä vaiheessa yksinkertaisiin komentokehotteisiin, joiden avulla ohjelmoijat pystyivät toimimaan vuorovaikutuksessa varhaisten tietokoneiden kanssa.

Tietokoneiden suosio alkoi kasvaa 1960-luvun aikana. Tuolloin termi ohjelmointirajapinta ymmärrettiin yksittäisen sovelluksen ja muun tietokonejärjestelmän vuorovaikutukseksi. Ohjelmointirajapintojen avulla voitiin kommunikoida suurtietokoneiden ja muiden järjestelmien, kuten päätelaitteiden ja tulostimien, välillä. Tietokoneverkot alkoivat yleistyä 1980-luvulla ja ohjelmoijat tarvitsivat pääsyn kirjastoihin, jotka sijaitsivat heidän paikallisissa tietokoneissaan tai muualla olevissa tietokoneissa. Etäproseduurikutsun näihin kirjastoihin mahdollisti RPC-ohjelmointirajapinta (Mikula, 2023).

Internetin yleistyessä 1990-luvulla, ohjelmointirajapinnat antoivat sovelluksille mahdollisuuden vaihtaa tietoja muiden tietokonejärjestelmien sovellusten kanssa internetin kautta, sen sijaan että viestintä olisi rajoittunut oman tietokonejärjestelmän sisälle. Ohjelmointirajapinnat olivat tuolloin yhä kehitysvaiheessa. Vuonna 2000 Roy Fielding esitteli väitöskirjassaan REST-protokollan, joka määritteli standardin laitteiden väliselle kommunikoinnille internetissä (Mikula, 2023). Verkkosovellusten suosion kasvaessa organisaatiot alkoivat siirtää palveluitaan pilvipalveluihin. Salesforce, eBay ja Amazon olivat edelläkävijöitä palvelujen toimittamisessa käyttäen HTTP:tä, mikä tarjosi pääsyn koneellisesti luettavaan dataan JSON- tai XML-muodossa web-rajapintojen kautta. Apple julkaisi iPhonen vuonna 2007, jolloin APIt olivat jo merkittävästi muuttaneet tapaa, jolla yritykset käyttävät infrastruktuuria (Mikula, 2023).

Vuoteen 2010 mennessä sosiaalinen media oli saavuttanut laajan suosion, ja sen sovellukset loivat perustan uuden sukupolven ohjelmointirajapinnoille. Kun tehokkaiden, kustannustehokkaiden ja hyvin skaalautuvien sovellusten kysyntä kasvoi, tarjosivat ohjelmointirajapinnat organisaatioille helpon tavan integroida tietojärjestelmänsä kolmansien osapuolten palveluihin ja ohjelmointirajapintoihin perustuviin pilvialustoihin (Mikula, 2023).

Kun COVID-19 lisäsi riippuvuuttamme verkkopalveluista, ohjelmointirajapintojen käyttö lisääntyi entisestään, kun järjestelmiä jaettiin yhä useampiin mikropalveluihin (Mikula, 2023). Nykyään ohjelmointirajapinnat ovat keskeisiä esineiden internet -laitteiden (IoT, Internet of Things) kehityksessä, rakentaen toisiinsa kytketyn, mutta hajautetun tietomaailman. Ne ovat myös yhä tärkeämpiä tekoälyn kehittämisessä. Organisaatiot hyödyntävät sovelluksia eri pilvipalveluntarjoajien hajautetuissa järjestelmissä, ja tehokkaat ohjelmointirajapinnat ovat avainasemassa, kun halutaan varmistaa, että kaikki palvelut voivat kommunikoida keskenään (Mikula, 2023).

2.2 Ohjelmointirajapintojen käytön hyödyt

Ohjelmointirajapintojen tarpeellisuus korostuu erityisesti, kun tarkastellaan erilaisia internetin sääpalveluita. Ilman ohjelmointirajapintoja, jokaisen sääpalveluita tarjoavan yrityksen olisi muun muassa luotava tyhjästä oma infrastruktuuri, rakennettava sääsatelliitteja ja asennettava lämpötila-antureita kaikkialle maailmaan. Tämä olisi taloudellisesti kestämätöntä. Sen sijaan, että ”pyörää keksitään uudelleen”, ohjelmointirajapintojen avulla sovelluskehittäjät voivat hyödyntää valmista, olemassa olevaa palvelua ja näin keskittyä muihin oleellisiin asioihin kuten uusien ominaisuuksien rakentamiseen (Nylas, n.d.). Näin toimivatkin useat yritykset kuten yhdysvaltalainen henkilökuljetuspalvelua tarjoava Uber, joka käyttää Google Mapsin ohjelmointirajapintaa sijaintitietojen hakemiseen sen sijaan, että olisi kehittänyt oman karttajärjestelmänsä, (Lastovetska, 2021).

Ohjelmointirajapintojen merkitys on kasvanut entisestään nykyaikaisessa ohjelmistokehityksessä. Ne eivät ole pelkästään teknisiä työkaluja, vaan niistä on tullut kriittisiä osia, jotka vaikuttavat siihen, miten sovellukset rakennetaan, toimivat ja ovat yhteydessä muihin digitaalisiin palveluihin. Yksi ohjelmointirajapintojen hyödyistä on yhteentoimivuuden ja integraation mahdollistaminen. Ne mahdollistavat saumattoman integroinnin ohjelmistojärjestelmien ja ulkoisten palvelujen välillä sekä helpottavat pääsyä eri tietolähteisiin (Nylas, n.d.). Lisäksi ohjelmointirajapinnat mahdollistavat myös alustarajojen ylittävän integraation, mikä tarkoittaa sitä, että ohjelmointirajapintojen ansiosta sovellukset voivat toimia eri laitteissa ja ympäristöissä (Nylas, n.d.). Ohjelmointirajapinnat myös helpottavat ohjelmiston jakamista moduuleihin tai komponentteihin, mikä helpottaa koodin uudelleenkäyttöä ja siten parantaa ohjelmiston ylläpidettävyyttä, kehittämistä ja testaamista (Nylas, n.d.).

Ohjelmointirajapintojen käyttö edistää myös modulaarista lähestymistapaa ohjelmistojen kehityksessä (Nylas, n.d.). Tämä mahdollistaa sovelluskehitysprosessin jakamisen pienempiin, hallittaviin osiin, joissa jokainen moduuli voidaan kehittää, testata ja ottaa käyttöön itsenäisesti. Ohjelmointirajapinnat toimivat näiden moduulien välisinä yhdyssiteinä. Luokittelemalla eri toiminnallisuudet erillisiin palveluihin, joihin pääsee käsiksi ohjelmointirajapintojen kautta, yksinkertaistetaan myös sovellusten päivittämistä ja ylläpitoa. Kehittäjät saavat näin mahdollisuuden päivittää tai korjata sovelluksen tiettyjä osia ilman riskiä, että se vaikuttaa koko järjestelmään. Tämä mahdollistaa myös uusien ominaisuuksien nopeamman käyttöönoton ja tehokkaammat virheenkorjaukset (Nylas, n.d.).

Myös uusien toimintojen ja ominaisuuksien lisääminen olemassa oleviin sovelluksiin ilman perustavanlaatuisia muutoksia mahdollistetaan ohjelmointirajapintojen avulla. Tällä tavoin

kehityksessä oleviin ohjelmistoihin voidaan helposti sisällyttää uusia toimintoja, joita kolmas osapuoli tarjoaa palveluna (Nylas, n.d.). Tämä edelleen parantaa käyttökokemusta tarjoamalla kehittyneempiä toimintoja ja integraatioita käyttäjille. Esimerkiksi älypuhelimen sovellukset hyödyntävät usein laitteiston rajapintoja (kuten kameraa, paikannusta jne.) monipuolisten toimintojen tarjoamiseksi. Ohjelmointirajapintojen avulla eri organisaatioiden kehittämät sovellukset ja palvelut voivat työskennellä yhdessä, jakaa tietoja ja tarjota integroituja ratkaisuja (Nylas, n.d.).

Digitaalisessa maailmassa on lukuisia ohjelmointirajapintoja, jotka on suunniteltu ohjelmistokehittäjien käyttöön. Esimerkiksi sosiaalisen median alustat, kuten Facebook, Twitter ja Instagram tarjoavat omia ohjelmointirajapintojaan. Karttoihin ja navigointiin liittyviä rajapintoja taas tarjoavat Google Maps ja Apple Maps. Sähköiseen kaupankäyntiin liittyvät rajapinnat ovat saatavilla esimerkiksi Amazonilta ja Ebayltä, kun taas Paypal tarjoaa rajapintoja verkko- ja mobiilimaksujen toteuttamiseen (Junaid Baig, 2023).

2.3 Ohjelmointirajapintatyypit ja arkkitehtuurit

Ohjelmointirajapintoja esiintyy monissa muodoissa ja niitä voidaan luokitella usein eri tavoin. Yleisimpiä ohjelmointirajapintojen luokittelutapoja ovat niiden julkaisukäytännöt, käyttötarkoitus sekä arkkitehtuuri.

Julkaisukäytäntöjen mukaan ohjelmointirajapinnat voidaan jakaa kolmeen eri luokkaan: yksityisiin, julkisiin ja yhteistyökumppanirajapintoihin. Yksityiset ohjelmointirajapinnat, jotka tunnetaan myös sisäisinä rajapintoina, on suunniteltu käytettäväksi tietyssä organisaatiossa tai yrityksessä vain sisäisesti. Julkiset ohjelmointirajapinnat ovat avoimia kaikille ja niitä voi käyttää kuka tahansa. Yhteistyökumppanirajapinnat puolestaan jaetaan vain valittujen liikekumppaneiden tai kolmannen osapuolen kehittäjien kesken (Nylas, n.d.).

Ohjelmointirajapintoja voidaan myös luokitella niiden käyttötarkoituksen perusteella. Yleisimmät ohjelmointirajapintojen käyttötavat ovat web-ohjelmointirajapinnat, tietokantaohjelmointirajapinnat, käyttöjärjestelmän ohjelmointirajapinnat, pilvipalveluiden ohjelmointirajapinnat ja kirjastopohjaiset ohjelmointirajapinnat (Nylas, n.d.).

Web-ohjelmointirajapinnat mahdollistavat HTTP/HTTPS-protokollia käyttävän verkkoviestinnän, jonka avulla verkkosovellukset voivat olla vuorovaikutuksessa palvelimien ja muiden palvelujen kanssa. Tyypillisesti web-ohjelmointirajapinnat pohjautuvat REST-, GraphQL- tai SOAP-arkkitehtuuriin (Nylas, n.d.). Tietokantaohjelmointirajapintojen avulla ohjelmistosovellukset voivat olla vuorovaikutuksessa tietokannan hallintajärjestelmien

kanssa, tarjoten keinon tietokantaan pääsemiseen ja siitä tiedon hakemiseen, tallentamiseen, päivittämiseen ja poistamiseen. Näin sovellukset voivat käyttää tietokantaa ja sen sisältöä tuntematta tietokannan toimintatapoja (Nylas, n.d.).

Käyttöjärjestelmän ohjelmointirajapintojen avulla sovellukset voivat olla vuorovaikutuksessa taustalla olevan käyttöjärjestelmän kanssa ja suorittaa siihen liittyviä toimintoja. Näiden rajapintojen avulla hallinnoidaan laitteistoresursseja, tiedostojärjestelmiä ja prosessien ohjausta, tarjoten keskeisiä palveluja, kuten muistinhallintaa ja laitteiden ohjausta (Nylas, n.d.).

Pilvipalvelualustojen tarjoamat ohjelmointirajapinnat mahdollistavat sovellusten saumattoman vuorovaikutuksen pilvipohjaisten resurssien ja palvelujen kanssa. Näitä tarjoavat esimerkiksi Amazon Web Services API ja Microsoft Azure API (Nylas, n.d.).

Kirjastopohjaiset ohjelmointirajapinnat ovat ohjelmistokirjastojen osia, jotka tarjoavat valmiiksi kirjoitettua koodia ja auttavat kehittäjiä suorittamaan yleisiä ohjelmointitehtäviä, mikä nopeuttaa kehitysprosessia. Nämä ohjelmointirajapinnat ovat kielikohtaisia, ja ne tarjoavat toimintoja perusapuohjelmista monimutkaisiin graafisen käyttöliittymän (GUI) komponentteihin. Esimerkkinä kirjastopohjaisesta ohjelmointirajapinnasta on Java API, mikä on olennainen osa Android-sovelluskehitystä (Nylas, n.d.).

Ohjelmointirajapinnat voidaan luokitella myös arkkitehtuurin mukaan. Seuraavaksi esitellään muutamia yleisimmin käytettyjä arkkitehtuureja.

2.3.1 REST API (Representational State Transfer)

REST API, joka tunnetaan myös nimellä RESTful API, on nykyään laajalti käytetyin ohjelmointirajapinta, joka perustuu REST-arkkitehtuuriin. REST toteutetaan HTTP- tai HTTPS-protokollien avulla, ja se hyödyntää HTTP-pyyntöjä tiedon käsittelyyn. Yleisimpiä HTTP-metodeja, joita REST-ohjelmointirajapinnassa käytetään, ovat GET, POST, PUT ja DELETE. Jokaisella resurssilla REST-ohjelmointirajapinnassa on yksilöllinen URI (Uniform Resource Identifier), jota käytetään resurssin tunnistamiseen ja käsittelemiseen. Tiedon palauttamiseen serveriltä REST käyttää yleisimmin JSON-, XML- tai CSV-tiedostomuotoja (Junaid Baig, 2023).

REST-rajapinnassa käytetään yleisesti kuutta arkkitehtuurista rajoitusta, joiden on täytyttävä, jotta rajapintaa voidaan kutsua todelliseksi REST APIksi. Nämä rajoitukset ovat seuraavat:

1. Asiakas-palvelin-malli (client-server model): Asiakas- ja palvelinpuolet ovat selkeästi erotettuja, toisistaan riippumattomia ja voivat kehittyä erikseen ilman keskinäistä riippuvuutta (Nylas, n.d.).

2. Tilattomuus (statelessness): Palvelin käsittelee jokaista pyyntöä uutena eikä koskaan tallenna muistiin mitään viimeisimmästä HTTP-pyynnöstä. Tämä tarkoittaa, ettei ole istuntoa eikä historiaa (Nylas, n.d.).

3. Välimuistitettavuus (cacheable): Välimuistiin tallentamisen mahdollisuudella parannetaan asiakaspuolen suorituskykyä ja vähennetään palvelimen kuormitusta (Nylas, n.d.).

4. Yhtenäinen käyttöliittymä (uniform interface): Resurssien tulisi käyttäytyä johdonmukaisesti koko ohjelmointirajapinnassa, niillä pitäisi olla yksi tunniste, jolla niitä voidaan edustaa, ja niiden pitäisi sisältää kaikki tarvittavat tiedot, jotta ne voidaan edustaa täydellisesti, mutta jättää tarpeettomat tiedot pois. Sen tulisi myös noudattaa selkeitä nimeämiskäytäntöjä sekä linkki- ja dataformaatteja (Nylas, n.d.).

5. Kerrostettu järjestelmä (layered system): APIt, tallennetut tiedot ja todennuspyynnöt on jaettava eri rajapintoihin. Tämän avulla voidaan parantaa sovelluksen turvallisuutta rajoittamalla kunkin kerroksen komponenttien vuorovaikutus vain välittömästi seuraavan kerroksen kanssa (Nylas, n.d.).

6. Koodi tilauksesta (code on demand): Tämä on valinnainen rajoitus, joka sallii asiakkaan ladata rajapinnasta koodia suoritettavaksi (Nylas, n.d.).

REST ohjelmointirajapinnat ovat erittäin suosittuja modernien web-sovellusten kehityksessä, koska ne tarjoavat selkeän ja joustavan tavan rakentaa ja integroida palveluita hajautettuihin järjestelmiin. Niiden käyttö mahdollistaa myös eri ohjelmointikielillä ja -alustoilla toteutettujen sovellusten välisen yhteensopivuuden. REST-rajapinnat on suunniteltu kevyiksi ja joustaviksi, mikä tekee niistä sopivia skaalautuvien ja helppohoitoisten verkkopalvelujen rakentamiseen (Junaid Baig, 2023).

Tämän opinnäytetyön käytännön osuudessa hyödynnetään REST-arkkitehtuurin pohjautuvia rajapintoja, ja REST-rajapinnan toimintaperiaatteita tarkastellaan tarkemmin myöhemmissä luvuissa.

2.3.2 SOAP API (Simple Object Access Protocol)

XML-pohjainen (Extensible Markup Language) SOAP-tiedonsiirtoprotokollaan perustuva SOAP API on ohjelmointirajapinta, jonka kautta viestintä tapahtuu XML-muotoisten viestien välityksellä. Nämä viestit sisältävät rakenteellista tietoa, kuten pyyntöjen parametreja ja vastausten tietoja.

SOAP APIssa noudatetaan tiukkoja sääntöjä ja viestintästandardeja, ja ne ovat rakenteellisempia verrattuna REST APIiin. Tämä tekee siitä samalla RESTiä turvallisemman protokollan. SOAP-ohjelmointirajapintoja on perinteisesti käytetty enemmän yritystason sovelluksissa, kuten suurissa yritysjärjestelmissä ja integraatioratkaisuissa.

SOAP APIn käyttö vaatii yleensä erityisten kirjastojen tai työkalujen käyttöä XML-viestien muodostamiseen ja käsittelyyn. Vaikka REST APIt ovatkin nykyään suositumpia kevyemmän rakenteensa ja nopeutensa vuoksi, SOAP APIa käytetään edelleen yrityksissä ja järjestelmissä, joissa vaaditaan muun muassa vankkaa tietoturvaa ja joissa järjestelmän päivittäminen uusiin teknologioihin voi olla kallista ja monimutkaista. (Junaid Baig, 2023).

2.3.3 GraphQL API

GraphQL-ohjelmointirajapinta käyttää avoimen lähdekoodin GraphQL-kyselykieltä tietojen kyselyyn ja manipulointiin. GraphQL:n kehitti alun perin Facebook vaihtoehtona REST- ja SOAP ohjelmointirajapinnoille (Nylas, n.d.).

GraphQL ohjelmointirajapinnan merkittävin ominaisuus on se, että se mahdollistaa asiakasohjelmien määrittää tarkasti, mitä tietoja palvelimelta halutaan hakea. Sen sijaan että palvelin määrittelisi datan rakenteen ja palauttaisi valmiiksi määritellyn datan, asiakasohjelma voi muotoilla kyselyn haluamallaan tavalla ja pyytää vain tarvitsemansa tiedot (Junaid Baig, 2023).

GraphQL ohjelmointirajapinta määrittelee yhden päätepisteen, johon asiakasohjelmat lähettävät POST-pyyntöjä, jotka sisältävät GraphQL-kyselyn. Kyselyssä määritellään, mitä tietoja halutaan hakea ja millä ehdoilla. Palvelin käsittelee kyselyn, suorittaa tarvittavat toimenpiteet ja palauttaa vastauksena vain pyydetyt tiedot. GraphQL mahdollistaa myös useiden resurssien yhdistämisen yhteen kyselyyn, mikä vähentää tarvetta useille peräkkäisille kyselyille ja parantaa tehokkuutta. Lisäksi GraphQL tarjoaa vahvan tyypityksen, joka auttaa kehittäjiä ymmärtämään kyselyitä ja vastauksia paremmin (Junaid Baig, 2023).

GraphQL on ihanteellinen silloin, kun tarvitaan optimoitua, asiakkaan tarpeiden mukaan räätälöityä tiedonhakua. Se sopii skenaarioihin, joissa asiakkaat vaativat joustavuutta tiedonhaussa, kuten mobiilisovelluksissa tai monimutkaisten tietorakenteiden käsittelyssä (Junaid Baig, 2023).

2.3.4 RPC API (Remote Procedure Call)

RPC API on yksinkertainen ohjelmointirajapintaprotokolla, joka on suunniteltu toimenpiteiden suorittamiseen toisessa järjestelmässä. Toisin kuin esimerkiksi REST APIssa, jossa vaihdetaan keskenään tietoja ja resursseja, RPC APIt kutsuvat suoritettavia toimintoja tai prosesseja. Näin ollen RPC-protokolla palauttaa menetelmän sen sijaan, että palautettaisiin resurssi. RPC-tekniikka mahdollistaa kommunikoinnin eri ohjelmistokomponenttien välillä verkon yli, riippumatta käytetyistä ohjelmointikielistä (Junaid Baig, 2023).

Kutsu lähetetään verkossa ja vastaus palaa takaisin. Tämä tarkoittaa käytännössä sitä, että etäkutsuja voidaan käyttää kommunikoimaan eri järjestelmien ja komponenttien välillä, jotka voivat sijaita fyysisesti eri paikoissa. RPC APIa käytetään usein hajautettujen järjestelmien, kuten asiakas-palvelinsovellusten, väliseen kommunikointiin. Tyypillisiä käyttötapauksia ovat esimerkiksi tietokantapalvelujen kutsut tai muut etäpalvelimilla olevat toiminnot, jotka tarjoavat tiettyjä palveluita verkon yli (Junaid Baig, 2023).

RPC:stä on olemassa erilaisia toteutuksia, kuten gRPC, XML-RPC ja JSON-RPC. Näitä protokollia käytetään määrittelemään etäkutsujen muoto, parametrit ja vastaukset. Jokaisella näistä protokollista on omat menetelmänsä etämenettelykutsujen käsittelyyn (Junaid Baig, 2023).

RPC APIt ovat edelleen käytössä monissa hajautetuissa järjestelmissä, erityisesti tilanteissa, joissa suorituskyky ja tehokkuus ovat tärkeitä tekijöitä. Ne ovat hyödyllisiä, kun kehittäjien on kutsuttava toimintoja tai proseduureja etäpalvelimilla (Junaid Baig, 2023).

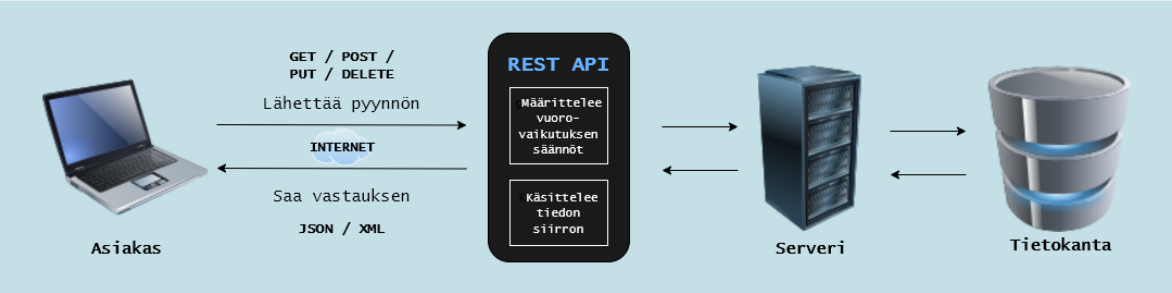
2.4 Ohjelmointirajapintojen toiminta

Seuraavaksi syvennytään tarkemmin REST-arkkitehtuurin toimintaperiaatteisiin, joka on yksi yleisimmin käytetyistä ohjelmointirajapinta-arkkitehtuureista.

REST-ohjelmointirajapintojen toiminta perustuu HTTP-viestintään, jossa ohjelmointirajapinta vastaanottaa HTTP-pyynnön ja palauttaa HTTP-vastauksen. Kuvassa 1 on esitetty REST-ohjelmointirajapinnan käytön vaiheet. Ensimmäiseksi asiakas lähettää pyynnön API-

palvelimelle käyttäen HTTP-protokollaa. Tämä pyyntö sisältää tietoja operaatiosta, jonka asiakas haluaa suorittaa, kuten tietojen hakemisesta. API-palvelin vastaanottaa ja käsittelee pyynnön. Se voi vahvistaa pyynnön, todentaa asiakkaan, valtuuttaa pyynnön tai suorittaa muita tarpeellisia toimintoja. Sen jälkeen API-palvelin lähettää asiakkaalle vastauksen, joka voi sisältää dataa, virhesanoman tai tilakoodin, joka ilmaisee toimenpiteen tuloksen. Lopuksi asiakas vastaanottaa ja käsittelee vastauksen (Nylas, n.d.).

Kuva 1 Esimerkki REST-rajapinnan arkkitehtuurista



REST ohjelmointirajapinnat käyttävät viestintään HTTP-protokollaa. Tämä protokolla kehitettiin 1990-luvun alussa erityisesti verkkoselaimien ja verkkopalvelimien väliseen viestintään ja tiedonsiirtoon, mutta sitä voidaan käyttää myös muihin tarkoituksiin. HTTP-protokollan avulla haetaan resursseja, kuten HTML-dokumentteja. HTTP on asiakas-palvelinprotokolla, mikä tarkoittaa, että pyynnön käynnistää asiakas, yleensä web-selain. Asiakkaat ja palvelimet kommunikoivat vaihtamalla yksittäisiä viestejä, joita kutsutaan pyynnöiksi (asiakkaalta) ja vastauksiksi (palvelimelta) (MDN Web Docs, 2023a).

2.4.1 HTTP-pyyntö

API-kutsu tai -pyyntö on pyyntö, joka tehdään ohjelmointirajapinnalle tietojen tai toimintojen käyttämiseksi. Asiakas luo API-kutsun ja lähettää pyynnön API-palvelimelle. REST-arkkitehtuurimalli noudattaa HTTP-protokollaa ja käyttää tätä protokollaa kutsujen lähettämiseen. API-pyyntö koostuu neljästä osasta: menetelmästä eli metodista (method), resurssin tunnisteesta (URI), otsikkotiedoista (header) ja viestirungosta (body). Nämä HTTP-menetelmät muodostavat API-vuorovaikutuksen ytimen, jonka avulla asiakas voi suorittaa erilaisia toimintoja, kuten tietojen hakua, uusien tietueiden luomista, olemassa olevien tietojen päivittämistä ja tietojen poistamista (Nylas, n.d.).

REST käyttää vakiomuotoisia HTTP-menetelmiä resursseille suoritettaviin toimintoihin. Käytetyimpiä menetelmiä ovat GET, jolla haetaan resursseja palvelimelta, POST, jolla

luodaan uusia resursseja, PUT ja PATCH, joilla päivitetään olemassa olevia resursseja, sekä DELETE, jolla poistetaan resursseja palvelimelta (Nylas, n.d.).

Asiakassovellus lähettää HTTP-pyynnöt päätepisteisiin (endpoint), jotka toimivat ”ovina” tai ”polkuina” API-palvelimelle. API-päätepisteet tunnistetaan URI:llä (Uniform Resource Identifier). Päätepisteet voivat myös hyväksyä kyselyparametreja, jotka muokkaavat pyyntöä lisätoimintojen tukemiseksi (Nylas, n.d.).

Otsikkotiedot sisältävät keskeisiä metatietoja API-pyyntöjen käsittelyyn. Näihin kuuluvat tiedot pyynnön ja vastauksen rungosta, valtuutuksesta, sisältötyypistä, todennuskoodeista ja välimuistin hallinnasta (Nylas, n.d.). Mahdollisia otsikkotietoja on runsaasti, mutta yleisimmin käytettyjä ovat:

- Accept: Ilmoittaa API-palvelimelle, missä muodossa odotat saavasi vastauksen, kuten JSON tai XML.

- Authorization: Sisältää todennustiedot, esimerkiksi API-avaimen tai käyttäjän tunnistetiedot.

- Content-type: Määrittää palvelimelle pyynnön rungossa lähetettävien tietojen tyypin.

- Cache-control. Ilmoittaa, pitäisikö vastaus tallentaa välimuistiin ja miten sitä hallitaan.

- User-Agent. Tunnistaa pyynnön esittävän asiakkaan (Ana, 2024a).

API-avain, joka sisällytetään otsikkotietoihin, on yksilöllinen tunniste, jota käytetään API-pyyntöjen todentamiseen. Ne mahdollistavat käytön seurannan, valvonnan ja rajoittamisen käyttöehtojen mukaisesti. Ne ovat yleensä palveluntarjoajan toimittamia (Nylas, n.d.). Opinnäytetyön käytännön osuudessa haetaan Googlelta API-avain Googlen ohjelmointirajapintojen pyyntöjen todentamista varten.

Kuvassa 2 on esitelty GET HTTP-pyyntö ja sen sisältämät otsikkotiedot. Pyyntö käyttää GET-metodia resurssin hakemiseen polusta ’/home.html’ ja hyödyntää HTTP/1.1 -versiota. Host-otsikkotieto määrittää isännän, jolta resurssi pyydetään. User-agent -otsikkotieto kertoo pyynnön lähettävän asiakasohjelman tiedot; tässä tapauksessa kyseessä on Mac OS X -käyttöjärjestelmässä toimiva Mozilla Firefox -selain. Accept -otsikkotieto ilmoittaa palvelimelle, minkä tyyppisiä vastauksia asiakas hyväksyy. Accept-Language puolestaan määrittää asiakkaan hyväksymät kielet, ja Accept-Encoding ilmoittaa minkälaisia

pakkausmenetelmiä asiakas hyväksyy vastauksessa. Referer -otsikkotieto paljastaa, mistä sivusta pyyntö on lähtöisin. Connection -otsikkotieto ilmaisee, että asiakas haluaa pitää yhteyden auki useampia pyyntöjä varten. Upgrade-Insecure-Requests -otsikkotieto kertoo palvelimelle, että asiakas haluaa päivittää epävarmat (HTTP) pyynnöt varmoiksi (HTTPS), jos mahdollista. If-Modified-Since -otsikkotieto pyytää palvelinta palauttamaan resurssin vain, jos sitä on muokattu annetun päivämäärän jälkeen, ja If-None-Match -otsikkotieto pyytää palautusta vain, jos resurssin ETag ei vastaa pyynnössä annettua arvoa. Lopuksi Cache-Control -otsikkotieto hallinnoi välimuistia; tässä tapauksessa max-age=0 tarkoittaa, että asiakas haluaa mahdollisimman tuoreen version resurssista, eikä halua käyttää välimuistia. Tässä esimerkissä käsiteltiin vain muutamia otsikkotietoja, vaikka niitä onkin lukuisia.

Kuva 2 Esimerkki verkko-ohjelmointirajapintaan lähetettävästä pyynnöstä (MDN Web Docs, 2023c)

```
GET /home.html HTTP/1.1
Host: developer.mozilla.org
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://developer.mozilla.org/testpage.html
Connection: keep-alive
Upgrade-Insecure-Requests: 1
If-Modified-Since: Mon, 18 Jul 2016 02:36:04 GMT
If-None-Match: "c561c68d0ba92bbeb8b0fff2a9199f722e3a621a"
Cache-Control: max-age=0
```

Kun ohjelmointirajapinnan pyyntömenetelmällä luodaan tai päivitetään resursseja palvelimella (esim. POST- tai DELETE-metodeilla), pyyntöön voidaan sisällyttää ylimääräinen elementti, jota kutsutaan viestirungoksi (body). Tätä kutsutaan myös nimellä payload. Viestirunko voi sisältää esimerkiksi uuden käyttäjätilin luomiseen tarvittavat tiedot, kuten käyttäjänimen, sähköpostiosoitteen ja salasanan. Viestirungot voivat olla eri rakenteisia, mutta yleisimmin ne ovat raakamuodossa kuten XML- tai JSON-muodossa tai binäärisessä muodossa esimerkiksi kuvien, videoiden ja äänen muodossa (Ana, 2024b).

2.4.2 HTTP-vastaus

Kun pyyntö on tehty, palvelin lähettää tämän käsiteltyään HTTP-vastauksen. API-vastaus muodostuu HTTP-tilakoodista, otsikkotiedoista ja tekstimuotoisesta vastauksesta, mikäli

pyyntö on onnistunut. HTTP-vastauksen tavoitteena on antaa käyttäjälle tietoa siitä, miten rajapinta on käsitellyt vastaanottamansa pyynnön.

HTTP-tilakoodi kertoo käyttäjälle, miten pyyntöä on käsitelty. Se ilmoittaa käyttäjälle, onko haku onnistunut vai onko ilmennyt virheitä, ja auttaa tunnistamaan mahdollisia ongelmatilanteita. Tilakoodit ovat kolminumeroisia ja ne jaetaan viiteen luokkaan ensimmäisen numeron perusteella. Nämä luokat käsittävät informatiiviset tilat (1xx), onnistuneet toiminnot (2xx), uudelleenohjaukset (3xx), asiakasvirheet (4xx) ja palvelinvirheet (5xx) (Restapitutorial.com, n.d.). On tärkeää, että palvelin lähettää tilakoodin, joka mahdollisimman tarkasti kuvaa ilmenneen ongelman. Yleisimmin käytetyt tilakoodit ohjelmointirajapinnoissa on esitetty taulukossa 1.

Taulukko 1 HTTP-vastauksen yleisimmät tilakoodit (Nylas, n.d.).

Tilakoodi	**Selitys**
200 (OK)	Vakiovastaus onnistuneista pyynnöistä.
201 (CREATED)	Vakiovastaus pyynnöstä, jonka tuloksena kohde on luotu onnistuneesti.
204 (NO CONTENT)	Vakiovastaus onnistuneesta pyynnöstä, joiden vastausrungossa ei palauteta mitään.
400 (BAD REQUEST)	Asiakkaan pyyntö on epäonnistunut pyynnön huonon syntaksin tai muun asiakkaan virheen vuoksi.
403 (FORBIDDEN)	Käyttäjällä ei ole oikeuksia käyttää tätä resurssia.
404 (NOT FOUND)	Resurssia ei löydy tällä hetkellä.
500 (INTERNAL SERVER ERROR)	Yleinen vastaus odottamattomaan virheeseen, jos tarkempia tietoja ei ole saatavilla.

Samoin kuin HTTP-pyyntö, myös HTTP-vastaus sisältää otsikkotietoja, jotka tarjoavat metatietoja vastauksesta ja sen ominaisuuksista. Nämä otsikot voivat olla yleisiä koko

viestiin liittyviä tietoja, vastauksen sisällön tyyppiä koskevia tietoja tai muista vastaukseen liittyviä tietoja (MDN Web Docs, 2023b).

Kuvassa 3 on esimerkkejä GET-pyynnön vastaukseen liittyvistä otsikkotiedoista. 200 OK osoittaa, että pyyntö onnistui ja palvelin palautti pyydetyn resurssin. Acces-Control-Allow-Origin sallii pääsyn resurssiin mistä tahansa alkuperästä (domainista), tässä tapauksessa kaikilta. Connection -otsikko ilmoittaa, että yhteys pysyy auki useampia pyyntöjä varten saman yhteyden aikana. Content-Encoding ilmoittaa vastauksen pakkausmenetelmän, Content-Type vastauksen sisällön tyypin ja merkkikoodauksen ja Date kertoo ajan, jolloin vastaus lähetettiin palvelimelta. Etag antaa resurssille yksilöllisen tunnisteen, mikä voi auttaa välimuistinhallinnassa ja muutosten havaitsemisessa. Keep-Alive määrittää yhteyden asetukset. Last-Modified kertoo ajan, jolloin resurssia viimeksi muokattiin, Server ilmoittaa palvelimen, joka vastasi pyyntöön, ja Set-Cookie - asettaa evästeen asiakaspuolelle. Transfer-Encoding ilmoittaa, miten data on jaettu vastauksessa. Vary osoittaa, että tiettyjen pyynnön otsikkotietojen perusteella vastauksen välimuistointi vaihtelee. X-Backend-Server paljastaa taustapalvelimen nimen, X-Cache-Info tarjoaa tietoa siitä, miten pyyntöä käsiteltiin välimuistissa, X-kuma-revision kertoo palvelimen käyttämän dokumentin version ja x-frame-options määrittää, onko sivun sallittua näyttää sisältöä ’<iframe>’-elementin sisällä, ja missä olosuhteissa.

Kuva 3 Esimerkki HTTP-vastauksen otsikkotiedoista (MDN Web Docs, 2023b).

```
200 OK
Access-Control-Allow-Origin: *
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Mon, 18 Jul 2016 16:06:00 GMT
Etag: "c561c68d0ba92bbeb8b0f612a9199f722e3a621a"
Keep-Alive: timeout=5, max=997
Last-Modified: Mon, 18 Jul 2016 02:36:04 GMT
Server: Apache
Set-Cookie: mykey=myvalue; expires=Mon, 17-Jul-2017 16:06:00 GMT; Max-Age=31449600; Path=/;
secure
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
X-Backend-Server: developer2.webapp.scl3.mozilla.com
X-Cache-Info: not cacheable; meta data too large
X-kuma-revision: 1085259
x-frame-options: DENY
```

Vastauksen sisällössä palautetaan asiakkaan pyytämät tiedot tai resurssit. Sisällön tyyppi määritellään ohjelmointirajapinnassa. Vastauksen sisältö voi olla esimerkiksi JSON-muodossa tai binäärisessä muodossa, joka sisältää kuvia, ääntä tai muita tiedostoja (Nylas, n.d.). Vastauksen otsikkotiedoissa Content-Type kertoo, missä muodossa vastaus palautetaan.

2.5 REST-ohjelmointirajapinnan käyttö (API-integraatio)

API-integraatio on prosessi, jossa eri ohjelmistosovellukset liitetään toisiinsa ohjelmointirajapintojen välityksellä, jotta ne voivat toimia saumattomasti yhdessä. Integrointi ohjelmointirajapinnan kanssa on monivaiheinen prosessi, joka sisältää perusteellisen ymmärryksen ohjelmointirajapinnasta, huolellisen suunnittelun ja harkitun toteutuksen sovelluksen arkkitehtuurissa. Onnistunut integraatio vaatii tehokasta virheenkäsittelyä, turvallisuuteen liittyvien näkökohtien huomioimista ja perusteellista testausta (Nylas, n.d.).

Ensimmäinen vaihe ohjelmointirajapinnan käyttöönotossa on tunnistaa sopiva ohjelmointirajapinta, joka parhaiten vastaa haluttuja palveluja ja toimintoja. Tähän sisältyy sovelluksen erityistarpeiden ymmärtäminen ja vaatimusten arviointi eli sen pohdinta, mitä toimintoja sovellukseen halutaan lisätä ja millaisia tietoja on käsiteltävä. Lisäksi on arvioitava ohjelmointirajapinnan dokumentaation laatu, käyttäjäyhteisön tarjoama tuki sekä millaiset käyttöehdot, rajoitukset ja kustannukset ohjelmointirajapinnalla on (Nylas, n.d.).

Seuraava vaihe on perehtyä valitun ohjelmointirajapinnan dokumentaatioon ja käyttöehtoihin. Hyvin laadittu dokumentaatio sisältää tietoa ohjelmointirajapinnan ominaisuuksista, pyyntöjen ja vastausten rakenteesta sekä mahdollisista todennus- tai valtuutusvaatimuksista. On myös tärkeää tutustua päätepisteisiin, joiden kautta voidaan pyytää tietoja tai suorittaa toimintoja, vaadittuihin ja valinnaisiin parametreihin sekä pyyntömuotoon, jota ohjelmointirajapinta odottaa ja vastausten rakenteisiin, mukaan lukien kuinka tiedot on järjestetty ja mitä tilakoodit tarkoittavat (Nylas, n.d.).

Ennen ohjelmointirajapinnan käyttöä on tärkeää tutustua sen käyttöehtoihin. Nämä ehdot määrittelevät ohjelmointirajapinnan mahdolliset rajoitukset sekä sen, miten tarjottuja tietoja voidaan hyödyntää ja jakaa. Samalla on hyvä selvittää, onko ohjelmointirajapinnan käyttö ilmaista vai maksullista. Useat APIt vaativat myös rekisteröitymisen (Nylas, n.d.).

Seuraavaksi hankitaan API-tunnukset ja API-avain, joka on yksilöllinen tunniste API-pyyntöjen todentamiseen. Monet ohjelmointirajapinnat vaativat todennusta turvallisuuden ja pääsynvalvonnan vuoksi, mikä edellyttää yleensä tilin rekisteröintiä ohjelmointirajapinnan

toimittajan verkkosivustolla sekä API-avainten, tunnisteiden ja asiakastunnusten hankkimista (Nylas, n.d.).

API-integraation seuraava vaihe on kehitysympäristön määrittäminen ja pyynnön rakentaminen. Tässä vaiheessa asennetaan tarvittavat työkalut ja kirjastot, jotka tukevat API-integraatiota kehitysympäristöön. Näitä voivat olla esimerkiksi HTTP-asiakaskirjastot ja todennuskirjastot. Kehittäjän on myös määriteltävä, missä vaiheessa API-kutsut integroidaan sovelluksen työkulkuun. Sen jälkeen voidaan aloittaa API-kutsujen tekeminen. On suositeltavaa aloittaa yksinkertaisella pyynnöllä yhteyden testaamiseksi ja pyynnön sekä vastauksen perusmekanismin ymmärtämiseksi. Ohjelmointirajapinnan palvelin käsittelee tämän pyynnön ja lähettää vastauksen, jonka analysointi auttaa ymmärtämään, mitä tietoja ohjelmointirajapinta tuottaa (Nylas, n.d.).

Seuraavaksi on käsiteltävä API-vastaus ja virheet. Onnistuneet vastaukset käsitellään, ja tiedot sekä tulokset integroidaan ja hyödynnetään sovelluksessa. Virheiden käsittelyä varten API-integraatiolle kirjoitetaan yksikkötestit, joilla varmistetaan, että jokainen integraation osa toimii odotetusti. Virheenkäsittely toteutetaan tapauksissa, joissa API-pyyntö epäonnistuu tai palauttaa virhevastauksen, jotta voidaan joustavasti reagoida ongelmiin, kuten verkko-ongelmiin tai virheelliseen API-avaimeen. Eri HTTP-tilakoodeja varten toteutetaan vankka virheenkäsittely, koska ohjelmointirajapinta saattaa palauttaa useita erilaisia koodeja. Yleiset virheet on hyvä ymmärtää ja käsitellä sujuvasti sovelluksessa. Lopuksi koko API-integraation toiminta testataan, jotta voidaan varmistaa sovelluksen toimivan odotetulla tavalla todellisessa skenaariossa (Nylas, n.d.).

3 Google Maps -karttapalvelu

Google on yhdysvaltalainen vuonna 1998 perustettu yritys, ja se tarjoaa verkkopalveluita, erityisesti keskittyen hakupalveluihin. Googlen kehittämä nopea, tarkka ja helppokäyttöinen hakukone on yksi suosituimmista ja tunnetuimmista hakukoneista (Britannica, 2024). Yritys palvelee myös yritysasiakkaita, kuten mainostajia, sisällöntuottajia ja sivustojen ylläpitäjiä, tarjoten tehokkaita mainosmahdollisuuksia sekä erilaisia tuottoja tuottavia hakupalveluita.

Google Maps on Googlen toteuttama, internetissä toimiva digitaalinen karttapalvelu, joka tarjoaa tarkkoja karttoja, satelliittikuvia, katunäkymiä ja reaaliaikaisia liikenneolosuhteita eri puolilla maailmaa. Lisäksi se mahdollistaa paikallisten yritysten sijainti- ja yhteystietojen etsimisen sekä reittiohjeiden hakemisen (Javatpoint, n.d).

Google Mapsin ensimmäinen beetaversio julkaistiin 8. helmikuuta 2005, ja samana vuonna Google julkaisi myös Google Earth -palvelun, 3D-kartoituspalvelun, jonka avulla käyttäjät voivat tutkia maailmaa tietokoneen näytöllä. Google Earth suunniteltiin ensisijaisesti pöytäkoneille ja vaati käyttäjien lataavan ja asentavan ohjelmiston, kun taas Google Maps luotiin helpommin käytettäväksi ja käyttäjäystävällisemmäksi ja palvelu suunniteltiin käytettäväksi verkkoselaimen kautta (Javatpoint, n.d).

Google on vuosien varrella lisännyt lukuisia ominaisuuksia ja toimintoja Google Mapsiin tehden siitä entistä hyödyllisemmän ja tehokkaamman. Esimerkiksi vuonna 2007 esiteltiin Street View -ominaisuus, joka mahdollistaa panoraamakuvien katselun kaduista ja paikoista, ja vuonna 2013 lisättiin reaaliaikaisia liikennetietoja auttamaan käyttäjiä välttämään liikenneruuhkia ja löytämään nopeimman reitin määränpäähänsä (Wayback machine, n.d.). Nykyään Google Maps kattaa 220 maata ja aluetta ja on käytettävissä sekä verkkoselaimessa ja mobiilisovelluksissa iOS- ja Android-laitteille (Javatpoint, n.d).

Google Maps tarjoaa laajan valikoiman ominaisuuksia ja toimintoja, jotka on suunniteltu avustamaan käyttäjiä navigoimaan ja tutkimaan ympäröivää maailmaa. Taulukossa 2 on esitetty joitakin Google Mapsin keskeisimpiä ominaisuuksia.

Taulukko 2 Google Mapsin keskeisimmät ominaisuudet (Javatpoint, n.d).

Street View	Käyttäjät voivat tarkastella panoraamakuvia kaduista ja paikoista saadakseen paremman käsityksen paikasta ennen niille menoa.

Reittiohjeet	Google Maps tarjoaa käyttäjille reittiohjeet kahden tai useamman pisteen välillä, olipa kyse ajamisesta, kävelystä, pyöräilystä ja julkisesta liikenteestä.
Reaaliaikaiset liikennepäivitykset	Palvelu tarjoaa reaaliaikaisia liikennepäivityksiä, jotka auttavat käyttäjiä välttämään liikenneruuhkia ja löytämään nopeimman reitin määränpäähänsä.
Satelliittikuvat	Google Maps tarjoaa satelliittikuvia paikoista ympäri maailmaa. Käyttäjät voivat vaihtaa satelliitti- ja karttanäkymien välillä ymmärtääkseen paremmin tutkimaansa aluetta.
Sisäkartat	Google Maps tarjoaa sisäkarttoja tuhansista paikoista, kuten lentokentistä, ostoskeskuksista ja museoista. Käyttäjät voivat tarkastella rakennusten asettelua ja navigoida niiden sisällä löytääkseen tiettyjä paikkoja.
Paikalliset yritystiedot	Palvelu tarjoaa tietoa paikallisista yrityksistä, mukaan lukien osoitteet, puhelinnumerot ja muiden käyttäjien arvostelut.
Tutki	Google Mapsin ”Tutki” -ominaisuuden avulla käyttäjät voivat löytää uusia paikkoja ja nähtävyyksiä alueellaan. Käyttäjät voivat etsiä tiettyjä luokkia kuten ravintoloita, kahviloita tai puistoja ja tutustua Googlen algoritmin suosituksiin.
Offline-kartat	Käyttäjät voivat ladata karttoja ja käyttää niitä offline-tilassa, mikä tekee siitä hyödyllisen työkalun matkustajille, jotka eivät ole internetin ulottuvilla.
Mukautetut kartat	Käyttäjät voivat luoda mukautettuja karttoja lisäämällä merkkejä, viivoja ja muotoja. Tämä ominaisuus on hyödyllinen matkojen suunnittelussa tai kiinnostavien paikkojen merkitsemisessä.

3.1 Google Mapsin -ohjelmointirajapinnat

Google on kehittänyt useita ohjelmointirajapintoja, joiden avulla pääsy ja kommunikointi Googlen palveluiden kanssa mahdollistetaan, kuten Google Mapsin, Youtuben, Google Driven ja lukuisien muiden palveluiden kanssa. Näiden ohjelmointirajapintojen lisäksi Google tarjoaa sovelluskirjastoja, jotka helpottavat ohjelmistokehittäjien työtä ja vähentävät koodin määrää (Google, n.d.a).

Google Mapsin ohjelmointirajapinnat ovat osa Googlen laajaa ohjelmointirajapintakirjastoa, joka sisältää useita ohjelmointirajapintoja sekä SDK-paketteja (Software Development Kit) sovelluskehittäjien käyttöön. Tarjolla on useita eri tapoja yhdistää Google Mapsin karttoja ja palveluita verkkosivuihin ja mobiilisovelluksiin, hakea tietoa karttapalveluista ja muokata karttanäkymää sovellukseen sopivaksi. Kehittäjä saattaa tarvita yhtä tai useampaa ohjelmointirajapintaa tarpeidensa mukaan. Tällä hetkellä Google Maps tarjoaa 19 eri ohjelmointirajapintaa, jotka on jaettu neljään eri kategoriaan: kartat, reitit, paikat ja ympäristö (Google, 2024a).

Google Mapsin ohjelmointirajapintojen käyttö on periaatteessa ilmaista, mutta samalla rajoitettua. Rajapintojen käyttämiseksi tulee aktivoida laskutus Google API -konsolissa luottokorttitietojen avulla, joita Google käyttää henkilöllisyyden varmistamiseen. Google tarjoaa kuukausittain 200 dollarin edestä ilmaista käyttöä, ja tämän määrän alittuessa käytöstä ei veloiteta. Jos käyttö ylittää tämän määrän, maksu perustuu käytettyjen palveluiden määrään. Maps Embed API, joka mahdollistaa interaktiivisen kartan tai Street View -panoraamakuvan sijoittamisen verkkosivulle, on täysin ilmainen. Google mahdollistaa myös päivittäisten pyyntörajojen ja laskutettavien enimmäisrajojen asettamisen, jotta ohjelmointirajapintojen käyttöä voidaan hallita ja suurten laskujen muodostuminen voidaan estää (Google, n.d.b).

Googlen ohjelmointirajapintoja otettaessa käyttöön, kehittäjällä tulee olla yksilöllinen API-avain HTTP-pyyntöjen vahvistamiseksi. Tämä avain voidaan luoda Googlen API -konsolissa, ja sille voidaan asettaa erilaisia rajoituksia, kuten turvallisuusrajoituksia ja sovellusrajoituksia (Google, 2024a). Seuraavassa osiossa esitellään muutamia Google Mapsin eri ohjelmointirajapintoja.

3.2 Maps JavaScript API

Maps JavaScript API on yksi Google Mapsin käytetyimmistä ohjelmointirajapinnoista. Sen avulla voidaan integroida interaktiivisia karttoja verkkosivuille ja mobiililaitteisiin JavaScriptin

avulla. Kehittäjät voivat käyttää tätä rajapintaa lisätäkseen karttoja, piirtääkseen reittejä, lisätäkseen merkkejä ja suorittaakseen monia muita toimintoja. Maps JavaScript APIssa on neljä peruskarttatyyppiä (tiekartta, satelliitti, hybridi ja maasto), joita voidaan muokata käyttämällä kerroksia ja tyylejä, ohjaimia ja tapahtumia sekä käyttämällä erilaisia palveluja ja kirjastoja (Google, 2024h).

3.3 Geocoding API ja Geolocation API

Geocoding APIlla voidaan muuttaa osoitteet maantieteellisiksi koordinaateiksi, kuten leveys- ja pituuskoordinaateiksi. Näiden koordinaattien avulla voidaan sijoittaa merkkejä tai sijainteja kartalle. Haku voidaan suorittaa myös käänteisesti, eli koordinaatit voidaan muuttaa osoitteeksi (Google, 2024f).

Geolocation APIn avulla voidaan hakea leveys- ja pituuskoordinaatteja käyttämällä matkapuhelinlaitetietoja, matkapuhelinmastotietoja ja WiFi-yhteyspistekentän tietoja. Geolocation APIlla voidaan saada laitteen sijainti, vaikka siinä ei olisikaan natiivia geopaikannusta tai GPS:ää (Google, 2024g).

3.4 Maps Static API ja Maps Embed API

Maps Static APIn avulla Google Maps -kartta voidaan upottaa kuvana verkkosivulle tai sovellukseen ilman JavaScriptiä tai dynaamista sivulatausta. Maps Static API -palvelu luo kartan tavallisella HTTP-pyynnöllä lähetettyjen URL-parametrien perusteella, ja se palauttaa kartan kuvana, joka voidaan näyttää verkkosivulla tai sovelluksessa (Google, 2024b). Kuva palautetaan joko GIF-, PNG- tai JPEG-muodossa. Jokaiseen API-kutsuun voidaan määrittää kartan sijainti, kuvan koko, zoomaustaso, karttatyyppi ja valinnaisten merkintöjen sijainti kartalla. Sijainnin määrittämiseksi leveys- ja pituusasteet määritellään kuuden desimaalin tarkkuudella (Google, 2024c).

Maps Embed API toimii samalla tavalla kuin Maps Static API, mutta sen sijaan, että ohjelmointirajapinta palauttaisi kartan kuvamuodossa, interaktiivinen kartta voidaan upottaa verkkosivulle tai sovellukseen yksinkertaisen HTML-upotuksen avulla (Google, 2024d).

3.5 Places API

Places APIlla voidaan hakea tietoja paikoista HTTP-pyyntöjen avulla. API palauttaa muotoiltuja sijaintitietoja ja kuvia toimipaikoista, maantieteellisistä sijainneista tai

merkittävistä kiinnostavista kohteista. Places APIlla voidaan hakea esimerkiksi paikan perustietoja kuten nimi, osoite, sijaintikoordinaatit, aukioloajat ja puhelinnumero, valokuvia paikasta, arvosteluita ja arvosanoja, lähialueen paikkojen hakua ja käyttäjä voi rajata hakutuloksia esimerkiksi etäisyyden, suosion, arvostelun tai muiden kriteerien perusteella. Google Maps Platform tarjoaa erilliset versiot Places-kirjastosta Androidille, iOS:lle ja JavaScriptille (Google, 2024e).

3.6 Routes, Roads, Directions ja muita ohjelmointirajapintoja

Routes APIn avulla voidaan palauttaa joko ihanteellinen reitti kahden paikan välillä tai etäisyydet ja matka-ajat eri lähtö- ja määräpaikkojen välisten reittien matriisille. Ohjelmointirajapinnan avulla saadaan tarkat reitti- ja matkatiedot, joissa hyödynnetään liikennetietoja, ajantasaisia liikenne- ja tieolosuhteita sekä reittiasetuksia (Google, 2024i).

Roads API on palvelu, joka ottaa HTTP-pyyntönä yhden tai useamman karttapisteen leveys- ja pituuskoordinaatit ja etsii näiden pisteiden avulla läheiset tieosuudet ja palauttaa paikkatunnuksen sekä metatietoja, kuten lähimmän tieosuuden, arvioidun saapumisajan ja nopeusrajoitukset (Google, 2024j).

Directions API palauttaa ohjeet sijaintien välissä JSON- tai XML-muodossa. Sitä käyttämällä voidaan saada ohjeita eri liikennemuotoihin, kuten joukkoliikenteeseen, autoiluun, kävelyyn tai pyöräilyyn (Google, 2024k).

Google Mapsilla on myös kolme ympäristöön liittyvää APIa. Platform Solar API:n tarkoituksena on nopeuttaa aurinko- ja energiajärjestelmien asennuksia. Air Quality APIn avulla voidaan pyytää ilmanlaatutietoja tietyltä sijainnilta. Pollen APIn avulla taas voidaan pyytää siitepölytietoja halutusta sijainnista (Google, 2024l, ks. myös Google, 2024m ja Google, 2024n).

Google Mapsilla on tarjolla myös monia muita ohjelmointirajapintoja. Googlella on sivustollaan kattava dokumentaatio kaikista tarjoamistaan ohjelmointirajapinnoista. Lisäksi Google Maps tarjoaa SDK:ita eli ohjelmistokehityksen työkalupaketteja kehittäjien käyttöön. Maps SDK for iOSin avulla voidaan lisätä iOS-sovelluksiin Google Maps -tietoihin perustuvia karttoja. SDK huolehtii automaattisesti pääsystä Google Maps -palvelimille, karttojen näyttämisestä ja reagoinnista käyttäjän eleisiin, kuten napsautuksiin ja vetoihin. Maps SDK for Android on vastaava työkalupaketti Android -sovelluksille. Se tukee sekä Kotlin- että Java -ohjelmointikieliä ja se tarjoaa lisäkirjastoja ja laajennuksia lisäominaisuuksia ja ohjelmointitekniikoita varten (Google, 2024o, ks. myös Google, 2024p).

4 Opinnäytetyössä käytettävät työkalut ja tekniikat

Opinnäytetyössä ohjelmoitava mobiilisovellus toteutettiin React Nativella. Tähän kehitysalustaan päädyttiin, koska opinnäytetyön tekijällä oli jo aiempaa kokemusta sen käytöstä, ja lisäksi React Native on tunnettu helppokäyttöisyydestään sekä laajasta ohjeistuksestaan ja tuestaan. Ohjelmointiympäristöksi valittiin Visual Studio Code, sillä ympäristönä se oli jo vakiintunut työkaluna opinnäytetyön tekijälle. Lisäksi Visual Studio code on kevyt, nopea ja mukautettavissa ja se tukee laajasti eri ohjelmointikieliä ja kehyksiä, kuten opinnäytetyössä käytettäviä JavaScriptiä ja React Nativea. Sovelluksen tietokantaratkaisuna käytettiin SQLiteä. Kehitysvaiheessa mobiilisovellusta testattiin Android Studion emulaattorilla ja lopullinen versio testattiin fyysisillä älypuhelimilla.

4.1 Visual Studio Code

Visual Studio Code (VS Studio) on Microsoftin kehittämä ilmainen ja avoimeen lähdekoodiin perustuva ohjelmointiin käytettävä kehitysympäristö. Se on saatavilla Windowsille, macOS:lle ja Linuxille. Visual Studio Coden lukuisien laajennusten avulla voi kehitysympäristön virittää itselle tärkeiden kehitystekniikoiden mukaan. Laajennusten avulla saa tuen eri ohjelmointikielille kuten C++, C#, Go, Java ja Python (Visual Studio Code, 2024).

4.2 React Native

React Native on JavaScript-pohjainen avoimen lähdekoodin kehys, jonka Facebook kehitti vuonna 2015 natiivien mobiilisovellusten rakentamiseen. Tämä kehys perustuu React-kirjastoon, jota käytetään verkkosovellusten kehittämiseen. React Native käyttää JavaScriptiä hyödyntääkseen alustan ohjelmointirajapintoja ja React-komponentteja määritellessään käyttöliittymän ulkoasun ja käyttäytymisen (React Native, 2024). Tarjolla on runsaasti valmiita komponentteja, kirjastoja ja työkaluja, jotka helpottavat ja nopeuttavat sovelluskehitystä. React Nativella voidaan luoda sovelluksia sekä iOS-, että Android-alustalle käyttämällä samaa koodipohjaa (Netguru, n.d.). Sillä voidaan ohjelmoida JavaScriptillä ilman tarvetta oppia alustakohtaisia ohjelmointikieliä, kuten Javaa, Swiftiä tai Objective-C:ä.

Natiivilla mobiiliohjelmoinnilla tarkoitetaan mobiilisovellusten kehittämistä, suoraan tietylle mobiilialustalle käyttäen kyseisen alustan omia työkaluja ja ohjelmointikieliä. Tällaisen lähestymistavan etuja ovat muun muassa parempi suorituskyky, koska sovellus on optimoitu tietylle käyttöjärjestelmälle, parempi pääsy laitteen ominaisuuksiin ja käyttöjärjestelmän

tarjoamiin palveluihin sekä parempi käyttäjäkokemus, koska sovellus voi hyödyntää käyttöjärjestelmän omia käyttöliittymäkomponentteja ja navigointimalleja.

React Native ei ole perinteistä natiiviohjelmointia, mutta sen avulla voidaan luoda natiivilta tuntuvia mobiilisovelluksia ja sama koodi toimii eri käyttöjärjestelmillä. React Native -arkkitehtuurissa ”silta” on keskeinen osa järjestelmää, joka mahdollistaa JavaScriptin ja alkuperäisen, natiivin koodin välisen kommunikoinnin. Tämä järjestelmä käyttää React-kirjastoa sovelluksen renderöintiin laitteella. Renderöinnillä tarkoitetaan prosessia, jossa komponentti tai näkymä muunnetaan JavaScript-koodista visuaaliseksi käyttöliittymäksi mobiililaitteen näytölle. Yksinkertaistettuna silta kääntää JavaScript-koodin alustakohtaisiksi natiivikomponenteiksi, mahdollistaen sovelluksen natiivin renderöinnin. Tämä prosessi ei vaikuta käyttäjäkokemukseen, sillä asynkroniset kutsut tapahtuvat pääsäikeen ulkopuolella (Max Tsurbeliov, 2021).

React Native -sovellukset rakennetaan JavaScriptillä ja JSX:llä, joka on JavaScriptin merkintäkieli käyttöliittymäkomponenttien luomiseen ja määrittämiseen. React Native höydyntää natiivikomponentteja käyttöliittymän rakentamisessa, kuten <View>, <Text> ja <Image>. Tyylittelyssä käytetään JavaScript-pohjaista tyylittelyä, joka muistuttaa CSS:ää, mutta sisältää omat erityispiirteensä. Navigointi React Nativen sovelluksissa toteutetaan usein kirjastoilla kuten react-navigation, joka tarjoaa natiivimaisia navigointiratkaisuja, kuten pino- ja välilehtinavigoinnin.

React Nativesta on tullut nopeasti valtavan suositun sovelluskehittäjien keskuudessa. React Native -kehitystä käytetään joidenkin maailman johtavien mobiilisovellusten, kuten Instagramin, Facebookin ja Skypen kehittämiseen (Netguru, n.d.).

4.3 Android Studio

Android Studio on Googlen kehittämä kehitysympäristö (IDE), joka on suunniteltu erityisesti Android-sovellusten kehittämiseen. Se tarjoaa kehittäjille monipuoliset työkalut sovellusten suunnitteluun, koodaamiseen, testaamiseen ja julkaisemiseen Android -alustalle. Android Studio sisältää Android-kehitykseen tarvittavia työkaluja, kuten XML-editorin käyttöliittymäsuunnittelua varten, Java- ja Kotlin-koodieditorit, emulaattorin Android-sovellusten testaamiseen, virheenkorjaustyökalut ja integroidun versionhallinnan tuen (Alex Mullis, 2017). Koska opinnäytetyössä ohjelmoitava mobiilisovellus ohjelmoitiin JavaScript -pohjaisella React Nativella, ei Android Studiota käytetty ohjelmointiympäristönä, koska se tukee vain Java- ja Kotlin -ohjelmointikieliä. Mobiilisovelluksen ohjelmointivaiheessa käytettiin Android Studion emulaattoria testaamiseen ja kehittämiseen.

4.4 SQLite

SQLite on kevyt ja itsenäinen tietokantajärjestelmä, joka tarjoaa relaatiotietokannan ominaisuuksia ilman erillistä tietokantapalvelinta. Se on avoimen lähdekoodin tietokantajärjestelmä ja se on suunniteltu olemaan yksinkertainen, nopea ja helppo integroida erilaisiin sovelluksiin. SQLite on maailman laajimmin käytetty tietokanta. SQLitessä ei ole erillistä palvelinprosessia eli SQLite lukee ja kirjoittaa tietoa suoraan tavallisiin levytiedostoihin. SQLlite on sisäänrakennettu kaikkiin matkapuhelimiin ja useimpiin tietokoneisiin (SQLite, 2023). Opinnäytetyössä käytettiin SQLiteä tietokantaratkaisuna lenkin tietojen tallentamiseen.

4.5 Käytettävät Google ohjelmointirajapintapalvelut

Google Mapsin ohjelmointirajapintojen integrointi React Native -sovellukseen toteutettiin käyttämällä React Nativen React Native Maps -kirjastoa. Tämä kirjasto perustuu Google Mapsin Maps SDK for Android -työkalupakettiin, joka mahdollistaa karttojen näyttämisen sovelluksessa, sijaintitietojen hakemisen ja käyttämisen, reittien suunnittelun, karttatyylien mukauttamisen ja monia muita toimintoja. React Native Maps -kirjasto ei suoranaisesti lähetä pyyntöjä Google Mapsin ohjelmointirajapintaan, kuten selain tai palvelinsovellukset REST-pyyntöjen kautta tekisivät. Sen sijaan kirjasto hyödyntää laitteeseen asennettuja Google Maps -komponentteja ja API-avainta karttojen näyttämiseen ja kartan toimintojen käyttämäiseen.

Alun perin suunniteltiin, että käyttäjän sijainti haettaisiin Google Mapsin Geolocation APIlla, joka hyödyntää matkapuhelinlaitetietoja, matkapuhelinmastotietoja ja WiFi -yhteyspisteiden tietoja käyttäjän leveys- ja pituuskoordinaattien määrittämiseksi. Lopulta kuitenkin päädyttiin käyttämään React Nativen omaa Geolocation -ohjelmointirajapintaa sen helppokäyttöisyyden ja vaivattoman integroinnin vuoksi. Tämä ohjelmointirajapinta hakee samat sijaintiedot laitteen käyttöjärjestelmän tarjoamista sijaintipalveluista. Lisäksi React Nativen Geolocation -ohjelmointirajapinnan käyttö on maksuton, kun taas Googlen vastaavasta palvelusta voi koitua kustannuksia pyyntöjen ylittäessä sallitun määrän.

Koska ohjelmointirajapinnat sekä REST-pyyntöjen lähettäminen ja vastausten käsittely ovat keskeisiä aiheita opinnäytetyössä, käytännön osuudessa esitellään ensin, kuinka React Nativella voidaan lähettää API-pyyntö Google Mapsin Geolocation APIlle. Lisäksi tarkastellaan, mitä tietoja ohjelmointirajapinnasta saadaan ja kuinka näitä tietoja voidaan käsitellä.

5 Mobiilisovelluksen ohjelmointi

Opinnäytetyön käytännön osuudessa pyrittiin suunnittelemaan, ohjelmoimaan ja ottamaan käyttöön yksinkertainen hyvinvointisovelluksen prototyyppi. Sovelluksen tarkoituksena oli demonstroida, miten puhelimen sijaintiedot voidaan hakea ja integroida mobiilisovellukseen sekä kuinka Google Mapsin rajapintojen avulla voidaan hakea karttatietoja ja yhdistää sijaintitiedot haettuun karttaan. Teoreettisessa osuudessa tutkittiin, mikä Googlen tarjoamista rajapinnoista soveltui parhaiten sovelluksen tarpeisiin.

Sovelluksen ohjelmointiin idea tuli opinnäytetyön kirjoittajan lenkkeilyharrastuksesta sekä kiinnostuksesta sovelluskehitykseen ja -ohjelmointiin. Sovellus on suunniteltu ainoastaan opinnäytetyön kirjoittajan henkilökohtaiseen käyttöön ja koska kyseessä on sovelluksen prototyyppi, käyttäjäkokemukseen ja ulkoasun suunnitteluun keskityttiin vain pintapuolisesti. Vaikka React Nativella olisi voitu helposti toteuttaa sovelluksen ohjelmointi sekä Android-, että iOS-alustoille, tämä opinnäytetyö rajoittui toteuttamaan sovelluksen ainoastaan Android-alustalle.

Opinnäytetyön tekijällä oli jonkin verran aiempaa kokemusta ohjelmoinnista ja React Nativen käytöstä mobiiliohjelmoinnissa. Lisäksi hänellä oli hieman tuntemusta muista opinnäytetyössä käytettävistä työkaluista. Työn edetessä tuli kuitenkin jatkuvasti eteen uusia asioita, joita piti opetella ja joiden opiskeluun ja ymmärtämiseen kului paljon aikaa. Myös uusien ongelmien ratkaisemiseen kului runsaasti aikaa. Sovelluksen toteutus eteni vähitellen, sitä mukaa kun tekijän tiedot ja taidot karttuivat.

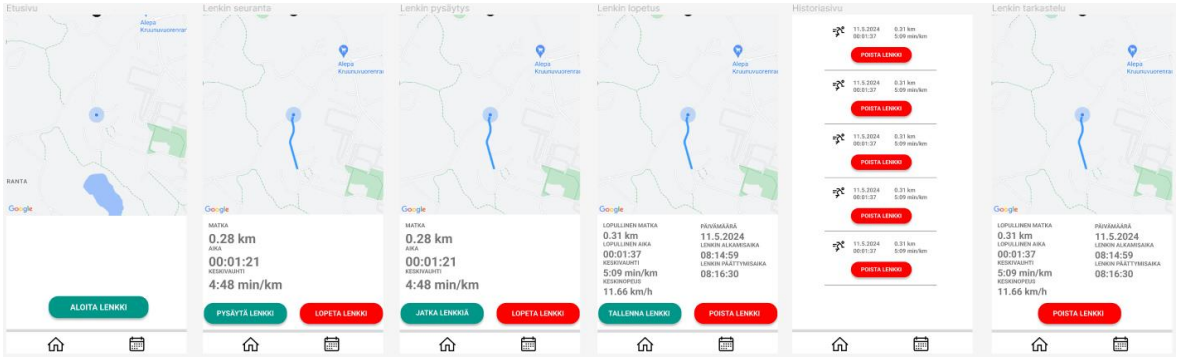
5.1 Sovelluksen idea ja suunnittelu

Opinnäytetyön tarkoituksena oli toteuttaa yksinkertainen lenkkeilysovellus, joka seuraa käyttäjän lenkkiä näyttäen hänen sijaintinsa Google Mapsin ohjelmointirajapinnasta haetulla kartalla ja samalla piirtäen karttaan käyttäjän kuljetun reitin. Sovellus hakee käyttäjän sijainnin joko Googlen ohjelmointirajapinnasta tai mobiililaitteen ja käyttöjärjestelmän antamista tiedoista. Lenkin aikana käyttäjälle näytetään matkan pituus kilometreinä, siihen kulunut aika ja keskinopeus (minuuttia/kilometri), ja nämä tiedot päivitetään säännöllisesti. Lenkin päätyttyä sovellus näyttää nämä tiedot sekä muita lenkkiin liittyviä tietoja. Kaikki tiedot sekä kartalla näkyvä käyttäjän juoksema reitti, tallennetaan puhelimen sisäiseen tietokantaan. Käyttäjä voi myöhemmin tarkastella aiempia lenkkejään, jotka esitetään omalla sivullaan järjestyksessä uusimmasta vanhimpaan.

Käyttöliittymäsuunnitelma laadittiin Figmalla, ja sen haluttiin olevan yksinkertainen, tehokas ja selkeä sekä soveltuvan hyvin lenkkeilyolosuhteisiin. Siksi lenkin aikana ruudulle tulevien tekstien ja käyttäjäpainikkeiden haluttiin olevan helposti luettavia ja riittävän suuria. Sovelluksen ensimmäisen version toteutus ja ohjelmointi aloitettiin ennen varsinaista käyttöliittymän suunnittelua, joten sovelluksen ulkoasu muotoutui tietynlaiseksi jo sovelluksen ohjelmointivaiheessa.

Sovelluksessa on kaksi pääsivua: ensimmäinen sisältää lenkin seurannan ja toinen näyttää käyttäjän lenkkihistorian. Navigaatiopalkki, joka sijaitsee ruudun alaosassa, mahdollistaa siirtymisen näiden kahden pääsivun välillä. Navigaatiopalkin kuvakkeet esittävät sivuja: talon kuva symboloi lenkin seurantaa ja kalenterin kuva lenkkihistoriaa. Lisäksi sovelluksessa on neljä muuta sivua, jotka avautuvat toimintopainikkeista. Ensimmäinen toimintosivu avautuu painamalla ”Aloita lenkki” -painiketta, jolloin lenkin seuranta käynnistyy. Lenkin seurannan ollessa käynnissä ”Pysäytä lenkki” -painiketta painettaessa, siirrytään toiselle toimintosivulle, jolloin lenkin seuranta pysähtyy. Painamalla ”Lopeta lenkki” -painiketta siirrytään kolmannelle toimintosivulle, missä lenkin seuranta päätetään, lenkin tiedot tulostuvat ruudulle ja ruudulle ilmestyvät käyttäjäpainikkeet, joista lenkin voi tallentaa tietokantaan tai poistaa muistista. Kun historiasivulta painetaan lenkkiä, jota halutaan tarkastella, avautuu neljäs toimintoikkuna, joka hakee tietokannasta tarkastelun kohteena olevan lenkin tiedot ja tulostaa ne ruudulle. Lenkin voi myös poistaa tietokannasta ”Poista lenkki” -painikkeella, jolloin näytölle ilmestyy ponnahdusikkuna, joka ilmoittaa toimenpiteen tuloksen. Jokainen Figmassa suunniteltu sivu on esitetty kuvassa 4.

Kuva 4 Figmassa suunnitellut sivut.



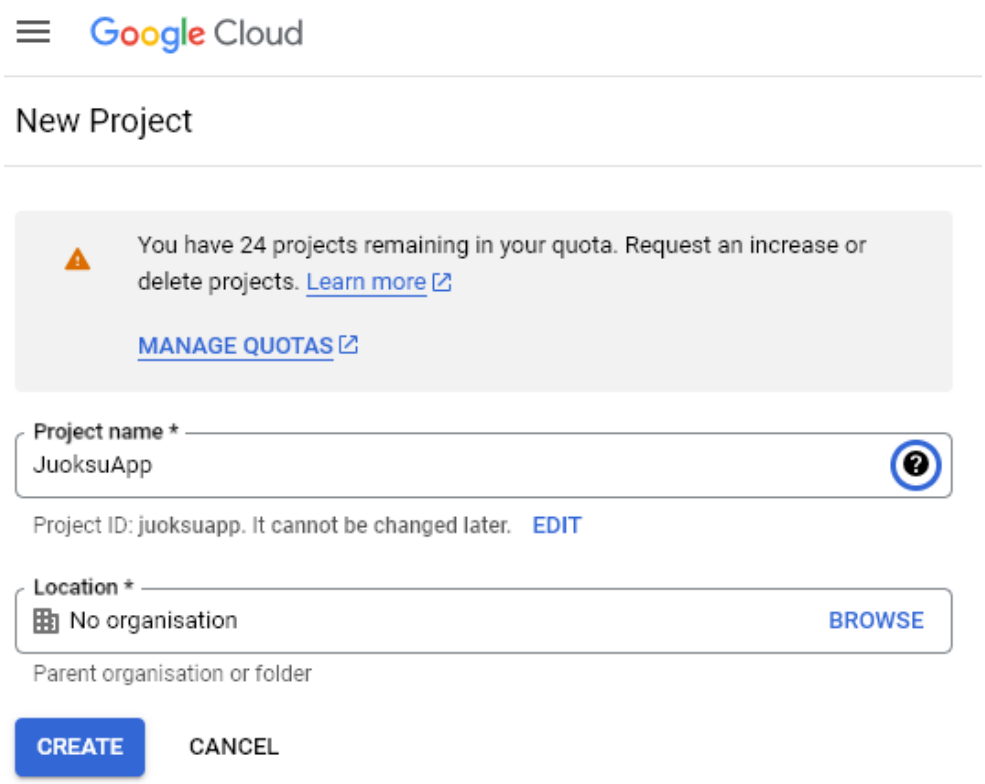
5.2 Google API-avaimen käyttöönotto

Googlen ohjelmointirajapintojen käyttöönottoa varten kehittäjän on hankittava yksilöllinen API-avain HTTP-pyyntöjen vahvistamiseksi. Tämän avaimen voi luoda Googlen Cloud API konsolissa. Avaimen luonti edellyttää kehittäjän laskutuksen aktivointia Google Cloud

konsolissa luottokorttitietojen avulla, joita Google käyttää myös henkilöllisyyden tarkistamiseen.

Avaimen luontia varten kehittäjän on ensin kirjauduttava Google Cloud Consoleen Google-tunnuksillaan. Ennen API-avaimen käyttöönottoa, täytyi ensiksi luoda uusi projekti sovellusta varten. Kuten kuvassa 5 näkyy, nimettiin projekti JuoksuApp:ksi.

Kuva 5 Uuden Google Cloud -projektin luonti.

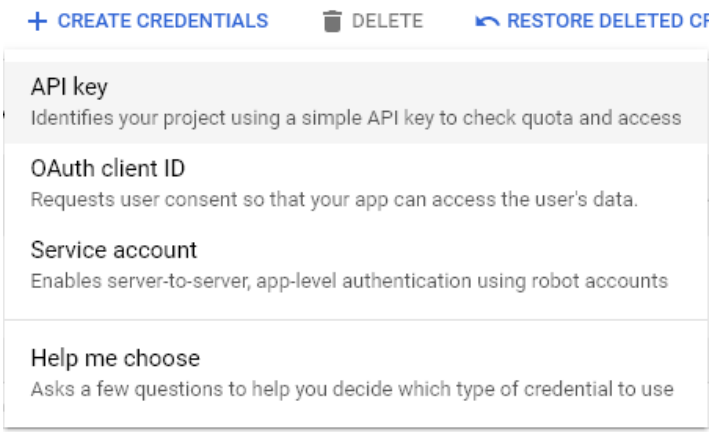


Tämän jälkeen tuli luoda uusi laskutustili. Tämä tapahtui valitsemalla ”Billing” -valikosta ”Manage billing accounts” ja sen jälkeen ”Create account”. Konsoli kysyy tilin luomista varten useita tietoja, kuten laskutustilin nimen, osoitteen ja luottokorttitiedot. Valitsemalla lopuksi ”Submit and enable billing”, luo konsoli laskutustilin sen jälkeen, kun olet vahvistanut henkilöllisyytesi verkkopankin kautta. Tämän jälkeen laskutustilille voidaan määritellä haluamia asetuksia kuten kuukausittainen laskutettava enimmäisraja tai muita rajoituksia. Tässä vaiheessa ei laskutukselle asetettu rajoituksia.

Seuraavaksi valittiin luotavalle API-avaimelle käyttöoikeudet ja luotiin itse API-avain. Tämä tehtiin valitsemalla ”APIs and services” -valikon alta ”Credentials”. Käyttöoikeuksilla voidaan määritellä millä tavoin yhdistäminen suoritetaan. Ohjelmoitavassa sovelluksessa riittää API-avain, joten valittiin tämä kuten kuvassa 6 näkyy. Seuraavaksi palvelu loi API-avaimen, mikä tallennettiin koneelle turvalliseen paikkaan. Halutessaan API-avaimelle pystyy luomaan

rajoituksia esimerkiksi, että avainta käytetään ainoastaan Android-sovelluksissa. Tässä vaiheessa ei API-avaimelle asetettu mitään rajoituksia.

Kuva 6 Käyttöoikeuksien valinta.



Lopuksi määriteltiin ohjelmoitavassa sovelluksessa käytettävät ohjelmointirajapinnat. Tämä onnistui ”APIs and services” -valikon alta kohdasta ”Enabled APIs and services”. Tästä kohdasta päästiin valitsemaan projektissa käytettävät ohjelmointirajapinnat. Opinnäytetyössä ohjelmoitavassa sovelluksessa käytettiin React Nativen Maps -kirjastoa, mikä käyttää Maps SDK for Android -työkalupakettia, joten tämä valittiin käyttöön. Konsoli kysyy muutamia kysymyksiä liittyen käytettävään APIin ja kehittämään sovellukseen. Käyttöön otettiin myös Geolocation APIn siltä varalta, että sitä käytettäisiin sijainnin hakemiseen.

Tämän jälkeen API-avain on valmis käytettäväksi sovelluksessa. API-avain piti vielä lisätä projektin AndroidManifest.xml-tiedostoon lisäämällä sinne Ohjelmakoodissa 1 näkyvät meta-data rivit. Tämä edellyttää tietenkin, että uusi projekti on jo luotu React Nativessa. AndroidManifest.xml-tiedosto sisältää keskeiset tiedot Android -sovelluksesta, ja siinä määritellään muun muassa sovelluksen rakenne, toiminnallisuudet ja käyttöoikeudet. Ohjelmakoodissa 1 ei näy tietosuojasyistä käyttämääni API-avainta.

Ohjelmakoodi 1 Projektin AndroidManifest.xml -tiedosto.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"   />
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
    <application
      android:name=".MainApplication"
      android:label="@string/app_name"
```

```
      android:icon="@mipmap/ic_launcher"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:allowBackup="false"
      android:theme="@style/AppTheme">
      <meta-data
        android:name="com.google.android.geo.API_KEY"
        android:value="Replace this with your API key"/>
      <activity
        android:name=".MainActivity"
        android:label="@string/app_name"
        android:configChanges="keyboard|keyboardHidden|orientation|scr
eenLayout|screenSize|smallestScreenSize|uiMode"
        android:launchMode="singleTask"
        android:windowSoftInputMode="adjustResize"
        android:exported="true">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER"
/>
        </intent-filter>
      </activity>
    </application>
</manifest>
```

5.3 React Native -projektin luonti

Sovelluksen kehittäminen alkoi uuden projektin käynnistämisellä. Ennen tätä oli kuitenkin asennettava tarvittavat sovellukset ja kehitysympäristöt, jotka opinnäytetyön tekijällä olivat jo asennettuna. Näihin kuuluu muun muassa Java JDK, joka tarjoaa työkalut ja resurssit Java-sovellusten kehittämiseen, Node.JS, joka on välttämätön JavaScript -koodin suorittamiseen, sekä aiemmin mainitut Visual Studio Code ja Android Studio. Sovelluksen kehitysvaiheen testausta varten asennettiin Android Studio ja siellä luotiin emulaattori. Lisäksi React Native CLI, komentorivityökalu, joka helpottaa uuden React Native -projektin luomista, hallintaa ja rakentamista asennettiin seuraavalla komennolla komentokehotteessa:

```
npm install -g react-native-cli
```

Tämän jälkeen luotiin uusi projekti komennolla:

```
npx react-native init runningApp
```

Tarvittavat kirjastot asennettiin komennoilla:

```
npm install react react-native
npm install react-native-maps –save
npm install @react-native-community/geolocation
npm install react-native-sqlite-storage –save
npm install @react-navigation/native
npm install react-native-screens react-native-safe-area-contex
npm install @react-navigation/bottom-tabs
npm install react-native-vector-icons
```

```
npm install @sayem314/react-native-keep-awake
```

Kun uusi projekti oli luotu ja kirjastot asennettu, voitiin siirtyä itse sovelluksen kehittämiseen. Asennetut kirjastot tuotiin koodiin käyttämällä ’import’ -lausuntoa. Ohjelmakoodissa 2 on esitetty, miten kaikki React Native -projektissa tarvittavat kirjastot ja komponentit on otettu käyttöön.

Ohjelmakoodi 2 Sovelluksen kirjastot ja komponentit.

```
import React, { useState, useEffect, useRef, useCallback } from
'react';
import { StyleSheet, Text, View, ScrollView, TouchableOpacity, Modal,
Image } from 'react-native';
import MapView, { PROVIDER_GOOGLE, Polyline, Marker } from 'react-
native-maps';
import Geolocation from '@react-native-community/geolocation';
import { openDatabase } from "react-native-sqlite-storage";
import { useFocusEffect, NavigationContainer } from '@react-
navigation/native';
import { createBottomTabNavigator } from '@react-navigation/bottom-
tabs';
import Icon from 'react-native-vector-icons/Ionicons';
import { activateKeepAwake, deactivateKeepAwake } from
"@sayem314/react-native-keep-awake";
```

5.4 API-kutsun lähettäminen Google Mapsin ohjelmointirajapintaan React Nativella

Kuten aiemmin mainittiin, päädyttiin sovelluksessa käyttämään React Native Maps -kirjastoa integroimaan Google Maps -karttoja React Native -sovellukseen. Opinnäytetyön suunnitteluvaiheessa mietittiin aluksi myös käyttäjän sijaintietojen hakemista Google Mapsin ohjelmointirajapinnasta Geolocation APIn avulla. Opinnäytetyön edetessä kuitenkin kävi ilmi, että tämä vaihtoehto ei ollut ihanteellinen, sillä se olisi saattanut olla hidas ja aiheuttaa tarpeettomia kuluja API-avaimen omistajalle. Lisäksi samoja sijaintitietoja voidaan hakea tehokkaammin ja nopeammin laitteen käyttöjärjestelmän tarjoamien sijaintipalveluiden kautta.

React Native mahdollistaa REST-pyyntöjen lähettämisen helposti JavaScriptin fetch -metodin avulla, joka käynnistää resurssien noutoprosessin palvelimelta. Ohjelmakoodissa 3 on esitetty esimerkki siitä, miten React Nativella voidaan lähettää API-pyyntö Google Mapsin Geolocation -ohjelmointirajapintaan.

Ohjelmakoodi 3 Geolocation API-pyyntö React Nativessa.

```
import React, { useEffect, useState } from 'react';
import { View, Text, StyleSheet } from 'react-native';
```

```
const App = () => {
  const [location, setLocation] = useState(null);

  useEffect(() => {
    const fetchLocation = async () => {
      try {
        const response = await fetch(
          'https://www.googleapis.com/geolocation/v1/geolocate?key=YOU
R_API_KEY,
          {
            method: 'POST',
            headers: {
              'Accept': 'application/json',
              'Content-Type': 'application/json',
            },
            body: JSON.stringify({
              considerIp: true,
            }),
          }
        );
        const status = response.status;
        const headers = response.headers;
        const data = await response.json();
        setLocation(data.location);

        console.log('Response Status:', status);
        console.log('Response Headers:', [...headers.entries()]);
        console.log('API Response Data:', data);

      } catch (error) {
        console.error(error);
      }
    };

    fetchLocation();
  }, []);

  return (
    <View>
      {location ? (
        <Text>
          Latitude: {location.lat}, Longitude: {location.lng}
        </Text>
      ) : (
        <Text>Haetaan koordinaatteja...</Text>
      )}
    </View>
  );
};

export default App;
```

Ohjelmakoodissa 3 esitetty API-pyyntö ohjataan ohjelmointirajapinnan dokumentaatiossa määriteltyyn päätepisteeseen ja siinä käytetään kyselyparametrinä pyyntöä lähettävän API-avainta. Tietosuojasyistä API-avainta ei näytetä tässä koodissa. Tämän jälkeen määritellään metodi, mikä käytettävässä ohjelmointirajapinnassa on poikkeuksellisesti POST, vaikka ohjelmointirajapinnasta haetaankin tietoja eikä luoda uusia resursseja. Pyynnön otsikkotiedot ja viestirunko määritellään seuraavaksi. Viestirungon pitää olla JSON-muodossa Googlen Geolocation -APIssa, ja siinä lähetettävät lisätiedot ovat valinnaisia. Tässä esimerkissä

lähetetään tieto ’considerIp: true’, joka tarkoittaa, että IP-osoitetta käytetään sijainnin arvioimisessa. Google Mapsin Geolocation -API pyynnön rungossa voidaan myös määrittää muita lisätietoja, kuten Wi-Fi-tukiasemien ja solutornien tiedot, jotka auttavat parantamaan sijainnin tarkkuutta.

Kun pyyntö on suoritettu onnistuneesti, API palauttaa vastauksen. Ohjelmakoodissa 3 esitetty pyyntö sai seuraavanlaisen vastauksen, joka tulostettiin konsoliin:

```
Response Status: 200
Response Headers: [["alt-svc", "h3=\":443\"; ma=2592000,h3-
29=\":443\"; ma=2592000"], ["cache-control", "private"], ["content-
type", "application/json; charset=UTF-8"], ["date", "Sun, 04 Aug 2024
12:58:55 GMT"], ["server", "scaffolding on HTTPServer2"], ["vary",
"Origin, X-Origin, Referer"], ["x-content-type-options", "nosniff"],
["x-frame-options", "SAMEORIGIN"], ["x-xss-protection", "0"]]
Response Data: {"accuracy": 36668.2245806135, "location": {"lat":
60.2374144, "lng": 25.0609664}}
```

Vastauksessa näkyy tilakoodi 200, joka osoittaa, että pyyntö on onnistunut ja palvelin on palauttanut pyydetyt tiedot ilman virheitä. Sen jälkeen vastauksessa annetaan otsikkotiedot, jotka sisältävät metatietoja vastauksen ominaisuuksista. Lopuksi palautetaan käyttäjän pyytämät tiedot, jotka ovat käyttäjän sijainnin tarkkuus metreinä ja sijainnin arvioidut leveys- ja pituuskoordinaatit asteina.

Edellä mainittu REST-pyyntö ja Google Mapsin Geolocation APIn käyttö eivät kuitenkaan olleet tarpeen opinnäytetyössä kehitetyn mobiilisovelluksen toteutuksessa.

5.5 Lenkin seurantasivun luonti

Ensimmäinen vaihe varsinaisen sovelluksen kehittämisessä oli toteuttaa perustoiminnot, jotka liittyivät käyttäjän sijaintiedon käsittelyyn ja kartan integrointiin. Tähän sisältyi käyttäjän laitteen sijaintikoordinaattien hakeminen, Google Mapsin kartan hakeminen ohjelmointirajapinnasta ja sen integroiminen sovellukseen, käyttäjän sijainnin merkitseminen kartalle ja kartan päivittäminen käyttäjän liikkuessa. Lisäksi toteutettiin toiminnallisuudet lenkin seurannan aloittamiseen, pysäyttämiseen ja lopettamiseen, käyttäjän kulkeman reitin piirtämiseen kartalle sekä lenkin seurantaan ja tilastointiin liittyvät toiminnot.

React Native -sovelluksissa voidaan käyttää koukkuja (hooks), jotka mahdollistavat tilan hallinnan funktionaalisissa komponenteissa ilman tarvetta käyttää luokkakomponentteja. Koukut tarjoavat tehokkaan tavan lisätä tilan hallintaa ja toiminnallisuutta funktionaalisiin komponentteihin. Tässä sovelluksessa hyödynnetään useState-, useRef-, useEffect ja useFocusEffect-koukkuja. Sovelluksen alkuvaiheessa määritellään tarvittavat tilat, joita

hallitaan useState-koukun avulla. useState-koukku on erityisesti hyödyllinen, kun käsitellään tietoja, jotka muuttuvat ajan myötä tai jotka ovat peräisin käyttäjän toiminnasta. Näille tilamuuttujille asetetaan alkuarvot ja luodaan asettajat, joilla tilaa voidaan hallita. Ohjelmakoodissa 4 esitellään sovelluksessa käytetyt koukut, ja esimerkiksi ensimmäinen useState-koukku, location, määrittelee käyttäjän tämänhetkisen sijainnin ja sitä voidaan hallita setLocation-asettajalla. Kun useState-koukun tilamuuttujan arvoa muutetaan sen tilametodin avulla, näyttö päivittyy automaattisesti. Lisäksi sovelluksessa hyödynnetään useRef-koukkua, jonka avulla voidaan tallentaa ja päivittää viitteitä ilman, että ne aiheuttavat näytön uudelleenrenderöintiä.

Ohjelmakoodi 4 useState- ja useRef-koukut.

```
const [location, setLocation] = useState(null);
const [region, setRegion] = useState(null);
const distanceTraveled = useRef(0);
const prevLocation = useRef(null);
const [tracking, setTracking] = useState(false);
const [isPaused, setIsPaused] = useState(false);
const [elapsedTime, setElapsedTime] = useState(0);
const intervalRef = useRef(null);
const [startButtonVisible, setStartButtonVisible] = useState(true);
const [pauseButtonVisible, setPauseButtonVisible] = useState(false);
const [saveButtonVisible, setSaveButtonVisible] = useState(false);
const [showFinalDetails, setShowFinalDetails] = useState(false);
const [averageSpeedPerHour, setAverageSpeedPerHour] = useState(0);
const [polylineCoordinates, setPolylineCoordinates] = useState([]);
const [startTime, setStartTime] = useState(null);
const [endTime, setEndTime] = useState(null);
```

5.5.1 Kartan liittäminen sovellukseen ja käyttäjän sijaintietojen haku

Sovelluksen oleellisin osa on hakea kartta Google Mapsin ohjelmointirajapinnasta ja tulostaa se ruudulle. Tämä tehdään React Native Maps -kirjaston MapView -komponentilla.

React Native Maps on Google Maps SDK -työkalupakettia käyttävä kirjasto, joka tarjoaa interaktiivisten karttojen integroinnin React Native -sovelluksiin. MapView -komponentilla saadaan sisällytettyä interaktiivinen karttanäkymä sovellukseen. Karttanäkymää varten tulee määritellä alue (region). Alueeseen tulee neljä ominaisuutta, jotka ovat latitude ja longitude, mitkä määrittelevät näytettävän sijainnin sekä latitudeDelta ja longitudeDelta, mitkä määrittelevät kuinka suurennettuna näytettävä sijainti esitetään. Nämä voidaan määritellä syöttämällä halutun sijainnin koordinaatit ominaisuuksien arvoiksi. Koska sovelluksessa halutaan hakea käyttäjän senhetkinen sijainti, käytetään sijainnin hakuun ohjelmointirajapintaa.

Käyttäjän sijaintiedot voidaan hakea laitteen käyttöjärjestelmän tarjoamista sijaintipalveluista React Nativen Geolocation -ohjelmointirajapinnan avulla. Käyttäjän laitteen sijainnin

koordinaatit haetaan ohjelmakoodin 5 mukaisesti Geolocation -APIn getCurrentPosition -funktion avulla. Nämä määritetään region -muuttujan arvoiksi, mitkä MapView -komponentti hakee näytettävän sijainnin ominaisuuksiksi. Käyttämällä useFocusEffect -koukkua, haetaan käyttäjän sijaintiedot aina, kun navigoinnin kohdenäyttö tulee fokukseen eli kun käyttäjä navigoi takaisin lenkin seurannan aloituksen näkymään navigointipalkista.

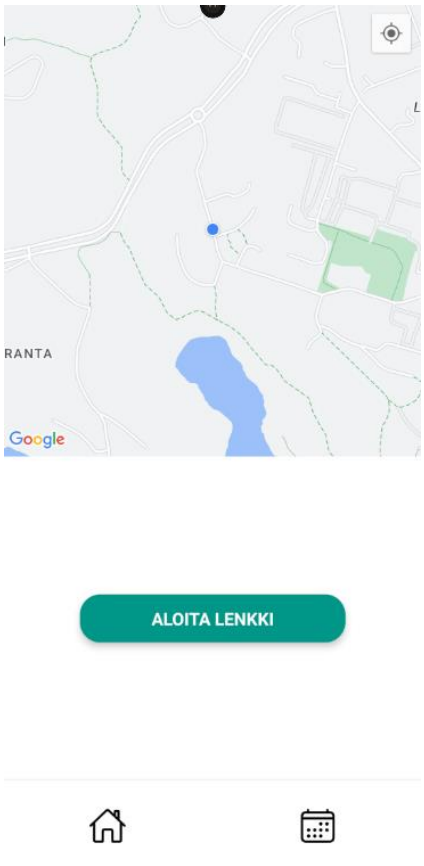
Ohjelmakoodi 5 Laitteen nykyisen sijainnin haku.

```
useFocusEffect(
useCallback(() => {
  fetchCurrentPosition();
}, [])
);

const fetchCurrentPosition = () => {
  Geolocation.getCurrentPosition(
    position => {
      const { latitude, longitude } = position.coords;
      setLocation({ latitude, longitude });
      setRegion({
        latitude,
        longitude,
        latitudeDelta: 0.01,
        longitudeDelta: 0.01,
      });
    },
    error => console.log(error),
    { enableHighAccuracy: true, timeout: 20000, maximumAge: 1000 }
  );
};
```

Kun käyttäjän nykyinen sijainti on haettu, tulee kartta käyttäjän sijainnilla näkyviin sovelluksen aloitusnäkymään kuten kuvassa 7 näkyy.

Kuva 7 Sovelluksen aloitusnäkymä.

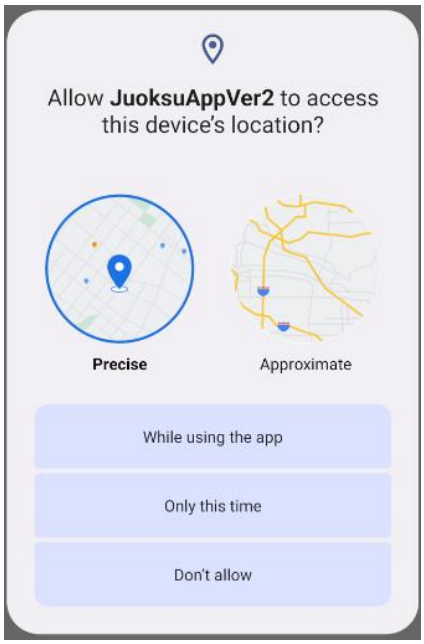


Jotta sovellus voi seurata laitteen sijaintia, sijaintitietojen lukeminen edellyttää käyttäjältä lupaa sijainti-APIn käyttämiseen. Tämä lupa varmistaa käyttäjän yksityisyyden säilymisen ja antaa sovellukselle tarvittavat oikeudet käyttää laitteen sijaintitietoja. Sovellukselle voidaan myös antaa oikeus käyttää sijaintitietoja taustalla, jolloin sovellus voi kerätä tietoja silloinkin, kun se ei ole aktiivisessa käytössä. Näiden lupa-asetusten saamiseksi on lisättävä seuraavat rivit AndroidManifest.xml -tiedostoon, joka esiteltiin aiemmin ohjelmakoodissa 1:

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
```

Kun sovellus käynnistetään näiden rivien lisäämisen jälkeen, pyytää sovellus pääsyä ja käyttöoikeutta sijaintitietoihin ponnahdusikkunalla kuten kuvassa 8 on esitetty. Käyttäjä voi valita, haluaako hän myöntää tai kieltää luvan.

Kuva 8 Käyttäjältä kysyttävä lupa sijaintietojen käyttämiseen.



5.5.2 Lenkin seuranta ja kuljetun lenkin piirtäminen kartalle

Käyttäjä voi käynnistää lenkin seurannan painamalla ruudulla ja kuvassa 7 näkyvää ”Aloita lenkki” -painiketta. Tämän jälkeen käyttäjän sijaintia seurataan ja päivitetään karttaan aina, kun se muuttuu. Tämä toteutetaan Geolocation -APIn watchPosition -funktion avulla, joka tarkkailee jatkuvasti laitteen sijaintia ja päivittää sijaintitiedot aina niiden muuttuessa. Ohjelmakoodi 6 mukaisesti watchPosition -funktio on asetettu useEffect-koukun sisään, johon on määritelty riippuvuus, ja se suoritetaan, kun tracking -tila saa arvon true eli kun lenkin seuranta on käynnissä ja kun isPaused -tila saa arvon true eli kun lenkin seuranta on hetkellisesti keskeytetty. useEffect-koukku mahdollistaa toiminnallisuuden suorittamisen tiettyihin aikoihin, kuten näytön renderöinnin jälkeen tai tietyn tilan muuttuessa. Jos lenkin seuranta on käynnissä ja seuranta ei olla pysäytetty, uudet koordinaatit tallennetaan region -muuttujan arvoiksi ohjelmakoodissa 7 esitetyssä updateLocation -funktiossa, jotka ohjelmakoodissa 9 esitetty MapView -komponentti käyttää karttaan päivittämiseen.

Ohjelmakoodi 6 Laitteen muuttuvan sijainnin haku.

```
useEffect(() => {
  let watchId;

  if (tracking && !isPaused) {
    watchId = Geolocation.watchPosition(
    updateLocation,
    (error) => console.log(error),
    {
      enableHighAccuracy: true,
      distanceFilter: 1,
```

```
        interval: 5000,
        fastestInterval: 2000,
        timeout: 30000,
        maximumAge: 0
      }
      );

      intervalRef.current = setInterval(() => {
        setElapsedTime(prev => prev + 1);
      }, 1000);
    } else {
      clearInterval(intervalRef.current);
    }

    return () => Geolocation.clearWatch(watchId);
  }, [tracking, isPaused]);
```

Ohjelmakoodi 7 updateLocation -funktio muuttuvan sijainnin päivitykseen.

```
const updateLocation = (position) => {
  const { latitude, longitude } = position.coords;
  const newCoordinate = { latitude, longitude };

  if (prevLocation.current && tracking) {
    const distance = calculateDistance(prevLocation.current, {
latitude, longitude });
    distanceTraveled.current += distance;
  }
  setLocation({ latitude, longitude });
  setRegion({
    latitude,
    longitude,
    latitudeDelta: 0.01,
    longitudeDelta: 0.01,
  });
  prevLocation.current = { latitude, longitude };
  setPolylineCoordinates(prevState => [...prevState, newCoordinate]);
};
```

Lenkin seurantasivulla on aluksi näkyvissä ”Aloita lenkki” -painike lenkin seurannan aloittamiseksi. Kun tätä painiketta painetaan, kutsutaan startTracking -funktiota, joka käynnistää useita toimenpiteitä, kuten ohjelmakoodissa 8 on esitetty. Funktio asettaa tracking-tilan arvoksi true, mikä käynnistää lenkin seurannan. Lisäksi sekuntilaskuri käynnistetään, joka laskee lenkin keston nollasta alkaen, kaikki lenkin tilastotiedot nollataan ja ”Aloita lenkki” -painike piilotetaan samalla, kun ”Lopeta lenkki” ja ”Pysäytä lenkki” -painikkeet tuodaan näkyviin.

Ohjelmakoodi 8 startTracking -funktio.

```
const startTracking = () => {
  setTracking(true);
  setStartTime(new Date());
  setElapsedTime(0);
  setAverageSpeedPerHour(0);
  distanceTraveled.current = 0;
  setStartButtonVisible(false);
  setPauseButtonVisible(true);
  setShowFinalDetails(false);
```

```
    setPolylineCoordinates([]);
  }
```

Käyttäjän sijainti päivittyy kartalla dynaamisesti, joten aina kun uusi sijainti haetaan, kartta päivittyy automaattisesti näyttämään käyttäjän ajantasaisen sijainnin. Käyttäjän sijainti näytetään MapView -komponentin showUserLocation -ominaisuudella. Käyttäjän kulkema reitti kartalla piirretään React Native Mapsin Polyline -komponentilla. Polyline -komponentti mahdollistaa viivojen piirtämisen kartalle. Nämä viivat koostuvat useista peräkkäisistä pisteistä, jotka on määritelty koordinaattien avulla.

Koska kartta renderöidään aina uudelleen jokaisen sijainnin muutoksen yhteydessä, aiemmat kartalle piirretyt viivat pyyhkiytyvät pois. Sovelluksessa on tämän takia luotu array-tilamuuttuja nimeltä polylineCoordinates, johon tallennetaan kaikki käyttäjän kulkeman reitin koordinaatit. Aina kun käyttäjän sijainti muuttuu ja samalla uudet sijaintikoordinaatit haetaan, ne lisätään tilamuuttujaan setPolylineCoordinates -asettajalla kuten ohjelmakoodissa 7 oli esitetty. Jokaisen kartan uudelleen renderöinnin yhteydessä käyttäjän kulkeman reitin viivat piirretään uudelleen käyttäen polylineCoordinates -tilamuuttujassa olevia koordinaatteja. Tämä varmistaa, että käyttäjän kulkema reitti pysyy piirrettynä kartalla koko lenkin ajan. Ohjelmakoodissa 9 on esitetty MapView- ja Polyline- komponenttien käyttö sovelluksessa ja niille määritellyt ominaisuudet.

Ohjelmakoodi 9 MapView -komponentti.

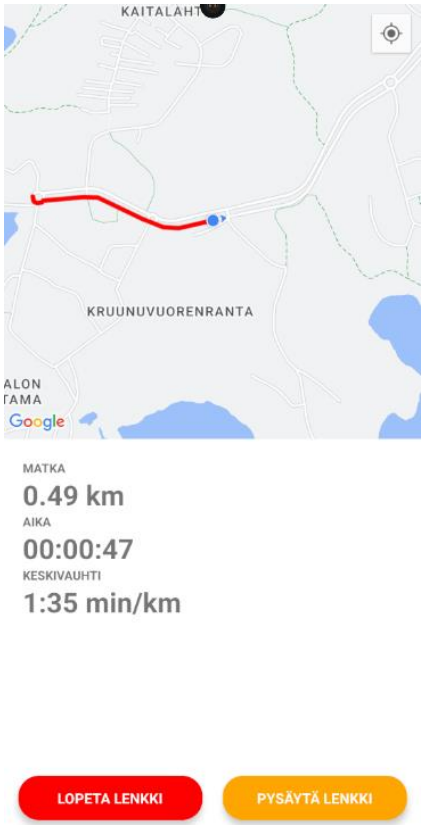
```
<MapView
  style={styles.map}
  mapType={"standard"}
  provider={PROVIDER_GOOGLE}
  showsMyLocationButton={true}
  showsUserLocation= {true}
  region={region}>
  {tracking && (
    <Polyline
      coordinates={polylineCoordinates}
      strokeWidth={4}
      strokeColor={"#ff0000"}
    />
  )}
  {showFinalDetails && (
    <Polyline
      coordinates={polylineCoordinates}
      strokeWidth={4}
      strokeColor={"#ff0000"}
    />
  )}
</MapView>
```

5.5.3 Lenkin tietojen päivitys ruudulle sekä lenkin pysäytys ja päättäminen

Sovellus tallentaa käyttäjän lenkin tietoja useilla tavoilla. Kun lenkin seuranta käynnistetään (Ohjelmakoodi 8), startTime -tilamuuttujaan kirjataan lenkin aloitusajankohta. Tämän lisäksi siihen tallennetaan lenkin suorituksen päivämäärä, jota hyödynnetään lenkin tietojen tallentamisessa tietokantaan. ElapsedTime -tilamuuttujaan päivitetään lenkin kesto sekunteina, ja tätä tietoa näytetään myös käyttöliittymässä sen muuttuessa. DistanceTravelled -tilamuuttujaan tallennetaan käyttäjän kulkema matka kilometreinä. Matka lasketaan käyttämällä Haversinen kaavaa, joka perustuu käyttäjän edellisiin ja nykyisiin sijaintikoordinaatteihin.

Käyttäjän kulkema matka päivitetään näytölle aina sen muuttuessa, ja se esitetään kilometreinä. ElapsedTime -tilamuuttujaan päivitettävä lenkin kesto sekunteina näytetään myös ruudulla lenkin aikana. Lenkin seurannan käynnistyessä näytölle ilmestyy käyttäjän keskivauhti, jota päivitetään aina sen arvon muuttuessa. Tätä varten on luotu funktio, joka laskee käyttäjän keskivauhdin hyödyntämällä kuljettua matkaa ja kulunutta aikaa. Kuvassa 9 on esitetty, millainen näkymä on lenkin ollessa käynnissä.

Kuva 9 Käynnissä olevan lenkin näkymä.



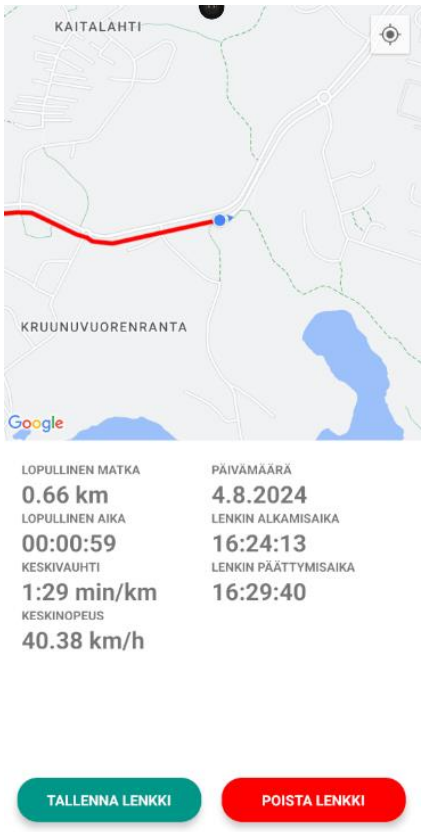
Lenkin seurannan ollessa käynnissä näytöllä näkyvät painikkeet lenkin lopettamiseen ja pysäyttämiseen. ”Pysäytä lenkki” -painikkeen painamisen jälkeen tilalle ilmestyy ”Jatka lenkkiä” -painike, ja ruudulla näkyvien lenkkitietojen päivitys pysäytetään siksi aikaa, kun lenkki on pysäytettynä. Tämä toteutetaan ohjelmakoodin 6 if -lauseella, jossa tietoja päivitetään vain, kun lenkin seuranta on aktiivinen eikä sitä ole pysäytetty. Kun ”Jatka lenkkiä” -painiketta painetaan, lenkin seurantaa jatketaan normaalisti.

Kun ”Lopeta lenkki” -painiketta painetaan, kutsutaan stopTracking -funktiota, joka on esitetty ohjelmakoodissa 10. Tämä funktio asettaa tracking -tilan arvoksi false, mikä pysäyttää lenkin seurannan. Lenkin seurannan päättyessä lenkin päättymisaika tallennetaan endTime -tilamuuttujaan, ja käyttäjän keskinopeus averageSpeedPerHour -tilamuuttujaan. Samalla ”Lopeta lenkki” -painike piilotetaan, ja ”Tallenna lenkki”- sekä ”Poista lenkki” -painikkeet tuodaan näkyviin. Ruudulle tuodaan näkyviin myös kaikki lenkin tiedot. Kuvassa 10 on esitetty näkymä päättyneestä lenkistä.

Ohjelmakoodi 10 stopTracking -funktio.

```
const stopTracking = async () => {
  setTracking(false);
  setEndTime(new Date());
  setStartButtonVisible(true);
  setSaveButtonVisible(true);
  setShowFinalDetails(true);
  setAverageSpeedPerHour(((distanceTraveled.current / elapsedTime) /
1000) * 3600);
}
```

Kuva 10 Päättyneen lenkin näkymä.



Kun lenkin seuranta on lopetettu, käyttäjälle tarjotaan mahdollisuus tallentaa lenkin tiedot tietokantaan tai poistaa ne muistista. Molempien toimenpiteiden jälkeen lenkin tiedot nollataan ja näkymä palautetaan alkuperäiseen lenkin seurannan aloitusnäkymään.

5.6 Tietokanta ja historiasivu

Sovelluksen tietokanta rakennettiin SQLitellä. Sen tallentamat tiedot tallennetaan suoraan käyttöjärjestelmän tiedostojärjestelmään. SQLite on ihanteellinen ratkaisu juuri tämänkaltaisille yksinkertaisille sovelluksille, joissa on vain yksi käyttäjä, koska se on kevyt ja helppo käyttää.

Sovelluksessa voidaan tallentaa käyttäjän juostun lenkin tiedot tietokantaan, käyttäjän niin halutessa myöhempää tarkastelua varten. Tallennettavat tiedot ovat id, lenkin päivämäärä, aloitusaika, päättymisaika, kokonaismatka, kokonaisaika, keskivauhti, keskinopeus ja lenkin koordinaatit. Automaattisesti muodostettava id-numero, joka toimii pääavaimena tiedonhakuun ja -muokkaukseen, tallennetaan tietokantaan kokonaislukuna (INTEGER). Lenkin päivämäärä, aloitusaika ja päättymisaika tallennetaan tekstimuodossa (TEXT).

Kokonaismatka, joka esittää lenkin keston metreinä, tallennetaan liukulukuna (REAL). Myös lenkkiin kulunut kokonaisaika tallennetaan liukulukuna. Keskivauhti, joka ilmaisee kuinka monta minuuttia kilometrin juoksemiseen kuluu, ja keskinopeus, joka mittaa kuinka monta kilometriä juostaan tunnissa, tallennetaan molemmat liukulukuna. PolylineCoordinates -tilamuuttujassa olevat koko juostun lenkin koordinaatit tallennetaan tekstimuodossa. Koska koordinaatit ovat JSON-muodossa, ne on muunnettava tekstimuotoon ennen tallentamista tietokantaan.

Sovelluksessa määriteltiin yhteys olemassa olevaan tietokantaan, ja mikäli tietokantaa ei ollut, luotiin uusi tietokanta ja yhteys. Ohjelmakoodissa 11, missä tämä yhteys ja tietokanta luodaan, sisällytettiin kaikkiin sovelluksen näyttöihin ja se suoritetaan vain kerran komponentin elinkaaren aikana. db.transaction -metodia hyödynnettiin tietokantatransaktion luomiseksi ja tx.executeSql -metodia käytettiin SQL-kyselyn suorittamiseen tietokannassa. SQL-kyselyssä luodaan uusi taulu nimeltä trackerDatabase, mikäli sellaista ei vielä ole olemassa.

Ohjelmakoodi 11 Tietokannan luominen.

```
const db = openDatabase({ name: 'trackerDatabase.db' });

useEffect(() => {
  db.transaction((tx) => {
    tx.executeSql(
      'CREATE TABLE IF NOT EXISTS trackerDatabase (id INTEGER PRIMARY
KEY AUTOINCREMENT, pvm TEXT NOT NULL, aloitusAika TEXT NOT NULL,
lopetusAika TEXT NOT NULL, matkaPituus REAL NOT NULL, lopullinenAika
REAL NOT NULL, keskiVauhti REAL NOT NULL, keskiNopeus REAL NOT NULL,
reittiKoordinaatit TEXT)',
      []
    );
  });
}, []);
```

5.6.1 Lenkin tietojen tallennus tietokantaan

Lenkin päätyttyä, käyttäjän halutessa tallentaa lenkin tiedot tietokantaan, kutsutaan ohjelmakoodissa 12 esitettyä saveTracking -funktiota. Funktiossa kerätään ensin lenkin tiedot ja muutetaan ne tarvittavaan muotoon tietokantaa varten. Tämän jälkeen suoritetaan SQL-kysely tx-executeSql -metodilla, joka lisää uuden rivin trackerDatabase -tauluun. Tauluun lisätään tiedot, jotka on valmisteltu parametreina. Tässä vaiheessa myös JSON-muodossa olevat lenkin koordinaatit muunnetaan merkkijonomuotoon JSON.stringify -metodilla, jotta ne voidaan tallentaa tietokantaan. Funktion lopuksi kutsutaan eraseTrackingInfo -funktiota, joka tyhjentää kaikki seurannan tiedot.

Ohjelmakoodi 12 Lenkin tietojen tallennus tietokantaan.

```
const saveTracking = () => {
  const dateOfRun = startTime.toLocaleDateString('fi-FI', { timeZone:
'UTC' });
  const runStartTime = startTime.toLocaleTimeString([], {hour: '2-
digit', minute:'2-digit', second:'2-digit', hour12: false});
  const runEndTime = endTime.toLocaleTimeString([], {hour: '2-digit',
minute:'2-digit', second:'2-digit', hour12: false});
  const totalDistance = (distanceTraveled.current / 1000).toFixed(2);
  const totalDuration = formatTime(elapsedTime);
  const pace = calculateMinutesPerKilometer();
  const averageSpeed = averageSpeedPerHour.toFixed(2);
  const routeCoordinates = polylineCoordinates;

  db.transaction(tx => {
    tx.executeSql(
      'INSERT INTO trackerDatabase (pvm, aloitusAika, lopetusAika,
matkaPituus, lopullinenAika, keskiVauhti, keskiNopeus,
reittiKoordinaatit) VALUES (?, ?, ?, ?, ?, ?, ?, ?)',
      [dateOfRun, runStartTime, runEndTime, totalDistance,
totalDuration, pace, averageSpeed, JSON.stringify(routeCoordinates)],
      (tx, results) => {
        if (results.rowsAffected > 0) {
          console.log('Data tallennettu tietokantaan onnistuneesti');
        } else {
          console.log('Datan tallennus epäonnistui');
        }
      },
      error => {
        console.log('Error: ' + error.message);
      }
    );
  });
  eraseTrackingInfo();
};
```

5.6.2 Historiasivun luonti ja lenkin poistaminen tietokannasta

Sovelluksen toisella pääsivulla, jota kutsutaan historiasivuksi, käyttäjä voi tarkastella sekä poistaa tietokantaan tallennettuja lenkkejä. Kaikki puhelimeen tallennetut lenkkitiedot haetaan ohjelmakoodin 13 mukaisesti fetchTrackingHistory -funktiolla ja jokainen tietokannassa oleva lenkki tulostetaan ruudulle ScrollView -komponentin sisälle ohjelmakoodin 14 mukaisesti. ScrollView -komponentin avulla sisältöä voidaan vierittää pystysuunnassa käyttöliittymässä, mikä mahdollistaa sisällön näyttämisen, joka ei mahdu kerralla näytölle. Näytölle tulostetaan vain osa tiedoista, ja käyttäjä voi valita haluamansa lenkin tarkastellakseen sen kaikkia tietoja erikseen.

Ohjelmakoodi 13 Historiatietojen haku tietokannasta.

```
const fetchTrackingHistory = () => {
  db.transaction(tx => {
    tx.executeSql(
      'SELECT * FROM trackerDatabase',
      [],
      (tx, results) => {
```

```
        let history = [];
        for (let i = 0; i < results.rows.length; i++) {
          history.push(results.rows.item(i));
        }
        history.sort((a, b) => b.id - a.id);
        setTrackingHistory(history);
      },
      error => {
        console.log('Error: ' + error.message);
      }
    );
  });
};
```

Ohjelmakoodi 14 Lenkkihistorian listaus.

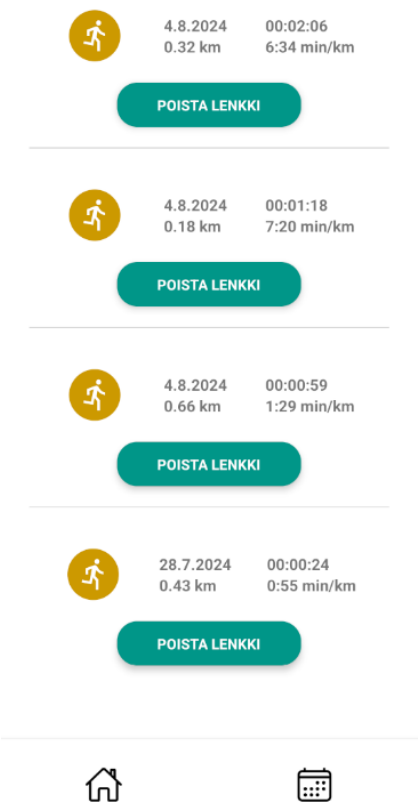
```
<ScrollView style={styles.historyContainer}
contentContainerStyle={styles.scrollViewContent}>
  {trackingHistory.map((item, index) => (
    <Fragment key={item.id}>
      <TouchableOpacity onPress={() => openRunDetails(item)}>
        <View style={styles.historyContainer2}>
          <View style={styles.historyItem}>
            <Icon name="run-circle" size={60} color="#cc9900" />
          </View>
          <View style={styles.historyItem}>
            <Text style={styles.text1}>{item.pvm}</Text>
            <Text style={styles.text1}>{item.matkaPituus} km</Text>
          </View>
          <View style={styles.historyItem}>
            <Text style={styles.text1}>{item.lopullinenAika}</Text>
            <Text style={styles.text1}>{item.keskiVauhti}
min/km</Text>
          </View>
        </View>
      </TouchableOpacity>

      <View style={styles.historyItem2}>
        <TouchableOpacity onPress={() => confirmDeleteRun(item.id)}
style={styles.appButtonContainer1}>
          <Text style={styles.buttonText}>Poista lenkki</Text>
        </TouchableOpacity>
      </View>

      {index < trackingHistory.length - 1 && (
        <View style={styles.separator} />
      )}
    </Fragment>
  ))}
</ScrollView>
```

Jokainen erillinen lenkki on laitettu TouchableOpacity -komponentin sisään. TouchableOpacity -komponentti tarjoaa kosketus- tai painalluspalautteen käyttäjälle. Käyttäjän painaessa tarkastelun kohteeksi haluamaansa lenkkiä, avataan se omaan ikkunaansa Modal -komponentin avulla. Modal -komponentti tarjoaa tavan näyttää sisältöä, joka peittää nykyisen näkymän ja estää käyttäjää vuorovaikuttamasta taustan kanssa, kunnes modal-ikkuna suljetaan. Kuvassa 11 on esitetty historiasivun näkymä.

Kuva 11 Historiasivun näkymä.



Kun kaikki tietokannassa olevat lenkit tulostetaan historiasivulle, ruudulle tulostetaan myös jokaisen lenkin alle käyttäjäpainike, jonka avulla käyttäjä voi poistaa haluamansa lenkin tietokannasta. Kun ”Poista lenkki” -painiketta painetaan, avataan modal-ikkuna, jossa varmistetaan, haluaako käyttäjä poistaa kyseisen lenkin. Jos käyttäjä vahvistaa poistamisen, suoritetaan ohjelmakoodi 15, joka poistaa käyttäjän valitseman lenkin tietokannasta.

Ohjelmakoodi 15 Lenkin poistaminen tietokannasta.

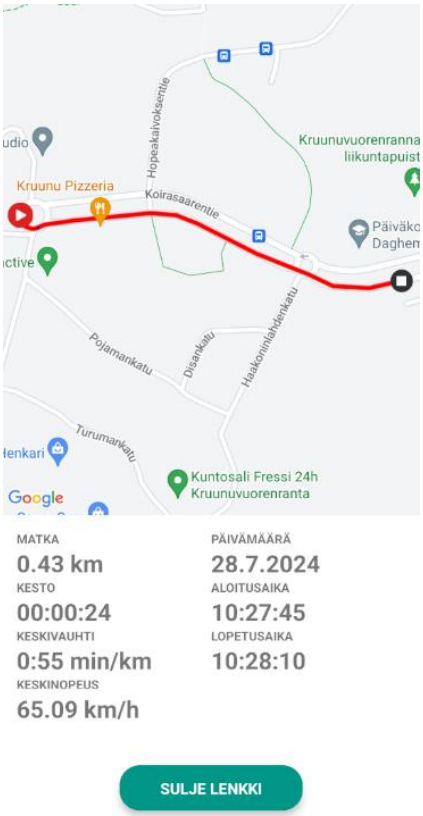
```
const eraseRunFromDatabase = () => {
  if (runToDelete === null) return;

  db.transaction(tx => {
    tx.executeSql(
      'DELETE FROM trackerDatabase WHERE id = ?',
      [runToDelete],
      () => {
        console.log(`Lenkki id:llä ${runToDelete} poistettu
onnistuneesti`);
        fetchTrackingHistory();
        setConfirmDeleteModalVisible(false);
        setRunToDelete(null);
      },
      error => {
        console.log('Virhe poistettaessa lenkkiä id:llä: ' +
error.message);
      }
```

```
        );
      });
    };
```

Kun lenkki valitaan tarkasteltavaksi historiasivulta, tulostetaan lenkin kaikki tiedot koko ruudun kattavaan modal-ikkunaan. Ohjelmakoodissa 13 on jo haettu kaikki tietokannassa olevat lenkit ja niiden tiedot, joten lenkin tietoja ei tarvitse enää hakea tietokannasta. Tiedot tulostetaan ruudulle samalla tavalla kuin lenkin seurantasivulla, kun lenkki päätettiin ja lenkin tilastotiedot tulostettiin. Käyttäjän juoksema lenkki näytetään myös kartalla käyttämällä MapView -komponenttia, ja lenkki piirretään PolyLine -komponentilla tietokantaan tallennettujen koordinaattien avulla. Koordinaateista haetaan äärisijainnit ja lasketaan kartan keskipiste, mikä varmistaa, että kartan näkymä on oikeassa paikassa ja koko kuljettu lenkki mahtuu karttanäkymään. Karttaan merkitään myös Marker -komponentilla ja tallennetulla kuvatiedostolla lenkin alku- ja päätepisteet selkeästi. Tarkasteltava lenkki voidaan sulkea painamalla ’Sulje lenkki’ -painiketta. Kuvassa 12 on esitetty, miltä valitun lenkin näkymä näyttää tarkastelutilassa.

Kuva 12 Tarkasteltavan lenkin näkymä.



5.7 Navigointijärjestelmän luonti

Mobiilisovellus koostuu kahdesta eri näytöstä: lenkin seurantasivusta, jolla lenkki käynnistetään ja sitä seurataan, sekä historiasivusta, jossa tietokantaan tallennettuja lenkkejä voi tarkastella. Navigointi näiden kahden sivun välillä on toteutettu React navigation -kirjastolla ja sen native- ja bottom-tabs -moduuleilla. Bottom-tabs-moduulilla luodaan yksinkertainen välilehtipalkki ruudun alareunaan, jonka avulla eri näyttöjen välillä voidaan navigoida. Sovelluksen navigointijärjestelmä on esitetty ohjelmakoodissa 16. Aluksi tuodaan tarvittavat kirjastot ja näytöt, jotka navigoinnissa esitetään. Pohja välilehtinavigoinnille luodaan CreateBottomTabNavigator -komponentilla. Koko navigointirakenne on kääritty NavigationContainer -komponentin alle. Navigator -komponentilla määritellään välilehtinavigoinnin asetukset ja näytöt, joiden välillä navigoidaan. Ionicons -kirjastoa käytetään navigointipalkin ulkoasun ikonien tuomiseen. Kuvan 11 alalaidassa on esitetty, miltä navigointipalkki näyttää.

Ohjelmakoodi 16 Navigointijärjestelmä

```
import React, { useEffect } from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createBottomTabNavigator } from '@react-navigation/bottom-
tabs';
import Icon from 'react-native-vector-icons/Ionicons';
import TrackerScreen from './screens/TrackerScreen';
import HistoryScreen from './screens/HistoryScreen';

const Tab = createBottomTabNavigator();

const MyTabs = () => {
return (
<NavigationContainer>
  <Tab.Navigator
    screenOptions={{
      headerShown: false,
      tabBarStyle: { backgroundColor: '#FFFFFF', height: '12%' },
      tabBarLabelStyle: {
        color: '#000000',
        fontWeight: 'bold',
      },
      tabBarShowLabel: false,
    }}>
    <Tab.Screen
      name="Tracker"
      component={TrackerScreen}
      options={{
        title: 'Tracker',
        tabBarIcon: ({ color, size }) => (
          <Icon name="home-outline" color={'#000000'} size={38}
style={{ alignSelf: 'center', marginTop: -15 }} />
        ),
      }}
    />
    <Tab.Screen
      name="History"
      component={HistoryScreen}
```

```
      options={{
        title: 'History',
        tabBarIcon: ({ color, size }) => (
          <Icon name="calendar-outline" color={'#000000'} size={38}
style={{ alignSelf: 'center', marginTop: -15 }} />
        ),
      }}
    />
  </Tab.Navigator>
</NavigationContainer>
  );
};

export default MyTabs;
```

5.8 Sovelluksen testaus

Aluksi sovellusta kehitettiin ja testattiin Expo -kehitysalustalla, mutta tämä lähestymistapa hylättiin melko nopeasti Expon emulaattorin ongelmien vuoksi, erityisesti laitteen sijaintitietojen hakemisessa testauksen aikana. Tämän vuoksi päätettiin siirtyä Android Studion Android -emulaattorin käyttöön, mikä mahdollisti sovelluksen testaamisen kehitysvaiheessa fyysisen laitteen kaltaisesti. Koska Android Studio ja sen emulaattori olivat jo ennestään tuttuja, ne osoittautuivat hyväksi valinnaksi opinnäytetyön testausalustaksi. Emulaattoriin voitiin määrittää halutut koordinaatit, jossa haluttiin emulaattorin sijaitsevan, ja laitteen liikettä simuloitiin luomalla reitti, jota pitkin laite liikkuisi määritetyllä nopeudella. Nämä ominaisuudet osoittautuivat erittäin hyödyllisiksi sovelluksen testauksessa jo varhaisessa kehitysvaiheessa.

Sovelluksen valmistuttua sitä testattiin myös fyysisillä laitteilla. Tämä varten luotiin APK-tiedosto, joka on Android-sovelluksen asennuspaketti sisältäen kaikki sovelluksen ajamiseen tarvittavat tiedostot ja koodit. APK-tiedosto siirrettiin mobiililaitteelle ja asennettiin avaamalla se suoraan laitteessa. Vaihtoehtona olisi ollut julkaista sovellus Googlen Play -kaupassa, mutta tämä jätettiin tekemättä sovellukseen sisältyvän API-avaimeen liittyvän mahdollisen tietoturvariskin vuoksi.

Sovellus toimi odotetusti myös fyysisellä mobiililaitteella, mutta testauksen aikana ilmeni asioita, joita emulaattorilla ei voitu ennakoida. Esimerkiksi kävi ilmi, että lenkin seuranta ei toiminut kunnolla, kun laitteen näyttö sammui tai sovellus oli käynnissä taustalla muiden sovellusten ollessa auki. Yritin ratkaista ongelman käyttämällä background geolocation -pakettia, joka mahdollistaa sijainnin seurannan myös taustalla. Kuitenkin kävi nopeasti selväksi, että kyseinen palvelu on maksullinen, ja vaihtoehtoisen tavan koodaaminen olisi tässä vaiheessa ollut liian aikaa vievää ja haasteellista ohjelmointitaitoni huomioon ottaen. Tämän vuoksi ongelma ratkaistiin väliaikaisesti lisäämällä toiminto, joka pitää sovelluksen jatkuvasti päällä.

Fyysisellä laitteella testattaessa havaittiin myös, että lenkin seurannan aikana käyttäjä pystyi poistumaan seurannasta siirtymällä toiseen näkymään valikointipalkin kautta, joten tämä mahdollisuus poistettiin myöhemmin. Lisäksi sijaintikoordinaattien haussa ilmeni ajoittain ongelmia, esimerkiksi suurten rakennusten sisällä tai sillan alta kuljettaessa. Näiden ongelmien vuoksi sovellus ei aina toiminut odotetulla tavalla, sillä sijaintitietojen puuttuessa karttaa ei voitu näyttää ruudulla. Myös lenkin tietoihin ja koordinaatteihin tuli vääristymiä, kun kesken lenkin ei saatukaan kerättyä haluttuja tietoja.

Opinnäytetyön teoriaosuudessa korostettiin, että ohjelmointirajapinnan onnistunut integrointi vaatii asianmukaista virheiden käsittelyä, turvallisuusnäkökulmien huomioimista ja perusteellista testausta. Tämä ei kuitenkaan täysin toteutunut projektin aikana ajanpuutteen vuoksi, mikä olisi todennäköisesti voinut ratkaista myös aiemmin mainitun ongelman sijaintietojen haussa. Sovelluksen ohjelmointivaiheessa ei toteutettu kattavaa virheidenkäsittelyä tilanteisiin, joissa ohjelma käyttäytyy odottamattomasti. Myöskään tilanteisiin, joissa internetyhteys puuttuu tai GPS on pois päältä, ei ole lisätty virheenkäsittelyä. Näihin asioihin tullaan kiinnittämään enemmän huomiota myöhemmin sovellusta tulevassa kehityksessä.

Sovelluksen toimivuutta arvioitiin myös vertaamalla sen keräämiä ja laskemia tietoja markkinoilla olevan vastaavan sovelluksen hakemiin tietoihin. Testi suoritettiin juoksemalla lenkki ja käynnistämällä molemmat sovellukset samanaikaisesti, jotta voitiin arvioida ohjelmoidun sovelluksen tarkkuutta. Testissä ei havaittu merkittäviä eroja sovellusten antamissa tiedoissa. Ohjelmoidun sovelluksen mittaama matka oli muutamia kymmeniä metrejä pidempi, mikä saattoi johtua siitä, että sijaintitietojen haku oli määritelty tarkemmaksi ja päivittyväksi tiheämmin kuin vertailusovelluksessa. Näin ollen ohjelmoidun sovelluksen hakemien tietojen todettiin olevan varsin lähellä todellisia arvoja.

5.9 Mobiilisovelluksen kehitysmahdollisuudet

Opinnäytetyössä kehitettyä mobiilisovellusta aiotaan laajentaa lisäämällä siihen uusia toimintoja. Tällä hetkellä sovellus on suunniteltu ensisijaisesti lenkkeilyyn, mutta koska se seuraa ainoastaan laitteen sijaintia ja sen muutosta, sitä voidaan helposti soveltaa myös muihin liikuntamuotoihin. Tulevassa kehityksessä on tarkoitus lisätä ominaisuus, joka mahdollistaa liikuntamuodon valinnan. Tällöin sovelluksen näyttämät ja tietokantaan tallennettavat tiedot voidaan räätälöidä paremmin vastaamaan valittua liikuntamuotoa.

Sovellukseen on suunnitteilla kalenteri, jonka avulla käyttäjä voi tarkastella lenkkihistoriaansa. Tarkoituksena on myös lisätä toiminto, joka mahdollistaa käyttäjän

asettaa tavoitteita ja suunnitelmia liikunnan suhteen, kuten kilometrimäärän lenkillä, lenkin pituuden, keskinopeuden tai vauhdin tavoitteet. Sovellus ilmoittaisi käyttäjälle, kun nämä tavoitteet on saavutettu, ja myös silloin, kun henkilökohtaisia ennätyksiä, kuten juoksunopeus tai lenkin pituus, on rikottu. Lisäksi suunnitelmissa on lisätä sovellukseen diagrammeja, jotka kuvaavat kehitystä näissä asioissa.

Sovellusta jatkokehityksessä voitaisiin myös laajentaa sen toimintoja hyvinvointisovelluksen suuntaan, esimerkiksi lisäämällä kuntosaliharjoitusten seurantasovellus, ruokavaliokalenteri sekä mahdollisuus manuaalisesti lisätä ja seurata käyttäjän painoa ja painotavoitteita. Myös aiemmin mainitut virheenkäsittely ja sovelluksen toiminta taustalla ja näytön ollessa suljettuna tullaan jatkossa lisäämään sovellukseen.

Koska sovelluksen ohjelmointi oli opinnäytetyön tekijälle myös oppimiskokemus, toteutui sovelluksen ohjelmointi sitä mukaa, kun tieto ja kokemus kasvoi. Tämän seurauksena ohjelmoitava koodi kehittyi ja muokkautui projektin aikana, ja on todennäköistä, että tietyt osat olisi voitu toteuttaa lyhyemmällä ja siistimmällä koodilla. Sovelluksessa saattaa olla käytetty menetelmiä, jotka eivät ole tarkoitukseensa tehokkaita. Jos ohjelmointi aloitettaisiin nyt täysin alusta, monet asiat voitaisiin toteuttaa toisin.

6 Yhteenveto

Opinnäytetyössä tutkittiin aluksi ohjelmointirajapintojen käsitettä, niiden toimintaa ja kommunikointitapoja. Samalla perehdyttiin Google Mapsin ohjelmointirajapintoihin ja niiden hyödyntämiseen ja integrointiin mobiilisovellukseen. Käytännön työn aikana toteutettiin onnistuneesti lenkkeilysovellus, johon saatiin yhdistettyä Google Mapsin kartta ja käytettyä ohjelmointirajapintaa käyttäjän sijainnin hakuun.

Sovellusta varten luotiin API-avain Googlen Cloud -alustalla ja se otettiin käyttöön onnistuneesti. Lisäksi mobiilisovellukseen yhdistettiin Google Mapsin Geolocation API, jotta käyttäjän sijaintitietoja voitiin hakea. Google tarjoaa kattavan dokumentaation ohjelmointirajapintojensa käyttöön, mikä teki niiden integroinnista mobiilisovellukseen suhteellisen vaivatonta. Lopullisessa mobiilisovelluksessa ei kuitenkaan käytetty suoraan Google Mapsin ohjelmointirajapintaa. Sen sijaan Google Mapsin kartta integroitiin sovellukseen käyttämällä React Native Maps -kirjastoa, joka hyödyntää Google Maps SDK-työkalupakettia. Tämä osoittautui sovelluksen tarpeisiin myös hyväksi valinnaksi, koska se tarjosi runsaasti ominaisuuksia ja oli helppo ja vaivaton käyttää. Laitteen sijaintitiedot haettiin React Nativen Geolocation APIn avulla. Näin saavutettiin haluttu lopputulos, vaikka toteutustapa poikkesikin alkuperäisestä suunnitelmasta. Ohjelmointirajapintojen, kirjastojen ja SDK-työkalupakettien käyttö sovelluksen ohjelmoinnissa nopeutti sovelluksen kehitystä ja teki prosessista sujuvampaa. Ilman näitä resursseja olisi halutun lopputuloksen saavuttaminen ollut huomattavasti vaikeampaa.

Opinnäytetyön tavoitteena oli myös perehtyä ohjelmointiin ja mobiilisovelluskehitykseen sekä oppia, miten mobiilisovellus toteutetaan React Nativella. Tätä varten pyrittiin kehittämään toimiva versio lenkkeilysovelluksesta. React Native osoittautui erinomaiseksi kehykseksi mobiilisovelluskehitykseen ja opinnäytetyön suunnitteluvaiheessa ei muita vaihtoehtoja sovelluksen kehittämiseen edes pohdittu. Vaikka aiempaa kokemusta React Nativen käytöstä jo oli, uusia asioita oli silti paljon opittavana. React Nativesta löytyy runsaasti oppimateriaalia ja tietoa internetissä, ja sen käyttäjäyhteisö tarjoaa paljon tukea. Uusiin asioihin perehtyminen saattoi joskus olla aikaa vievää, mutta React Native on selkeä ja helposti ymmärrettävä kehys, mikä teki uusien konseptien oppimisesta ja sisäistämisestä helppoa.

Mobiilisovelluksen tietokantaratkaisuksi valittiin SQLite. Tämä päätös tehtiin jo ennen ohjelmoinnin aloittamista, koska myös tämän käytöstä oli aiempaa kokemusta ja sen tiedettiin soveltuvan hyvin yksinkertaisten sovellusten tarpeisiin. SQLite osoittautui täydelliseksi valinnaksi, erityisesti koska tietojen haluttiin tallentuvan vain laitteen sisäiseen

tietokantaan. SQLite on myös erittäin helppo käyttää ja sen integrointi sovellukseen sujui vaivattomasti.

Käytännön osuus tarjosi arvokasta oppia mobiilisovelluksen ohjelmoinnista ja sovelluskehityksessä huomioon otettavista seikoista. Sovelluksen kehittämisen aikana kävi ilmi, että mobiilikehityksessä on otettava huomioon monia sovelluksen toimintaan vaikuttavia tekijöitä eikä riitä, että pelkästään ohjelmoi itsenäisesti toimivan ohjelman. Sovellusta kehitettäessä eteen tulleita huomioon otettavia asioita olivat esimerkiksi, miten sovellus toimii mobiililaitteessa taustalla tai näytön ollessa sammutettuna. Lisäksi käyttäjältä pyydettävät luvat muun muassa laitteen sijaintitietojen käsittelyyn oli otettava huomioon sekä responsiivisuus ja puhelimen asetusten vaikutus sovelluksen toimintaan ja ulkoasuun.

Näitä haasteita ratkaistiin niiden ilmetessä kehitysprojektin ja testauksen aikana. Uskon kuitenkin, että tulevaisuudessa olisi hyödyllisempää ennakoida vastaavia ongelmia ja kehittää niihin ratkaisuja jo ennen sovelluksen varsinaista toteuttamista. Vaikka matkan varrella kohdattiinkin muutamia vaikeuksia, sovellus pystyttiin lopulta toteuttamaan suunnitelmien mukaisesti. Opinnäytetyön käytännön osuus oli erinomainen oppimiskokemus ja loistava tilaisuus syventää osaamista ohjelmoinnista ja mobiilisovelluskehityksestä. Työn aikana kertyi runsaasti tietoa React Nativesta, Android -ohjelmoinnista ja myös käyttöliittymäsuunnittelusta.

Lähteet

Alex Mullis (2017) Android Studio tutorial for beginners. Päivitetty 11.11.2017. Haettu 21.4.2024 osoitteesta https://www.androidauthority.com/android-studio-tutorial-beginners-637572/

Ana (2024a) What is an API Request Body? Päivitetty 16.1.2024. Haettu 7.4.2024 osoitteesta https://mixedanalytics.com/knowledge-base/api-headers-explained/

Ana (2024b) What is an API Request Body? Päivitetty 16.1.2024. Haettu 7.4.2024 osoitteesta https://mixedanalytics.com/knowledge-base/request-bodies-explained/

Anastasiia Lastovetska (2021). Why to integrate Uber API: Benefits and successful cases. Päivitetty 12.11.2021. Haettu 31.3.2024 osoitteesta https://mlsdev.com/blog/uber-api

Britannica (2024) Google. Päivitetty 26.7.2024. Haettu 28.7.2024 osoitteesta https://www.britannica.com/money/Google-Inc

Google (2024a) Google Maps Platform FAQ. Päivitetty 17.4.2024. Haettu 17.4.2024 osoitteesta https://developers.google.com/maps/faq

Google (2024b) Google Maps Platform, Maps Static API, Overview. Päivitetty 17.4.2024. Haettu 19.4.20224 osoitteesta https://developers.google.com/maps/documentation/maps-static/overview

Google (2024c) Google Maps Platform, Maps Static API, Get Started. Päivitetty 17.4.2024. Haettu 19.4.2024 osoitteesta https://developers.google.com/maps/documentation/maps-static/start

Google (2024d) Google Maps Platform, Maps Embed API, The Maps Embed API overview. Päivitetty 17.4.2024. Haettu 19.4.2024 osoitteesta https://developers.google.com/maps/documentation/embed/get-started

Google (2024e) Google Maps Platform, Places API, Overview. Päivitetty 17.4.2024. Haettu 19.4.2024 osoitteesta https://developers.google.com/maps/documentation/places/web-service/overview

Google (2024f) Google Maps Platform, Geocoding API, Get Started. Päivitetty 17.4.2024. Haettu 19.4.2024 osoitteesta https://developers.google.com/maps/documentation/geocoding/start

Google (2024g) Google Maps Platform, Geolocation API, Geolocation API Overview. Päivitetty 17.4.2024. Haettu 19.4.2024 osoitteesta https://developers.google.com/maps/documentation/geolocation/overview

Google (2024h) Google Maps Platform, JavaScript API, Overview. Päivitetty 17.4.2024. Haettu 20.4.2024 osoitteesta https://developers.google.com/maps/documentation/javascript/overview

Google (2024i) Google Maps Platform, Routes API, Routes API Overview. Päivitetty 17.4.2024. Haettu 20.4.2024 osoitteesta https://developers.google.com/maps/documentation/routes/overview

Google (2024j) Google Maps Platform, Roads API, Roads API Overview. Päivitetty 17.4.2024. Haettu 20.4.2024 osoitteesta https://developers.google.com/maps/documentation/roads/overview

Google (2024k) Google Maps Platform, Directions API, Directions API Overview. Päivitetty 17.4.2024. Haettu 20.4.2024 osoitteesta https://developers.google.com/maps/documentation/directions/overview

Google (2024l) Google Maps Platform, Solar API, Solar API Overview. Päivitetty 17.4.2024. Haettu 20.4.2024 osoitteesta https://developers.google.com/maps/documentation/solar/overview

Google (2024m) Google Maps Platform, Air Quality API, Air Quality API Overview. Päivitetty 17.4.2024. Haettu 20.4.2024 osoitteesta https://developers.google.com/maps/documentation/air-quality/overview

Google (2024n) Google Maps Platform, Pollen API, Pollen API Overview. Päivitetty 17.4.2024. Haettu 20.4.2024 osoitteesta https://developers.google.com/maps/documentation/pollen/overview

Google (2024o) Google Maps Platform, iOS, Maps SDK for iOS, Overview. Päivitetty 17.4.2024. Haettu 20.4.2024 osoitteesta https://developers.google.com/maps/documentation/ios-sdk/overview

Google (2024p) Google Maps Platform, Android, Maps SDK for Android, Maps SDK for Android Overview. Päivitetty 17.4.2024. Haettu 20.4.2024 osoitteesta https://developers.google.com/maps/documentation/android-sdk/overview

Google (n.d.a) Google API Client Libraries. Haettu 17.4.2024 osoitteesta https://developers.google.com/api-client-library/

Google (n.d.b) Pricing that scales to fit your needs. Haettu 19.4.2024 osoitteesta https://mapsplatform.google.com/pricing/

Javatpoint (n.d) What is Google Map and how do you use it? Haettu 13.4.2024 osoitteesta https://www.javatpoint.com/what-is-google-map-and-how-do-you-use-it

Junaid Baig (2023). A Complete Guide to the Different Types of APIs. Päivitetty 21.12.2023. Haettu 6.4.2024 osoitteesta https://www.astera.com/type/blog/types-of-apis/

Kate Mikula (2023). The History and Evolution of APIs. Päivitetty 7.2.2023. Haettu 23.4. osoitteesta https://traefik.io/blog/the-history-and-evolution-of-apis/

Max Tsurbeliov (2021). How does React Native work? Päivitetty 8.8.2021. Haettu 3.8. osoitteesta https://www.akveo.com/blog/how-does-react-native-work

MDN Web Docs (2023a). An overview of HTTP. Päivitetty 16.12.2023. Haettu 27.4.2024 osoitteesta https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

MDN Web Docs (2023b). Response header. Päivitetty 8.6.2023. Haettu 27.4.2024 osoitteesta https://developer.mozilla.org/en-US/docs/Glossary/Response_header

MDN Web Docs (2023c). Request header. Päivitetty 17.5.2023. Haettu 28.7.2024 osoitteesta https://developer.mozilla.org/en-US/docs/Glossary/Request_header

Mlytics (n.d.). What is an API? Haettu 31.3.2024 osoitteesta https://learning.mlytics.com/the-internet/what-is-an-api/

Netguru (n.d.) What is React Native? Complex guide for 2023. Haettu 21.4.2024 osoitteesta https://www.netguru.com/glossary/react-native

Nylas (n.d.) Types of APIs. Haettu 25.4.2024 osoitteesta https://www.nylas.com/api-guide/

React Native (2024) React Native. Päivitetty 22.4.2024. Haettu 3.8.2024 osoitteesta https://reactnative.dev/docs/getting-started

Restapitutorial.com (n.d.) HTTP Status Codes. Haettu 7.4.2024 osoitteesta https://www.restapitutorial.com/httpstatuscodes.html

SQLite (2023) About SQLite. Päivitetty 10.10.2023. Haettu 21.4.2024 osoitteesta https://www.sqlite.org/about.html

Visual Studio Code (2024) Setting up Visual Studio Code. Päivitetty 4.4.2024. Haettu 21.4.2024 osoitteesta https://code.visualstudio.com/docs/setup/setup-overview

Wayback machine (n.d.) Google Company. Our history in depth. Haettu 13.4.2024 osoitteesta https://web.archive.org/web/20160406123606/http://www.google.co.uk/about/company/history/

Liite 1: Aineistonhallintasuunnitelma

Kehitysprojekti:

Kehitysprojektin aikana pidetään päiväkirjaa, johon kerätään teknistä tietoa projektista. Tämä tieto analysoidaan opinnäytetyötä varten. Päiväkirjaa säilytetään tekijän tietokoneen C-asemalla, ja siitä tehdään säännöllisesti varmuuskopioita HAMKin OneDriveen. Päiväkirjaa säilytetään C-asemalla ainakin vuoden verran opinnäytetyön valmistumisesta. Työ ei sisällä arkaluonteisia tietoja, jotka vaativat toimenpiteitä suojaamisen tai tietoturvan ja tietosuojan kanssa.

Opinnäytetyöaineiston jatkokäyttö työn valmistumisen jälkeen

En halua hyödyntää tai antaa tutkimusaineistoani jatkokäyttöön.

Tutkimusaineistoa ei jatkokäytetä. Opinnäytetyön tekijä säilyttää aineiston tietoturvallisesti vuoden ajan opinnäytetyön hyväksymispäivästä, jotta opinnäytetyön tulokset voidaan tarvittaessa varmistaa ja hävittää tämän jälkeen aineiston tietoturvallisesti.

Liite 2: Linkki sovelluksen GitHub -repositoryyn

https://github.com/hsaros81/RunningApp/tree/master