HAMK
Hämeen ammatti-
korkeakoulu

# ALUSTARIIPPUMATON MOBIILIKEHITYS

Ammattikorkeakoulututkinnon opinnäytetyö

Tieto- ja viestintätekniikka, insinööri (AMK)

Kevät 2024

Ilmari Ketola

HAMK
Hämeen ammatti-
korkeakoulu

Tieto- ja viestintätekniikka
Tekijä Ilmari Ketola
Työn nimi Alustariippumaton mobiilikehitys
Ohjaaja Toni Laitinen

Tiivistelmä
Vuosi 2024

---

Opinnäytetyössä tutustuttiin alustariippumattomaan mobiilikehitykseen Ionic, Flutter ja React Native -ohjelmistokehyksillä. Työssä käsiteltiin natiivikehityksen ja alustariippumattoman kehityksen toimintatavat, ja verrattiin natiivia sovelluskehitystä alustariippumattomaan sovelluskehitykseen.

Opinnäytetyö keskittyy Android- ja iOS-käyttöjärjestelmille suuntautuvaan mobiilikehitykseen, sillä ne hallitsevat mobiilikäyttöjärjestelmämarkkinaa 99 %:n osuudella (Statcounter, 2024).

Opinnäytetyön toiminnallisessa osuudessa toteutettiin laitteen paikannusominaisuutta hyödyntävä mobiilisovellus säätietojen hakuun, käyttäen työhön valittuja alustariippumattomia ohjelmistokehyksiä. Vaikka kehysten välillä löytyi eroja, jokainen niistä soveltui hyvin yksinkertaisen sääsovelluksen alustaksi.

Opinnäytetyössä tutkittiin myös sovelluksen julkaisua ja sen eri kanavia ja sovelluskauppojen vaatimuksia sovelluksille. Havaittiin, että sovelluksen saaminen sovelluskaupan jakeluun on monivaiheinen prosessi.

Avainsanat cross-platform, Flutter, Ionic, React Native
Sivut 25

HAMK
Hämeen ammatti-
korkeakoulu

Information and Communication Technology
Author Ilmari Ketola
Subject Cross-platform mobile development
Supervisors Toni Laitinen

Abstract
Year 2024

The thesis covers cross platform mobile development with Ionic, Flutter and React Native frameworks. Thesis discussed the development methods for developing native apps in Android and iOS -platforms and compared native app development to cross platform development.

The thesis focuses on developing apps to Android and iOS platforms, because their market share in mobile operating systems is 99% (Statcounter, 2024).

In the functional part of the thesis, a weather app was implemented using the selected cross platform frameworks. Even though differences were found, each of them was suitable framework for a simple weather app.

The thesis covered app distribution and its different distribution platforms. It was found that getting app for app marketplace is a multi-step process.

Keywords cross-platform, Flutter, Ionic, React Native
Pages 25

# Sisällys

## Kuvat, taulukot ja kaavat

# 1 Johdanto

Älylaitteita ja mobiilisovelluksia käytetään tänä päivänä enemmän kuin koskaan. Vuonna 2023 90 %:lla suomalaisista on ollut älypuhelin omassa käytössä (Tilastokeskus, n.d.). Mobiilisovellus tarjoaa yrityksille useita liiketoimintaa tukevia mahdollisuuksia, kuten brändin vahvistamisen, käyttäjien tavoittamisen push-ilmoituksilla ja lisäarvon tarjoamisen yrityksen asiakkaille, esimerkiksi tilauskanavan muodossa. Vaikka mobiilisovelluksista tutut toiminnallisuudet on mahdollista toteuttaa teknisesti verkkosivuna, verkkosivun käyttökokemus harvoin yltää mobiilisovelluksen tasolle (Adobe, 2023).

Mobiilisovellus halutaan lähes poikkeuksetta saataville niin Android- kuin iOS-käyttöjärjestelmälle. Perinteinen tapa toteuttaa mobiilisovellus on tehdä se natiivisovelluksena, joka tarkoittaa, että jokaiselle tuetulle alustalle kehitetään oma sovellus. Usean sovelluksen kehitys ja ylläpito on työlästä sekä kallista verrattuna yhteen sovellukseen.

Vaihtoehtona natiivisovelluskehitykselle on alustariippumaton mobiilikehitys, joka tarkoittaa, että yhdestä ohjelmakoodista saadaan tehtyä niin Android- kuin iOS-sovellus. Alustariippumaton mobiilikehitys aiheuttaa kuitenkin tiettyjä rajoitteita ja huomioon otettavia asioita ohjelmistokehityksen näkökulmasta.

Opinnäytetyössä käsitellään niin natiivia kuin alustariippumatonta kehitystä, selvitetään alustariippumattoman kehityksen hyötyjä ja haasteita verrattuna natiiviin kehitykseen. Lopuksi opinnäytetyössä käsitellään mobiilisovelluksen jakelua.

# 2 Natiivi mobiilikehitys

## 2.1 Android ja iOS -alustojen historia

Kesäkuussa 2007 Apple julkaisi iOS-käyttöjärjestelmän ensimmäisen version, joka kantoi nimeä iPhone OS 1 (Macworld, 2024). Applen visio ensimmäisen mobiilikäyttöjärjestelmäversion kohdalla oli se, että ulkopuolisten sovellusten asentamista ei sallita, vaan kaiken sisällön on oltava selaimessa käytettävää (9to5Mac, 2011). Suunnitelma muuttui kesäkuussa 2008 julkaistussa iPhone OS 2 -versiossa, joka sisälsi App Store -sovelluskaupan ja sitä kautta tuen ulkopuolisille sovellukselle (Apple, 2008). Tätä hetkeä voidaan pitää alkuna natiivisovellusten kehitykselle iOS-alustalle.

Android-käyttöjärjestelmän ensimmäinen versio koki päivänvalon syyskuussa 2008. Se sisälsi Android Market -sovelluskaupan sovellusten lataamista varten (Android Developers Blog, 2008).

## 2.2 Natiivikehitys Android ja iOS -alustoilla

Natiivia Android-kehitystä tehdään ohjelmointiympäristön (IDE) avulla. Käytetyin ja virallinen ohjelmointiympäristö Androidille on Android Studio. Android Studio sisältää työkalut käyttöliittymien luomiseen, koodieditorin, ajo- ja julkaisuympäristöt. Natiivit Android-sovellukset kirjoitetaan Kotlin tai Java -ohjelmointikielellä (Android Developers, n.d.-c).

Natiivia iOS-kehitystä tehdään Xcode-ohjelmalla. Xcode on saatavilla vain Mac-tietokoneille, joten natiivi iOS-kehitys vaatii Mac-tietokoneen. Objective C oli vuoteen 2014 asti iOS-kehityksessä käytetty ohjelmointikieli. Vuonna 2014 Apple julkaisi Swift-ohjelmointikielen, joka on pääosin tänä päivänä käytetty ohjelmointikieli iOS-kehityksessä.

# 3 Alustariippumattomat kehykset

## 3.1 React Native

React Native sai alkunsa Metan (ent. Facebook) sisäisestä tarpeesta. Tavoitteena oli pienentää sovelluskehityksen kustannuksia ja saada sovelluksen käyttökokemus yhdenmukaiseksi eri alustojen välillä (TechAhead, 2023).

React Native -sovelluksen koodi kirjoitetaan JavaScript- tai TypeScript-ohjelmointikielellä. TypeScript-kielen käyttö on suositeltavaa, sillä se mahdollistaa staattisen tyypityksen. Uudet projektit, jotka luodaan React Nativen komentorivityökalua käyttävät oletuksena TypeScript-kieltä (Meta Open Source, n.d.-c).

Käyttöliittymät luodaan käyttäen JSX-merkintätapaa. JSX mahdollistaa HTML-merkintäkielen kaltaisen elementtimerkitsemisen JavaScript-koodin sisällä (Meta Open Source, n.d.-d).

React Native -sovellusprojekti voidaan luoda kahdella eri tavalla. Perinteinen tapa on luoda projekti React Native CLI -komentorivityökalulla, jolloin projektiin ei lisätä Expo-kirjastoa, ja sovellus voi hyödyntää kaikkia React Nativen ominaisuuksia ja asetuksia.

Toinen tapa on käyttää Expo Go:ta, joka on suositeltava tapa React Nativeen tutustuttaessa. Expo Go:lla projektin luonnissa on vähemmän eri vaiheita. Expon käyttäminen antaa myös pääsyn valmiiksi luotuihin Expo SDK -kirjastoihin. Ne mahdollistavat esimerkiksi seuraavia toiminnallisuuksia:

- expo-av – äänen toisto ja tallennus
- expo-camera – kamera
- expo-location – sijainti

Expo Go:lla luodun projektin rajoitteena on rajoitetumpi tuki ulkopuolisille kirjastoille. Mikäli kirjasto hyödyntää React Nativen Native Modules -rajapintaa, sitä ei voida asentaa Expolla asennettuun projektiin. Toisaalta nykyään moni kirjasto on asennettavissa Expo-projektiin ongelmitta.

React Nativen mukana toimitetaan tärkeimmät komponentit käyttöliittymien luomiseen (Meta Open Source, n.d.-b).

- View – tärkein peruskomponentti ja käytännössä jokaisen näkymän pohjalla, komponentti toimii kääreenä muille komponenteille
- Button – painike
- Text – teksti
- TextInput – tekstikenttä

Näiden komponenttien avulla voidaan tehdä jopa kokonainen sovellus. Komponenttien tyylit määritellään style-ominaisuudella. React Nativen mukana tuleva komponenttivalikoima on suppein opinnäytetyössä käsitellyistä alustariippumattomista kehyksistä.

React Native -sovelluksen toiminnallisuuksia voidaan laajentaa ulkopuolisilla kirjastoilla. Kirjastot asennetaan npm- tai yarn-paketinhallintatyökalulla.

Kehitettävän sovelluksen vaatimuksena voi olla esimerkiksi valintaruutu. Valintaruutu-komponenttia ei toimiteta React Nativen mukana, joten pakettienhallinnasta tulee etsiä sopiva komponentti tai vaihtoehtoisesti luoda sellainen itse. Komponentti @react-native-community/checkbox tarjoaa halutun toiminnallisuuden ja se voidaan asentaa yarn-pakettienhallintatyökalulla komennolla yarn add @react-native-community/checkbox.

## 3.2 Flutter

Flutter on Googlen kehittämä ohjelmistokehys alustariippumattomien sovellusten kehitykseen. Flutterin varhainen alpha-versio esiteltiin ensimmäisen kerran Dart-ohjelmointikielen kehittäjätapahtumassa. Tavoitteena oli luoda tapa suorituskykyisten Android-sovellusten kehitykseen ilman Java-kieltä. Suorituskyvyn mittariksi määritettiin ruudunpäivitysnopeus, ja tavoitteeksi oli saada yksittäisen ruudun renderöintiaika alle kahdeksaan millisekuntiin, jotta 120 ruutua sekunnissa -päivitysnopeuteen päästäisiin. Ensimmäisessä demossa tavoitteeseen päästiin 1,2 millisekunnin ruudun renderöintiajalla (Ars Technica, 2015).

Flutter-sovelluksia kehitetään Dart-ohjelmointikielellä (Dart n.d.). Käyttöliittymät rakennetaan Flutter Widgeteillä (Flutter n.d.-a).

Flutterin peruskomponentteihin kuuluvat seuraavat (Flutter n.d.-a).

- Text – teksti
- Row, Column – rivi ja sarake, toimivat widgettien asettelun pohjana
- Stack – pino, mahdollistaa widgettien asettelun päällekkäin
- Container – säiliö, tärkein peruskomponentti ja käytännössä jokaisen näkymän pohjalla, komponentti toimii kääreenä muille komponenteille

Flutter tarjoaa laajan kirjaston komponentteja MaterialApp-widgetin kautta. Vaikka MaterialApp-widgetin käyttö ei ole edellytys Flutter-sovelluksen kehitykselle, sen käyttö on kuitenkin erittäin suositeltavaa, sillä Flutterin komponenttivalikoima on suppea, esimerkiksi lähes jokaisessa sovelluksessa tarvittava painike (Button) puuttuu siitä.

MaterialApp-widgetin kautta saatavia yleisiä komponentteja ovat esimerkiksi:

- Button – painike
- TextField – tekstikenttä

Flutteriin on mahdollista lisätä ulkopuolisia kirjastoja pub.dev -palvelusta. Lisättävä paketti lisätään projektin pubspec.yaml dependencies-kohdan alle jonka jälkeen projektin riippuvuudet ladataan ajamalla komento flutter pub get. Komento lukee pubspec.yaml -tiedoston ja asentaa projektin tarvitsemat riippuvuudet.

## 3.3 Ionic

Ionic on Driftyn kehittämä ohjelmistokehys alustariippumattomien sovellusten kehitykseen. Ionic mahdollistaa sovelluskehityksen web-kehityksestä tutuilla tekniikoilla, kuten HTML, CSS ja JavaScript. Ionic-sovelluksen koodi voi olla Angular, React, Vue -kehyksellä tehtyä tai pelkkää JavaScript-koodia. Ionic tarjoaa komponenttikirjastot Angularille, Reactille ja Vuelle (Ionic, n.d.-a).

Ionic tarjoaa laajan valikoiman erilaisia komponentteja UI Components -nimetyn komponenttikirjaston kautta (Ionic, n.d.-b):

- ion-text – teksti
- ion-input – tekstin syöttö
- ion-searchbar – hakukenttä

Peruskomponenttien lisäksi saatavilla on useita valmiiksi käyttöönotettavia komponentteja, kuten hakukenttä.

# 4 Alustariippumattomien kehysten ohjelmointikielet

## 4.1 JavaScript ja TypeScript

Ionic ja React Native -sovellukset kirjoitetaan JavaScript tai TypeScript-ohjelmointikielellä. JavaScript on dynaamisesti tyypitetty ohjelmointikieli. Dynaamisesti tyypitetyssä ohjelmointikielessä muuttujan tyyppi päätellään ajon aikana sen arvoon perustuen. Dynaamisen tyypityksen hyvänä puolena pidetään sen joustavuutta, sillä muuttujan tyyppiä ei tarvitse määritellä etukäteen, ja muuttujan tyyppi voi vaihtua suorituksen aikana.

```
let variableTypeCanChange = null;
variableTypeCanChange = 1;
variableTypeCanChange = 'one';
variableTypeCanChange = new Date();
```

Kuva 1. Dynaaminen tyypitys JavaScript-ohjelmointikielessä

Kuvan 1 koodissa muuttuja alustetaan null-arvolla. Tämän jälkeen muuttujalle asetetaan numeromuotoinen arvo, jonka jälkeen sille määritellään merkkijonomuotoinen arvo ja lopuksi

arvoksi asetetaan Date-tyyppinen objekti. Koodi ei sinällään ei ole kovin hyödyllinen, lähinnä se antaa esimerkin siitä, miten dynaaminen tyypitys toimii.

Dynaamisen tyypityksen hyvänä puolena pidetään joustavuutta. Toisaalta dynaamista tyypitystä käytettäessä virheiden mahdollisuus koodissa kasvaa. Tyypityksestä johtuvat virheet havaitaan vasta suoritusvaiheessa, aina ei silloinkaan (Solwey Consulting, 2023).

TypeScript on JavaScriptin päälle rakennettu ohjelmointikieli, joka lisää staattisen tyypityksen JavaScriptiin.

```
let variableTypeIsStaticInTypescript = 1;
variableTypeIsStaticInTypescript = 'one';    Type 'string' is not assignable to type 'number'.
variableTypeIsStaticInTypescript = new Date();    Type 'Date' is not assignable to type 'number'.
```

Kuva 2. Staattinen tyypitys TypeScript-ohjelmointikielessä

Staattisen tyypityksen takia kuvan 1 JavaScript-koodi ei toimi TypeScript-koodina. TypeScript-tulkin palauttama virhe kertoo, että string-tyyppistä arvoa ei voida asettaa muuttujalle, jonka tyyppi on numero. Tämä johtuu siitä, että TypeScript-kielessä muuttujan tyyppi on pysyvä (staattinen).

On tärkeä huomioida, että TypeScript on tyypitettyä vain kehitysvaiheessa. Ajovaiheessa TypeScript-koodi käännetään JavaScriptiksi, jolloin ajettava sovellus tiedä määritetyistä tyypeistä mitään, eikä anna virhettä, mikäli ajon aikana tapahtuu tyypityksestä johtuva virhe. Toisin sanoen TypeScriptissä ei ole ajonaikaista tyypitystä.

TypeScript-projekteissa kääntäminen JavaScriptiksi on lähes poikkeuksetta määritelty tapahtumaan automaattisesti, kun projekti ajetaan. Kääntäminen on mahdollista tehdä manuaalisesti tsc-komentorivityökalulla.

Varsinkin isommissa projekteissa TypeScriptin käyttö on suositeltavaa, sillä se tekee koodista luettavampaa ja virheiden mahdollisuus pienenee, kun muuttujan tyyppi ei voi muuttua lennosta. Tyyppien määrittely tuo kuitenkin lisää mietittävää ohjelmoijalle, joten ihan pienimmissä tai elinkaareltaan lyhyissä projekteissa JavaScript voi olla parempi valinta.

## 4.2 Dart

Flutter-sovellukset kirjoitetaan Dart-ohjelmointikielellä. TypeScriptin tavoin Dart on staattisesti tyypitetty, jonka lisäksi käytössä on ajonaikainen tyypitys. Mikäli ajon aikana tapahtuu

tyypitysvirhe, ohjelman suoritus pysähtyy virheeseen. TypeScriptin tavoin kuvassa 1 ei toimi Dart-ohjelmointikielellä.

```
var variableTypeIsStaticInDart = 1;
variableTypeIsStaticInDart = 'one';    A value of type 'String' can't be assigned to a variable of type 'int'.Try changi
variableTypeIsStaticInDart = DateTime.now();    A value of type 'DateTime' can't be assigned to a variable of type 'int'.
```

Kuva 3. Staattinen tyypitys Dart-ohjelmointikielessä

Dart-kielessä on Null Safety -ominaisuus, joka estää tahattoman null-arvon käytön muuttujan arvona (Dart, n.d.-b). Esimerkiksi tilanteessa, jossa funktio odottaa argumentiksi kokonaislukua mutta saa argumentiksi null-arvon, sovelluksen suoritus pysähtyy ajonaikaisen virheen.

Dart-ohjelmointikielen aiemmissa versioissa null safety -ominaisuuden on voinut konfiguroida pois käytöstä, mutta versiosta 2.12 alkaen ominaisuutta ei voi ottaa pois päältä. Null Safety -ominaisuus vaatii jokaisen muuttujan sisältävän arvon, ellei muuttujan tyyppiä ole määritelty erikseen nullable-tyyppiseksi.

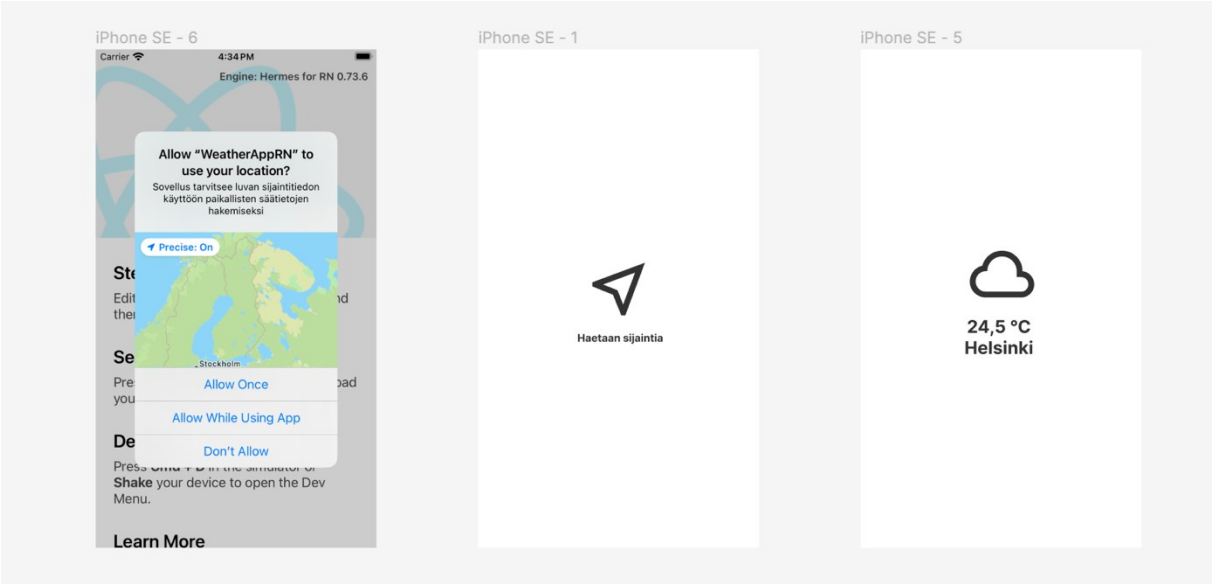
```
String name = "test";
name = null;    A value of type 'Null' can't be assigned to a variable of type 'String'.

String? nameNullable = "test";
nameNullable = null;
```

Kuva 4. Null Safety Dart-ohjelmointikielessä

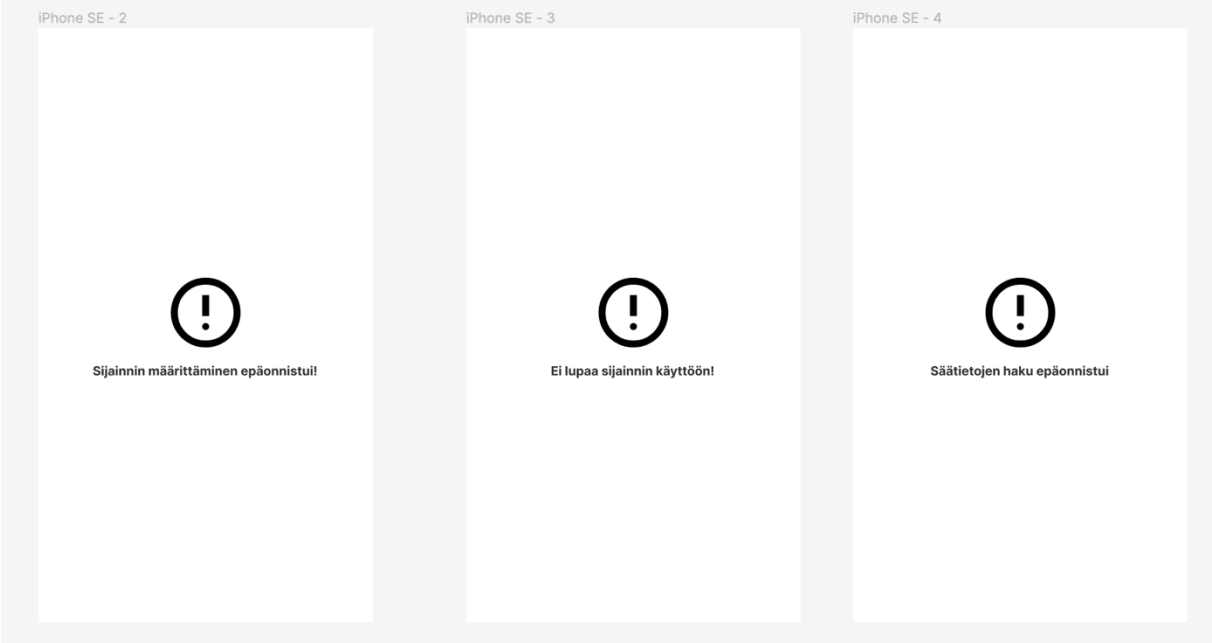
# 5 Esimerkkiprojekti: Sääsovellus

Sääsovellus on yksinkertainen mobiililaitteen ominaisuuksia hyödyntävä mobiilisovellus. Sääsovelluksesta tehtiin toteutus React Nativella, Flutterilla ja Ionicilla. Sääsovellus toimii seuraavasti:

1. Kun sovellus avataan, sovellus tarkistaa, onko sovelluksella lupa paikannuksen käyttöön. Mikäli lupaa ei ole, se pyydetään. Sovellus ei toimi ilman paikannuksen lupaa.
2. Mikäli käyttäjä ei anna lupaa paikannuksen käyttöön tai laite ei tue paikannusta, näytetään virhe.
3. Sovellus hakee laitteen sijainnin ja lähettää sen REST-rajapinnalle.
4. REST-rajapinta palauttaa sovellukselle käyttäjän koordinaattien perusteella määritetyn kaupungin sekä lämpötilan.
5. Mikäli verkkopyyntö pyyntö REST-rajapinnalle epäonnistuu, näytetään virhe.




Kuva 5. Sääsovelluksen toiminta normaalitilanteessa




Kuva 6. Sääsovelluksen mahdolliset virhetilanteet

## 5.1 Sijainnin käyttö mobiilisovelluksissa

Mobiililaitteen sijaintitiedon hyödyntäminen mobiilisovelluksessa vaatii luvan pyytämistä käyttäjältä. Näin varmistetaan käyttäjän tietosuoja ja mahdollisuus valita mitkä sovellukset saavat käyttää laitteen sijaintitietoa.

iOS-alustalla lupa kysytään seuraavasti (Zoontek, n.d.):

1. Tukeeko käytetty laite pyydettyä lupaa? Esimerkiksi iPad-mallit ilman mobiiliverkkoyhteyksiä eivät sisällä GPS-vastaanotinta, eivätkä sen takia pysty määrittämään tarkkaa sijaintia.
2. Onko lupapyyntö mahdollista suorittaa? Laite voi olla rajoitusten alainen, jolloin lupaa ei voida pyytää.
3. Lupapyyntö esitetään. Mikäli käyttäjä hyväksyy, pyydetty ominaisuus on sovelluksen käytössä.

Androidilla lupa kysytään seuraavasti:

1. Tukeeko käytetty laite pyydettyä lupaa?
2. Onko lupa aiemmin jo annettu?
3. Lupapyyntö esitetään. Mikäli käyttäjä hyväksyy pyynnön, pyydetty ominaisuus on käytössä.

Jotta sovellus voi pyytää lupaa sijainnin käyttöön, sovelluksen metatietoihin tulee merkitä mahdollisuus sijainnin käyttöön. Androidilla merkintä tehdään AndroidManifest.xml-tiedostoon ja iOSilla Info.plist-tiedostoon.

Lupa kysytään käyttäen käyttöjärjestelmän natiivirajapintaa. Natiivisovelluksessa sijaintiluvan pyytäminen onnistuisi suoraan ohjelmakoodista. Alustariippumatonta kehystä käytettäessä natiivia koodia ei voida kutsua suoraan, vaan natiivin koodin kutsumiseksi tulee käyttää erityistä siltaa, joka mahdollistaa natiivikoodin kutsumisen alustariippumattoman kehyksen ohjelmakoodista. Yleisimpien natiivirajapintojen käyttöön on olemassa valmis kirjasto.

Sääsovellus käyttää sijaintitiedon ja luvan kysymiseen kirjastoa @react-native-community/geolocation React Native -toteutuksessa, geolocator-pakettia Flutter-toteutuksessa ja @capacitor/geolocation -pakettia Ionic-toteutuksessa. Kyseisissä paketeissa on valmiit funktiot sijainnin luvan ja sijaintitiedon hakuun, joten natiivia koodia ei tarvitse itse kirjoittaa.

## 5.2 Sijainnin luvan pyytäminen ja sijainnin haku

### 5.2.1 React Native

Sääsovelluksen React Native -versiossa lupa sijainnin käyttöön ja itse sijaintiedon haku toteutetaan @react-native-community/geolocation -paketilla. Lupa kysytään requestAuthorization-funktiolla. Funktiolle annetaan parametrina kaksi callback-funktiota, ensimmäinen onnistuneen lupapyynnön käsittelyyn ja toinen epäonnistuneet lupapyynnön käsittelyyn. Callback-funktion koodi ajetaan, kun lupapyyntö on valmistunut onnistuneesti, eli käytännössä kun käyttäjä on hyväksynyt lupapyynnön.

Kun requestAuthorization-funktio on saatu ajettua ja lupa sijainnin hakuun saatu, kutsutaan getCurrentPosition-funktiota, jolle määritellään onnistuneen sijainnin haun callback-funktio ja virhetilanteen callback-funktio. Jos sijainti saadaan haettua onnistuneesti, suoritetaan verkkopyyntö ja sen jälkeen näytetään käyttäjälle sää. Jos sijainnin haku epäonnistuu, näytetään käyttäjälle virhenäkymä.

### 5.2.2 Flutter

Geolocator-paketin Geolocator-luokka sisältää asynkroniset funktiot checkPermission ja getCurrentPosition. Asynkroninen funktio tarkoittaa sitä, että funktion tulos ei ole heti saatavilla (Flutter, n.d.-c). Await-operattorilla asynkroninen funktio jää odottamaan kutsuttavan asynkronisen funktion vastausta.

```
Future<bool> isPermissionGranted() async {
  LocationPermission permission = await Geolocator.requestPermission();
  if (permission == LocationPermission.whileInUse) {
    return true;
  } else {
    return false;
  }
}
```

Kuva 7. Flutter-sovelluksen funktio sijainnin luvan kysymiseen

Asynkroninen funktio kutsuu Geolocator-luokan requestPermission-funktiota ja palauttaa arvon tosi, mikäli lupa annetaan ja arvon epätosi mikäli lupaa ei anneta.

Lupapyynnön jälkeen sijainti haetaan Geolocator-luokan getCurrentPosition-funktiolla. Funktiota tulee kutsua try-catch-rakenteen sisällä, jotta virhetilanteessa tarvittava koodi

saadaan suoritettua. Onnistunut getCurrentPosition-funktion suoritus palauttaa Position-luokan, joka sisältää säätietojen hakuun tarvittavat koordinaatit.

### 5.2.3 Ionic

@capacitor/geolocation -paketti sisältää asynkroniset funktiot requestPermissions ja getCurrentPosition. Funktioita kutsutaan asynkronisessa funktiossa await-opraattoria hyödyntäen, kuten Flutter-toteutuksessa.

## 5.3 Verkkopyynnöt

### 5.3.1 Sääsovelluksen verkkopyynnöt

Sovelluksessa näytettävät tiedot, lämpötila ja paikkakunta haetaan verkkopyynnöllä. Verkkopyyntö tehdään osoitteeseen:

https://{{url}}/weatherForLocation?latitude={{y}}&longitude={{x}}

{{url}}-parametrissa määritetään rajapinnan verkko-osoite. {{y}}-parametrissa leveysaste ja {{x}}-parametrissa pituusaste.


```
{
  city: "Helsinki",
  temperature: 24.12
}
```


Kuva 8. Sääsovelluksen käyttämän rajapinnan palautus JSON-muodossa

### 5.3.2 Verkkopyynnöt Android ja iOS -alustoilla

Verkkopyyntöjen suorittamiseen Androidilla tarvitaan merkintä AndroidManifest.xml-tiedostoon. Sovelluksen ei kuitenkaan tarvitse pyytää lupaa verkon käyttöön, kuten sijainnin kanssa (Android Developers, n.d.-b).

iOS-alustalla verkkopyyntöjen suorittamiseen ei tarvita lupaa erikseen, vaan kaikki sovellukset voivat tehdä verkkopyyntöjä. Poikkeuksen tästä muodostaa lähiverkossa tehtävät verkkopyynnöt, joihin tarvitaan lupa käyttäjältä (Apple, 2023).

Verkkopyyntö mobiilisovelluksessa on suositeltavaa toteuttaa https-protokollaa hyödyntäen. Https-protokollaa käytettäessä asiakkaan (tässä tapauksessa sääsovellus), ja palvelimen välinen liikenne on salattua. Salaamatonta http-protokollaa käyttäen näin ei ole, ja kaikki tahot, jotka pääsevät tarkkailemaan liikennettä, näkevät myös mitä tietoja asiakkaan ja palvelimen välillä siirretään (Cloudflare, n.d.).

iOS 9 -versiosta alkaen App Transport Security (ATS) -ominaisuus estää oletusarvioisesti http-verkkopyynnöt (Apple Developer, n.d.-e). Android-käyttöjärjestelmässä vastaava ominaisuus on Network security configuration, joka estää oletusarvoisesti http-verkkopyynnöt Android-versiosta 9 alkaen. Vaikka ATS ja Network security configuration -ominaisuudet on mahdollista konfiguroida sallimaan http-liikenne tiettyyn verkkotunnukseen, on ensisijainen toimintatapa selvittää, olisiko sovelluksen käyttämä palvelin mahdollista konfiguroida https-pyyntöjä tukevaksi (Apple Developer, n.d.-f).

### 5.3.3 Verkkopyyntö Flutterilla

Suositeltava tapa verkkopyynnön tekoon Flutterilla on http-kirjasto (Flutter, n.d.-b).

```
Future<http.Response> getWeatherData(latitude: double, longitude: double) {
  String weatherUrl = getWeatherUrl(latitude, longitude);
  return http.get(Uri.parse(weatherUrl));
}

String getWeatherUrl(double latitude, double longitude) {
  return 'https://domain.fi/weatherForLocation?latitude=$latitude&longitude=$longitude';
}
```

Kuva 9. Flutter-sovelluksen funktiot verkkopyynnön valmisteluun rajapinnalle

Funktio getWeatherData palauttaa Future http.Responsen, joka sisältää verkkopyynnön vastauksen. Funktio getWeatherUrl muodostaa url-osoitteen.

On kuitenkin epäkäytännöllistä käsitellä http.Response -luokkaa sellaisenaan sovelluskoodissa. Suositeltava tapa on luoda luokka, joka määritellään rajapinnan onnistunutta vastausta vastaavaksi (Flutter, n.d.-b).

```
class WeatherResponse {
  final String city;
  final double temperature;

  const WeatherResponse({
    required this.city,
    required this.temperature,
  });

  factory WeatherResponse.fromJson(Map<String, dynamic> json) {
    return switch (json) {
      {
        'city': String city,
        'temperature': double temperature,
      } =>
        WeatherResponse(
          city: city,
          temperature: temperature
          ),
      _ => throw const FormatException('Failed to load WeatherResponse.'),
    };
  }
}
```

Kuva 10. Flutter-sovelluksen WeatherResponse-luokka

Luokassa on kaksi muuttujaa, string-tyyppinen city ja double-tyyppinen temperature. Luokalle on määritelty factory-tyyppinen luontifunktio fromJson, joka mahdollistaa WeatherResponse-tyyppisen objektin luomisen json-vastauksesta.

```
Future<WeatherResponse> fetchWeather(double latitude, double longitude) async {
  final response =
      await http.get(Uri.parse(getWeatherUrl(latitude, longitude)));

  if (response.statusCode != 200) {
    throw Exception('Network request failed');
  }

  return WeatherResponse.fromJson(
      jsonDecode(response.body) as Map<String, dynamic>);
}
```

Kuva 11. Flutter-sovelluksen funktio verkkopyyntöön rajapinnalle

### 5.3.4 Verkkopyyntö React Nativella ja Ionicilla

React Native ja Ionic käyttävät samaa ohjelmointikieltä, joten verkkopyyntö voidaan toteuttaa molemmissa samalla koodilla. Verkkopyynnön tekoon JavaScript- tai TypeScript-koodista on useita tapoja. Kaksi yleistä tapaa on Fetch-rajapinta ja Axios-kirjasto.

Fetch-rajapinta on vanhemman XMLHttpRequest-rapinnan seuraaja.

```typescript
interface WeatherResponse {
    city: number;
    temperature: string;
}

function getWeatherUrl(latitude: number, longitude: number): string {
  return `https://domain.fi/weatherForLocation?latitude=${latitude}&longitude=${longitude}`;
}

async function fetchWeather(latitude: number, longitude: number):Promise<WeatherResponse> {
  try {
    const url = getWeatherUrl(latitude, longitude)
    const response = await fetch(url);
    const responseJson = await response.json();
    return responseJson as WeatherResponse;
  } catch (error) {
    throw new Error('Error fetching data');
  }
};
```

Kuva 12. Verkkopyyntö Fetch-rajapinnalla TypeScript-ohjelmointikielellä

Axios on pakettienhallinnan kautta asennettava kirjasto, joka hyödyntää sisäisesti XMLHTTPRequest-rajapintaa.

```typescript
async function fetchWeatherAxios(latitude: number, longitude: number):Promise<WeatherResponse> {
  try {
    const url = getWeatherUrl(latitude, longitude)
    const response = await axios.get(url);
    return response.data;
  } catch (error) {
    throw new Error('Error fetching data');
  }
};
```

Kuva 13. Verkkopyyntö Axios-kirjastolla TypeScript-ohjelmointikielellä

Erona Axiosin ja Fetchin välillä koodissa yksinkertaisessa get-pyynnössä on se, että vastausta ei tarvitse muuntaa json-muotoon, vaan Axios-kirjasto tekee sen automaattisesti. Axiosin idea on tehdä verkkopyyntöjen tekemisestä kehittäjälle yksinkertaisempaa.

Sääsovelluksessa lähetetään yksi get-pyyntö, joten mielestäni Axios-kirjaston käyttö ei tuo juurikaan lisäarvoa. Vaikka ulkopuolisen kirjaston käyttö voi olla kätevämpää, kirjaston lisääminen projektiin kasvattaa projektin kokoa ja tekee projektille riippuvuuden. Riippuvuuksia tulee päivittää, jotta ne toimivat oikein uudemmissa versioissa.

## 5.4 Erot React Nativen, Flutterin ja Ionicin välillä

Yksinkertaisen sääsovelluksen toteutuksessa ei kovin syvälle alustariippumattomien kehysten käytön osalta menty. Kaikilla kolmella kehyksellä onnistui ongelmitta sijaintia ja verkkopyyntöjä käyttävän sovelluksen toteutus.

React Native ja Ionic Reactin kanssa ovat hyvin samankaltaisia, sillä molemmissa käyttöliittymät tehdään Reactilla. Merkittävänä erona Ionic mahdollistaa myös web-sovelluksen toteutuksen, kun React Native -toteutuksesta ei sellaisenaan voida luoda web-versiota.

# 6 Natiivi vai alustariippumaton mobiilisovellus

## 6.1 Sovelluksen kohdealustat

Alustariippumaton kehitys mahdollistaa sovelluskehittämisen usealle alustalle. On olemassa kuitenkin tarpeita, joissa sovellus halutaan käyttöön vain yhdelle alustalle. Kuluttajasovelluksissa tämä on harvinaista, mutta esimerkiksi moni B2B-sovellus on vain yhdellä alustalla käytettävä. Yhdelle alustalle toteutettu sovellus voisi olla esimerkiksi yrityksen sisäiseen käyttöön tehty sovellus, jota käytetään yrityksen omilla Android-laitteilla.

Mikäli sovellus toteutetaan vain yhdelle alustalle, alustariippumattomuuden tärkein hyöty jää käyttämättä. Natiivisovellus voi olla järkevin valinta, jos tiedetään varmasti, että sovellusta tullaan käyttämään vain yhdellä alustalla.

## 6.2 Kehitystiimi ja kehittäjien osaaminen

Natiivi mobiilisovelluskehitys ja alustariippumattomalla kehyksellä tapahtuva kehitys vaatii hyvin erilaista ohjelmointiosaamista. Esimerkiksi React Native -kehittäminen perustuu pitkälti Reactiin ja web-tekniikoihin. Reactia osaava web-kehittäjä pääsee React Native -kehitykseen huomattavasti nopeammin jyvälle kuin vaikka natiiviin iOS-kehitykseen. Ionicia hyödyntämällä natiivin puolen osaamista tarvitaan vielä vähemmän, sillä sen avulla olemassa oleva web-sovellus voidaan paketoida natiiviksi sovellukseksi.

Natiivisovellusta kehittäessä tulee toteuttaa kaksi erillistä sovellusta, mikäli sovellus halutaan käyttöön niin Android kuin iOS-alustoilla. Näin ollen kehitystiimiltä vaaditaan osaamista niin Android- että iOS-alustasta. Alustariippumatonta kehystä käyttäessä tarvitaan merkittävästi

vähemmän ymmärrystä Android- ja iOS-alustoista. Kolmannen osapuolen kirjastoja hyödyntäen alustariippumattomalla kehyksellä on mahdollista tehdä mobiilisovellus ilman riviäkään natiivikoodin kirjoitusta.

Stack Overflown vuoden 2023 kyselyssä kehittäjiltä kysyttiin minkä ohjelmointikielen parissa on työskennelty sekä millä ohjelmointikielellä haluttaisiin työskennellä seuraavan vuoden aikana. Ammatikseen ohjelmoijista JavaScriptin oli maininnut n. 66 % kun taas iOS-natiivikehityksessä käytetyn Swiftin oli maininnut n. 5 % (Stack Overflow, 2023). Työmarkkinoilta löytyy todennäköisesti merkittävästi enemmän JavaScript-osaajia kuin iOS-osaajia. Kehittäjiä saa todennäköisesti helpoiten JavaScript-pohjaiseen alustariippumattomaan projektiin.

## 6.3 Suorituskyky

Mobiilisovelluksen suorituskykyä pidetään usein tärkeänä asiana. Heikko suorituskyky näkyy käyttäjälle mm. pitkinä latausaikoina ja jumittamisena. Teoreettisella tasolla tarkasteltuna natiivisovellus on suorituskyvyn osalta paras vaihtoehto, sillä natiivisovellus käyttää suoraan käyttöjärjestelmän rajapintoja.

Käytännössä kuitenkin suurin osa mobiilisovelluksista ei sisällä elementtejä, jotka tekisivät alustariippumattomalla kehyksellä toteutusta sovelluksesta riittämättömän suorituskyvyn osalta. Alustariippumattomasti toteutettu sovellus voi olla käyttäjälle yhtä sulava kuin natiivi sovellus. Tärkeämpää on se, että ohjelmakoodi on hyvälaatuista ja tehokasta. Toisin päin ajateltuna, huonosti ohjelmoitu natiivisovellus voi olla suorituskyvyltään erittäin heikko.

Mikäli sovellus on tyypillinen mobiilisovellus, alustariippumaton kehys harvoin asettaa rajoitteita sovelluksen suorituskyvylle. Mikäli sovellus sisältää esimerkiksi raskaita tai monimutkaisia animaatioita, tai sovellus on muuten erityisen resursseja käyttävä, voi alustariippumaton kehys muodostua pullonkaulaksi (Inverita, n.d.).

## 6.4 Käyttöjärjestelmän rajapinnat tai ominaisuudet

Kehitettävässä mobiilisovelluksessa on usein tarpeen hyödyntää laitteen ominaisuuksia monipuolisesti. Tyypillisesti hyödynnettävistä ominaisuuksia ovat esimerkiksi:

- sijainti
- kamera
- äänen tallennus

- äänen toisto

Natiivin sovelluksen ollessa kyseessä, laitteen ominaisuuksia voi kutsua suoraan ohjelmakoodista.

```
import AVFoundation

func playSound(url: URL) {

    do {
        let player = try AVAudioPlayer(contentsOf: url)
        player.play()

    } catch let error {
        print(error.localizedDescription)
    }
}
```

Kuva 14. Äänitiedoston toistaminen Swift-ohjelmointikielellä

Kuvan 14 koodissa toistetaan iOS-laitteella äänitiedosto. Ulkopuolisia kirjastoja ei tarvita, ensimmäisellä rivillä tuodaan AVFoundation-kirjasto, joka toimitetaan iOS SDK:n mukana.

Alustariippumattoman sovelluksen koodista ei voida kutsua suoraan käyttöjärjestelmän rajapintoja, vaan väliin tarvitaan oma toteutus tai kolmannen osapuolen kirjasto. Kolmannen osapuolten kirjastot ovat usein avoimen lähdekoodin yhteisön ylläpitämiä, ja niiden laatu voi vaihdella todella paljon. On myös paljon paketteja, joiden aktiivinen kehitys on lopetettu. Esimerkiksi äänen toistoon suosittu npm-paketti react-native-sound kerää viikoittaisia latauksia yli 100 000, vaikka viimeisin versio on julkaistu kaksi vuotta sitten.

Alustariippumattomassa kehityksessä tulee käyttää tarpeeksi aikaa ja huolellista harkintaa kolmannen osapuolten paketteja valitessa. Niitä tarvitaan merkittävästi enemmän kuin natiivissa kehityksessä.

## 6.5 Ylläpito

Mobiilisovelluksen kehitys harvoin päättyy sovelluksen julkaisuun. Mobiilikäyttöjärjestelmät päivittyvät tiheään tahtiin, ja tämä vaatii myös kehittäjältä huomiota.

Android ja iOS -käyttöjärjestelmistä julkaistaan keskimäärin kerran vuodessa uusi major-versio. Major-version mukana tulee käyttöjärjestelmään uusia ominaisuuksia ja samalla osa ominaisuuksista merkitään poistuviksi (deprecated) tai poistetaan kokonaan. Ylläpitäessä natiivia sovellusta poistuviksi merkittyjä ominaisuuksia tulee seurata. Monet

ohjelmointiympäristöt osaavat kertoa kehittäjälle, mikäli ohjelmakoodi käyttää poistuviksi merkittyjä ominaisuuksia.

Alustariippumattoman kehyksen versiopäivityksissä tulee myös vastaavanlaisia muutoksia. Kuitenkin muutoksia on usein enemmän kuin natiivikäyttöjärjestelmän versiopäivityksessä. Esimerkiksi iOS-käyttöjärjestelmän version 17 julkaisutiedoissa ainoastaan kaksi ominaisuutta on merkitty poistuvaksi ja yksi poistettu (Apple Developer, n.d.-g). React Nativen 0.70-versiopäivityksessä yhdeksän ominaisuutta poistettiin ja yksi merkittiin poistuvaksi (Meta Open Source, n.d.-a).

Niin natiivissa kuin alustariippumattoman sovelluksen ylläpitokehityksessä on pidettävä riippuvuudet ajan tasalla, sillä myös riippuvuuksien tulee olla yhteensopivia käytetyn alustan kanssa. Koska monet riippuvuudet ovat avoimen lähdekoodin yhteisön ylläpitämiä, voi niiden kehitys loppua yllättäen.

# 7 Sovelluksen julkaiseminen

## 7.1 Julkaisukanavat

Sovelluskauppa on tyypillisin tapa julkaista sovellus, varsinkin jos sovellus halutaan saada yleiseen jakeluun.

Valtaosa Android-laitteista sisältää ohjelmiston, jonka mukana on Google Play -sovelluskauppa. Google Play on Androidin pääkehittäjä Googlen sovelluskauppa, ja sitä voidaan pitää tärkeimpänä jakelukanavana Android-sovelluksille.

Kaikki iOS-laitteet (pois lukien 1. sukupolven iPhone ja iPod touch) sisältävät App Store -sovelluskaupan. App Store on ollut vuosina 2008–2023 ainoa jakelukanava iOS-sovelluksille. Vuonna 2024 julkaistu iOS-versio 17.4 mahdollistaa EU:ssa asuville käyttäjille sovellusten asentamisen vaihtoehtoisista sovelluskaupoista (Apple, 2024).

Android-käyttöjärjestelmälle on saatavilla vaihtoehtoisia sovelluskauppoja kuten Huawei AppGallery ja Amazon App Store.

## 7.2 Julkaisukanavien kustannukset

Sovelluksen jakeluun App Storessa vaaditaan voimassa oleva Apple Developer Program -jäsenyys, jonka hinta on 99 dollaria vuodessa (Apple Developer, n.d.-c). Jäsenyydellä julkaistavien sovellusten määrää ei ole rajoitettu. Mikäli App Storen kautta halutaan myydä maksullista sovellusta tai sovelluksen sisäisiä ostoja, App Storen komissio on iOS-alustalla 17% ja iPadOS-alustalla 27%. Tämän päälle veloitetaan maksutavan käsittelymaksu, joka on 3% (Apple Developer, n.d.-d).

Play Storessa vuosimaksua ei ole, mutta rekisteröityessä tulee maksaa 25 dollarin rekisteröintimaksu (Google, n.d.-a). Maksullisten sovellusten ja sovellusostojen myynnistä Play Store veloittaa 15% komission ensimmäisen miljoonan dollarin osalta vuodessa, jonka ylimenevältä osalta komissio on 30%. Tilausten osalta komissio on 15% riippumatta vuosittaisesta myynnistä (Google, n.d.-c).

## 7.3 Sovelluskaupan arviointiprosessi

Sovelluskaupalla on viime kädessä oikeus päättää mitkä sovellukset hyväksytään sovelluskaupan jakeluun. App Storessa ja Play Storessa on omat kriteerit julkaistavalle sovellukselle. Ennen julkaisua sovellukset tarkistetaan laajasti, esim. haitallisen sisällön ja toimivuuden osalta.

### 7.3.1 App Store

Julkaistaessa sovellusta App Storeen julkaisijan tulee perehtyä Applen ohjeeseen App Review Guidelines (Apple Developer, n.d.-a).

Ennen sovellusversion lähetystä arvioitavaksi tulee sovellus testata huolellisesti virheiden ja kaatumisten varalta. Lisäksi tulee varmistaa, että sovelluksen metatiedot ovat valmiina, kirjata yhteystiedot julkaisijan tavoittamista varten, sekä antaa sovelluskaupan arviointitiimille tarvittavat tiedot sovelluksen testaamista varten. Näitä tietoja voisi olla esimerkiksi käyttäjätunnus ja salasana, mikäli sovelluksen käyttö vaatii kirjautumisen. Jos sovellus kommunikoi rajapinnan kanssa, tulee rajapinnan olla käytettävissä ennen arviointia.

Vaikka sovellus toimisi teknisesti moitteetta, voidaan sen julkaisu silti evätä. Sovellus ei saa sisältää loukkaavaa tai muuten sopimatonta sisältöä. Sovelluksen tulee täyttää myös App Storen vähimmäislaatuvaatimukset, joiden mukaan sovelluksen tulee olla sovellusmainen, eikä esimerkiksi pelkästään uudelleenpakattu verkkosivusto. Vaatimuksia perustellaan sillä,

että Applen asiakkaat arvostavat laadukkaita, helppokäyttöisiä ja innovatiivisia sovelluksia. Jos sovellus ei tarjoa käyttäjälleen selvää hyötyä tai ainutlaatuisuutta, sen jakelua ei välttämättä sallita App Storessa. Opinnäytetyössä esitellyn sääsovelluksen julkaisua App Storeen ei todennäköisesti sallittaisi sovelluksen yksinkertaisuuden vuoksi. Varmaksi asiaa ei kuitenkaan voida sanoa, sillä päätös sovelluksen julkaisemisesta tai sen julkaisematta jättämisestä on loppupeleissä App Storen arviointitiimillä.

### 7.3.2 Google Play

Google Playn kehittäjien käytäntökeskuksessa on määritelty käytännöt, jotka tulee ottaa huomioon ennen sovelluksen julkaisemista Google Playssa.

Julkaistava sovellus ei saa sisältää sopimatonta tai laitonta sisältöä. Sovellus ei saa antaa vaikutelmaa, että se olisi jonkun toisen yrityksen tai yhteisön julkaisema sovellus. Esimerkiksi muun kuin Googlen julkaisema kuvitteellinen sovellus ”YouTube Analytics” loukkaisi tätä sääntöä, sillä sovelluksen nimi antaisi virheellisen vaikutelman virallisesta Googlen julkaisemasta sovelluksesta (Google, n.d.-b).

Lohkoketjua hyödyntäviin sovelluksiin on rajoituksia. Mikäli sovellus on kryptovaluutan kaupankäyntipaikka tai kryptovaluutan lompakko, tulee sovelluksen noudattaa paikallista lainsäädäntöä ja julkaisijan tulee varautua toimittamaan Googlelle asiakirjoja, joista käy ilmi toiminnan lainmukaisuus. Sovelluksia, joilla voidaan louhia kryptovaluuttaa laitteessa ei sallita. Kryptovaluuttaa louhivien laitteiden etähallintaan tarkoitetut sovellukset ovat sallittuja.

Sovellus ei saa sisältää roskapostia lähettäviä ominaisuuksia. Esimerkiksi sovelluksen jaa-painike, joka käyttäjältä lupaa kysymättä lähettää tekstiviestillä sovelluksen mainoksen jokaiselle yhteystiedolle täyttää tämän kriteerin.

Vähimmäislaatuvaatimukset ovat Google Playssa vähäisemmät kuin App Storessa. Yleisohjeena on, että sovelluksen tulee tarjota vakaa, vaikuttava ja sulava käyttökokemus. Ohjeessa mainitaan esimerkkinä sovellus, jossa ei ole mitään toiminnallisuutta ei täytä näitä vaatimuksia. Toisena esimerkkinä on sovellus, joka kaatuu virheeseen heti sen avatessaan. Kuitenkaan vaatimusta sovelluksen hyödyllisyydestä tai viihteellisyydestä ei mainita. Erittäin todennäköisesti opinnäytetyössä esitelty esimerkkisovellus menisi ongelmitta läpi Googlen arvioinnista.

## 7.4 Tarvittavat metatiedot sovelluksen julkaisua varten

Luvussa käsitellään yleisellä tasolla sitä, minkälaisia metatietoja sovelluksen julkaisuun sovelluskaupassa tarvitaan. Sovelluskauppojen välillä on eroja, vähintäänkin ominaisuuksia on nimetty eri tavalla, mutta lähtökohtaisesti kaikkiin sovelluskauppoihin julkaistaessa tarvitaan metatiedot.

Sovellus tarvitsee nimen, jolla se on listattuna ja haulla löydettävissä sovelluskaupassa. Sama nimi näkyy käyttäjän mobiililaitteen sovellusvalikossa tai kuvakkeissa.

Sovellukselle valitaan kategoria, joka auttaa sovelluksen löytämistä sovelluskaupassa. App Storessa kategorioita ovat esimerkiksi lapset, musiikki ja urheilu (Apple Developer, n.d.-b).

Kuvankaappaukset sovelluksesta ja kuvausteksti auttavat käyttäjiä tutustumaan sovelluksen ennen sen lataamista. Nämä ovat pakollisia sekä hyvin tärkeitä tietoja sovelluskaupan jakelua ajatellen.

Sovelluksen julkaisijan tulee arvioida ja määritellä sovellukselle oikea ikärajoitus. Google Playn ikärajoitukset Euroopassa määritellään mm. videopeleistä tutulla PEGI-asteikolla. Alin PEGI 3 asteikko tarkoittaa, että sovellus sopii kaikille, eikä sisällä esimerkiksi kiroilua tai pelottavia kuva- tai äänimateriaalia. PEGI 18 -luokitus on tarpeen, mikäli sovellus sisältää esimerkiksi törkeää väkivaltaa tai huumeiden käytön glamorisointia (Google, n.d.-d).

Linkki sovelluksen tietosuojaselosteen verkkosivulle on suositeltavaa ja monesti pakollista toimittaa. Tietosuojaselosteessa tulee kertoa, miten sovellus kerää tietoa ja miten sitä käsitellään.

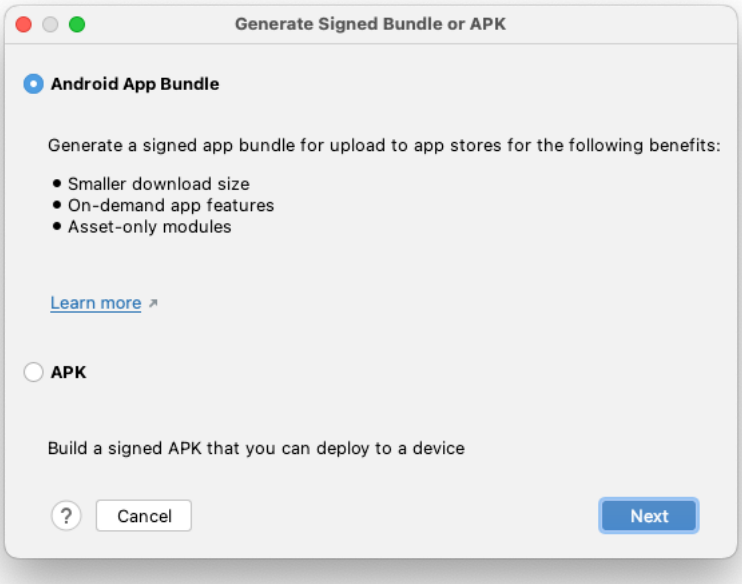
## 7.5 Sovellusversion lähetys sovelluskauppaan

Jakelua varten valmiista sovelluksesta luodaan arkistotiedosto, joka sisältää ajettavan sovelluksen ja sen tarvitsemat tiedostot.

Androidilla arkistotiedosto on Android Package (apk) tai Android App Bundle (aab) (Android Developers, n.d.-a). Android App Bundle on uudempi formaatti, ja uudet Google Playssa julkaistavat sovellukset tulee lähettää aab-muodossa (The Verge, 2021). iOSilla on yksi arkistotiedostoformaatti, ipa.

Opinnäytetyössä käsitellyistä alustariippumattomissa ohjelmistokehyksissä iOS-sovelluksen arkistointitiedosto voidaan tehdä Xcode-ohjelmointiympäristöstä. Sovelluksen ajokonfiguraatioksi tulee määriteltynä Release eli julkaisuversio. Tämä varmistetaan polusta Product – Scheme – Edit Scheme – Run. Arkistotiedosto tehdään valitsemalla Xcoden valikosta Product – Archive. Arkistotiedoston luonnin valmistuttua Xcode avaa Organizer-näkymän, josta sovellus voidaan viedä App Storeen Distribute App -painikkeen kautta.

Vastaavasti Android-sovelluksen arkistotiedosto voidaan tehdä Android Studiota käyttäen. Android Studion valikosta valitaan Build – Generate Signed Bundle / APK.




Kuva 15. Valintaikkuna Android Studiossa sovellusbinääriä tehdessä

# 8 Yhteenveto

Opinnäytetyössä käsiteltiin natiivia mobiilikehitystä, alustariippumatonta mobiilikehitystä ja haettiin eroja näiden välille. Esimerkkiprojektin toteutus onnistui ongelmitta kaikilla opinnäytetyöhön valituilla kehyksillä, Flutterilla, Ionicilla ja React Nativella.

Havaittiin, että valinnassa natiivin ja alustariippumattoman kehitystavan välillä ei ole yksiselitteistä vastausta, vaan valinta riippuu monesta eri tekijästä. Alustariippumattomat sovelluskehitykset ovat tänä päivänä niin kehittyneitä, että valinta tehdään usein kehitystiimin tai käytettävissä olevien resurssien perusteella. Useimmat sovellukset pystytään toteuttamaan laadukkaasti alustariippumattomina sovelluksina.

Sovelluksen julkaisemiseen liittyy monia eri vaiheita. Ei riitä, että valmiin sovelluksen lähettää sovelluskauppaan, vaan julkaisuun on varattava riittävä määrä aikaa. Vaiheita on useita, niin sovelluskaupan julkaisijatilin luomisesta alkaen, loppuen sovellusversion lähettämiseen kauppaan. Sovelluskauppojen välillä ehdot eroavat myös jonkin verran.

## Lähteet


9to5Mac. (21.10.2011). *Jobs' original vision for the iPhone: No third-party native apps.* https://9to5mac.com/2011/10/21/jobs-original-vision-for-the-iphone-no-third-party-native-apps/

Android Developers. (n.d.-a). *Application fundamentals.* Haettu 20.4.2024 osoitteesta https://developer.android.com/guide/components/fundamentals

Android Developers. (n.d.-b). *Connect to the network.* Haettu 28.3.2024 osoitteesta https://developer.android.com/develop/connectivity/network-ops/connecting

Android Developers. (n.d.-c). *Download Android Studio & Tools.* https://developer.android.com/studio

Android Developers Blog. (23.7.2008). *Announcing the Android 1.0 SDK, release 1.* https://android-developers.googleblog.com/2008/09/announcing-android-10-sdk-release-1.html

Apple. (6.3.2008). *Apple Announces iPhone 2.0 Software Beta.* https://www.apple.com/newsroom/2008/03/06Apple-Announces-iPhone-2-0-Software-Beta/

Apple. (2.6.2014). *Apple Releases iOS 8 SDK With Over 4,000 New APIs*. https://www.apple.com/newsroom/2014/06/02Apple-Releases-iOS-8-SDK-With-Over-4-000-New-APIs/

Apple. (12.10.2023). *Jos appi haluaa muodostaa yhteyden lähiverkon laitteisiin.* https://support.apple.com/fi-fi/102229

Apple. (11.3.2024). *Tietoja vaihtoehtoisista appien markkinapaikoista Euroopan unionissa.* https://support.apple.com/fi-fi/118110

Apple Developer. (n.d.-a). *App Review Guidelines.* Haettu 17.4.2024 osoitteesta https://developer.apple.com/app-store/review/guidelines/

Apple Developer. (n.d.-b). *Choosing a category.* Haettu 18.4.2024 osoitteesta https://developer.apple.com/app-store/categories/

Apple Developer. (n.d.-c). *Choosing a Membership.* Haettu 15.4.2024 osoitteesta https://developer.apple.com/support/compare-memberships/

Apple Developer. (n.d.-d). *Commissions, fees, and taxes.* Haettu 15.4.2024 osoitteesta https://developer.apple.com/help/app-store-connect/distributing-apps-in-the-european-union/commissions-fees-and-taxes/

Apple Developer. (n.d.-e). *NSAppTransportSecurity.* Haettu 1.4.2024 osoitteesta https://developer.apple.com/documentation/bundleresources/information_property_list/nsapptransportsecurity/

Apple Developer. (n.d.-f). *NSExceptionAllowsInsecureHTTPLoads.* Haettu 1.4.2024 osoitteesta

https://developer.apple.com/documentation/bundleresources/information_property_list/nsexceptionallowsinsecurehttploads

Apple Developer. (n.d.-g). *iOS & iPadOS 17 Release Notes.* https://developer.apple.com/documentation/ios-ipados-release-notes/ios-ipados-17-release-notes

Adobe. (26.5.2023). *Top 10 benefits of mobile apps for your business.* https://business.adobe.com/blog/basics/benefits-of-mobile-apps

Ars Technica. (2.5.2015). *Google's Dart language on Android aims for Java-free, 120 FPS apps.* https://arstechnica.com/gadgets/2015/05/googles-dart-language-on-android-aims-for-java-free-120-fps-apps/

Cloudflare. (n.d.). *Why is HTTP not secure? | HTTP vs. HTTPS.* Haettu 30.3.2024 osoitteesta https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/

Dart. (n.d.-a). *Dart Overview.* Haettu 21.3.2024 osoitteesta https://dart.dev/overview

Dart. (n.d.-b). *Sound null safety.* Haettu 24.3.2024 osoitteesta https://dart.dev/null-safety

Flutter. (n.d.-a) *Building user interfaces with Flutter.* Haettu 21.3.2024 osoitteesta https://docs.flutter.dev/ui

Flutter. (n.d.-b). *Fetch data from the internet.* Haettu 2.4.2024 osoitteesta https://docs.flutter.dev/cookbook/networking/fetch-data

Flutter. (n.d.-c). *Future<T> class.* Haettu 26.4.2024 osoitteesta https://api.flutter.dev/flutter/dart-async/Future-class.html

Google. (n.d.-a). *How to get started with Play Console.* Haettu 15.4.2024 osoitteesta https://support.google.com/googleplay/android-developer/answer/6112435?hl=en#zippy=%2Cstep-pay-registration-fee

Google. (n.d.-b). *Impersonation.* Haettu 17.4.2024 osoitteesta https://support.google.com/googleplay/android-developer/answer/9888374

Google. (n.d.-c). *Service fees.* Haettu 15.4.2024 osoitteesta https://support.google.com/googleplay/android-developer/answer/112622

Google. (n.d.-d). *Sovellusten ja pelien ikärajoitukset Google Playssa.* Haettu 18.4.2024 osoitteesta https://support.google.com/googleplay/answer/6209544?hl=fi#zippy=%2Ceurooppa-ja-lähi-itä

Ionic. (n.d.-a). *Introduction to Ionic.* Haettu 24.3.2024 osoitteesta https://ionicframework.com/docs

Ionic. (n.d.-b). *UI Components.* Haettu 24.3.2024 osoitteesta https://ionicframework.com/docs/components

Inverita. (n.d.). *Flutter vs React Native vs Native: Deep Performance Comparison.* Haettu 10.4.2024 osoitteesta https://inveritasoft.com/blog/flutter-vs-react-native-vs-native-deep-performance-comparison

Macworld. (28.2.2024). *iOS Versions: Every version of iOS from the oldest to the newest.* Haettu 5.3.2024 osoitteesta https://www.macworld.com/article/1659017/ios-versions-list.html

Meta Open Source. (n.d.-a). *Changelog.* Haettu 18.4.2024 osoitteesta https://github.com/facebook/react-native/blob/main/CHANGELOG.md#0700

Meta Open Source. (n.d.-b). *Core Components and APIs.* Haettu 15.3.2024 osoitteesta https://reactnative.dev/docs/components-and-apis

Meta Open Source. (n.d.-c). *Using TypeScript.* Haettu 15.3.2024 osoitteesta https://reactnative.dev/docs/typescript

Meta Open Source. (n.d.-d). *Writing Markup with JSX.* Haettu 15.3.2024 osoitteesta https://react.dev/learn/writing-markup-with-jsx

Stack Overflow. (13.6.2023). *Stack Overflow Developer Survey 2023.* https://survey.stackoverflow.co/2023/

Solwey Consulting. (24.3.2023). *Static vs Dynamic Typing: Choosing the Right Approach for Your Programming Needs.* https://www.solwey.com/posts/static-vs-dynamic-typing-choosing-the-right-approach-for-your-programming-needs

Statcounter. (1.3.2024). *Mobile Operating System Market Share Worldwide.* Haettu 5.3.2024 osoitteesta https://gs.statcounter.com/os-market-share/mobile/worldwide https://www.techaheadcorp.com/knowledge-center/history-of-react-native/

TechAhead. (16.11.2023). *The History of React Native: Facebook's Open Source App Development Framework.*

Tilastokeskus. (n.d.). *Understanding Permission Flow.* Haettu 15.4.2024 osoitteesta https://stat.fi/tup/suoluk/suoluk_digitalisaatio.html

The Verge. (30.6.2021). *Google is moving away from APKs on the Play Store.* https://www.theverge.com/2021/6/30/22557390/google-apk-app-bundles-package-format-play-store

Zoontek. (n.d.). *Understanding Permission Flow.* Haettu 27.3.2024 osoitteesta https://github.com/zoontek/react-native-permissions?tab=readme-ov-file#understanding-permission-flow