OULUN AMMATTIKORKEAKOULU

Hannes Kinnunen

MOBIILISOVELLUKSEN TOTEUTTAMINEN FLUTTERIN AVULLA

MOBIILISOVELLUKSEN TOTEUTTAMINEN FLUTTERIN AVULLA

Hannes Kinnunen
Opinnäytetyö
Kevät 2023
Tietojenkäsittelyn tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Informaatioteknologia, Tietojenkäsittelyn tutkinto-ohjelma

Tekijä: Hannes Kinnunen
Opinnäytetyön nimi: Mobiilisovelluksen toteuttaminen Flutterin avulla
Työn ohjaaja: Reima Riihimäki
Työn valmistumislukukausi ja -vuosi: Kevät 2023 Sivumäärä: 39

Opinnäytetyön aiheena on Android-mobiilisovelluksen toteutus Flutterilla. Flutter on Googlen kehittämä avoimen lähdekoodin ohjelmistokehys alustariippumattomien mobiilisovellusten luontiin. Sovelluksella ei ollut toimeksiantajaa, vaan sovelluksen ideointi, suunnittelu ja toteutus kuuluvat tähän opinnäytetyöhön.

Teoriaosuus käsittelee Android-mobiilisovelluskehitystä ja erilaisia vaihtoehtoja siihen. Teoriaosuudessa kerrotaan myös Flutterin perusteista, sillä niitä käytettiin tässä toimeksiannossa. Aineistona Flutterin perusteisiin käytettiin pääasiassa Flutterin verkkosivuilta löytyvää kattavaa dokumentaatiota. Teoriaosuudessa käytiin läpi myös, mitä Googlen Material Design -ohjeet ovat, sillä niitä noudatettiin sovelluksessa.

Raporttiosuudessa suunnitellaan ja toteutetaan meditointiin tarkoitettu mobiilisovellus. Suunnitteluun kuuluvat sovelluksen ideointi sekä käyttöliittymän rautalankamallin piirtäminen. Luvussa käydään myös läpi, miten Flutter-projekti luodaan ja mitä kaikkea ohjelmia Flutter-sovelluskehitys vaatii.

Opinnäytetyön tuloksena syntyi käytettävä, nopea ja uniikki meditointisovellus. Tavoitteena oli tutustua Flutteriin ja luoda sovellus, jossa on miellyttävä käyttäjäkokemus. Sovelluksen suunnittelijana ja pääasiallisena käyttäjänä olen tyytyväinen lopputulokseen ja käytän sovellusta melkein päivittäin.

Asiasanat: ohjelmistokehitys, mobiilisovellus, Flutter

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Degree Programme in Business Information Systems

---

Author: Hannes Kinnunen
Title of thesis: Building a mobile application with Flutter
Supervisor: Reima Riihimäki
Term and year when the thesis was submitted: Spring 2023 Number of pages: 39

The subject for this thesis was creating an Android application with Flutter. Flutter is open-source cross-platform framework for creating mobile applications, developed by Google. The application wasn't made for any client. Ideation, planning and development of the application is part of this thesis.

The theoretical portion is about Android mobile application development and different options for it. The chapter also talks about basics of Flutter, as they were used for the assignment. The information about Flutter was mostly gathered from the framework's own documentation. The theoretical portion also explains what's Material Design as it was used for the app.

The report focuses on planning and developing an application used for meditation. The planning includes coming up for the idea and features for the application, as well as drawing a wireframe for the user interface. The chapter also includes how you create a Flutter project, and what software Flutter development requires.

The thesis produced a usable, fast and unique meditation app. The goal was to get used to Flutter and create an application with a pleasant user experience. I'm satisfied with the result and use the app almost daily myself.

Keywords: Software development, mobile application, Flutter

SISÄLLYS

1 JOHDANTO

Suurimmassa osassa älypuhelimista on nykyään joko Googlen Android-käyttöjärjestelmä tai Applen iOS-käyttöjärjestelmä. Viimeisen vuoden aikana (tammikuu 2022–tammikuu 2023) iOS-käyttöjärjestelmän markkinaosuus oli maailmanlaajuisesti 27,63 % ja Android-käyttöjärjestelmän jopa 71,74 %. Yhteensä näiden kahden alustan markkinaosuus on siis 99,37 %. Sen sijaan että kehittäisi saman sovelluksen molemmille alustoille, Androidille Kotlinilla ja iOSsille Swiftillä, sovelluksen voi kehittää monialustaisella ohjelmistokehyksellä. Monialustaisen ohjelmistokehyksen avulla sovellus voidaan kääntää useammalle alustalle käyttäen samaa koodia. Tämä säästää aikaa ja rahaa, ja vaatii vain yhden ohjelmointikielen opettelun. (StatCounter 2023.)

Tämän opinnäytetyön tarkoituksena on perehtyä Flutteriin, joka on Google kehittämä avoimen lähdekoodin ohjelmistokehys alustariippumattomien mobiilisovellusten luomiseksi, ja kertoa siitä sekä sen hyödyistä. Raporttiosuudessa kerrotaan mobiilisovelluksen kehittämisestä, käydään läpi mitä työkaluja se vaatii ja miltä Flutter-koodi näyttää. Tavoitteena on saada aikaan toimiva, hyvännäköinen ja helposti käytettävä mobiilisovellus kyseisellä ohjelmistokehyksellä.

Kehitettävä sovellus tulee olemaan meditoinnissa avustava sovellus, jolla käyttäjä voi ajastaa meditointeja. Tärkeimpiä ominaisuuksia tulevat olemaan satunnaiset meditointiajat ja statistiikka omista meditointikerroista sekä niiden pituuksista. Sovelluksessa tullaan keskittymään siihen, että käyttöliittymä näyttää hyvältä on responsiivinen ja noudattaa Googlen Material Design -ohjeita.

Aihe opinnäytetyöhön syntyi, kun halusin lisätä vaihtelua omiin meditointeihin ja parantaa niiden kokemusta. Kun aloitin meditoimaan, käytin puhelimen omaa kellosovellusta ajastimena. Kellon oma ajastin, ajan loputtua, jää hälyttämään niin kauan, että käyttäjä sulkee hälytyksen. Tämän takia meditointia oli hankala jatkaa ajan loputtua. Halusin sovellukseen myös satunnaiset meditointiajat. Haluan että käyttäjä voi asettaa sovellukseen minimi- ja maksimiajan, joiden väliltä sovellus arpoo joka kerta eri meditointiajan. Tämän avulla jokainen meditointi pysyy hieman erilaisena.

Meditointiin on olemassa jo paljon valmiita sovelluksia, kuten Headspace yhtenä tunnetuimmista. Nämä suositut sovellukset kuitenkin vaativat yleensä rekisteröitymisen ja joskus jopa kuukausimaksuja. Tämän lisäksi näissä sovelluksissa on usein keskipisteenä opastetut meditoinnit. Opinnäytetyössä kehitettävä sovellus tulee olemaan ilmainen ja yksinkertainen, ilman opastettuja meditointeja. Tarkoitus ei kuitenkaan ole kilpailla suosittujen meditointisovellusten,

kuten Headspacen kanssa, vaan tuoda tarjolle avoimen lähdekoodin vaihtoehto sille pienelle ryhmälle, joka kaipaa satunnaisia meditointiaikoja.

Opinnäytetyöhön valittiin teknologiaksi Flutter valmiiden materiaalikomponenttien saatavuuden takia, koska on mahdollista kääntää sovellus usealle alustalle, ja siksi, että minulla on oma mielenkiinto kyseistä ohjelmistokehystä kohtaan. Opinnäytetyössä tullaan myös käyttämään Microsoftin omistamaa Visual Studio Codea ja Android Studion Android-emulaattoria, joka emuloi Android-puhelinta tietokoneella ilman fyysistä puhelinta.

2 KÄYTETTÄVÄT TEKNOLOGIAT JA TYÖKALUT

Android-sovellusten kehittämiseen on nykyään tarjolla paljon eri vaihtoehtoja. Java oli aluksi virallinen kieli Android-sovellusten kehittämiseen. Java on vanha ohjelmointikieli, luotu yli 25 vuotta sitten, mutta on vieläkin yksi suosituimmista ohjelmointikielistä. (GeeksforGeeks 2022; Forbes 2022.)

Vuonna 2019 Google julisti Kotlinin viralliseksi kieleksi Android-sovellusten kehittämiseen. Kotlin on JetBrainsin kehittämä avoimen lähdekoodin ohjelmointikieli. Se on yhteensopiva Javan kanssa, eli Java-koodi ja Kotlin-koodi osaavat kommunikoida keskenään. Kotlinin syntaksi on myös huomattavasti Javaa helpompi, mikä tekee kielestä aloittelijaystävällisen. Vaikka kieli onkin Javaa uudempi, eikä se ole ollut vielä montaa vuotta Androidin virallinen kieli, yli 80 % Androidin suosituimmista sovelluksista käyttää Kotlinia, (Plain Concepts 2022,)

Android-sovellusten kehittämiseen voi kuitenkin käyttää myös alustariippumatonta viitekehystä (framework). Alustariippumattomien viitekehyksien avulla voi luoda sovelluksen useammalle alustalle, esimerkiksi Androidille ja iOSsille, käyttämällä vain yhtä koodia (single codebase). Sovellusta ei tarvitse siis kirjoittaa erikseen molemmille alustoille. Tämän avulla kehittäjät säästävät aikaa sekä varmistavat, että sovellus näyttää samalta ja toimii samalla tavalla molemmilla alustoilla. Suosituimmat viitekehykset alustariippumattomalle sovelluskehitykselle ovat React Native ja Flutter. React Native on vuonna 2015 Meta Platformsin kehittämä avoimen lähdekoodin viitekehys. React Nativessa käytetään JavaScript-ohjelmointikieltä. React Native oli käytetyin alustariippumaton viitekehys mobiilisovellusten luontiin, kunnes vuonna 2021 Flutter ylitti tämän suosiossa. Statistan mukaan, vuonna 2021, alustariippumattoman viitekehyksen käyttäjistä mobiilisovellusten luontiin 42 % käytti Flutteria, ja 38 % käytti React Nativea. Tässä opinnäytetyössä sovellus luodaan Flutterilla. (BrowserStack 2022.)

2.1 Flutter

Flutter on Googlen vuonna 2017 kehittämä avoimen lähdekoodin (open source) Dart-ohjelmistokehys. Alun perin sen piti olla mobiilisovelluksiin keskittynyt ohjelmistokehys, mutta

nykyään Flutterilla voi kehittää myös webbi-, Windows-, Linux- ja macOS-sovelluksia (taulukko 1). (Flutter 2023a.)

TAULUKKO 1. Flutterin tukemat alustat

Alusta	**Versio**
Android	API 16 (Android 4.1) ja uudemmat
iOS	iOS 11 ja uudemmat
Linux	Debian, 64-bit
macOS	El Capitan (10.11) ja uudemmat
Web	Chrome 84 ja uudemmat
Web	Firefox 72.0 ja uudemmat
Web	Safari on El Capitan ja uudemmat
Web	Edge 1.2.0 ja uudemmat
Windows	Windows 8 ja uudemmat, 64-bit

Flutterilla tehdyissä sovelluksissa on korkea suorituskyky. Sovellukset eivät tarvitse siltaa (bridge), kuten React Native, kertomaan käyttöjärjestelmälle, mitä ruudulla halutaan näyttää. Flutterin oma Skia-grafiikkamoottori piirtää käyttöliittymän itse kokonaan. Tämän ansiosta Flutterilla tehdyt sovellukset ovat yhtä nopeita kuin natiivit Kotlinilla tai Javalla luodut sovellukset. Flutterissa on myös suosittu ominaisuus nimeltä Hot Reload. Hot Reload päivittää koodissa tehdyt muutokset puhelimeen tai emulaattoriin välittömästi, ilman että kehittäjän tarvitsee kääntää sovellusta uudelleen. Hot Reload ei myöskään muuta sovelluksen tilaa. Tämän ansiosta Flutterilla kehittäminen on erittäin nopeaa ja vaivatonta (Cleveroad 2020.)

2.2 Dart

Flutterissa käytetään Dart-ohjelmointikieltä. Dart on avoimen lähdekoodin olioperustainen kieli, jonka Google kehitti vuonna 2011. Dart on saanut inspiraatiota muista ohjelmointikielistä, kuten

Javasta (kuva 1), JavaScriptistä sekä C#:sta. Dart on käännettävä kieli, eli sitä ei voi suorittaa suoraan vaan se pitää ensin kääntää konekielelle.

Dart tukee yleisiä ohjelmointikäsitteitä, kuten luokkia, rajapintaluokkia (interface) ja funktioita. Dart on myös vahvasti tyypitetty (strongly typed) kieli, eli muuttujille pitää aina määritellä tyyppi. Tyyppejä voivat olla esimerkiksi merkkijono (string), numero (num, int, double), totuusarvomuuttuja (boolean) tai objekti (Javatpoint 2020.)

```java
// Java code
public static void main(String args[]) {

    List<String> names = Arrays.asList("Matti", "Teppo", "Johannes");
    for(int i = 0; i < names.size(); i++) {
        String name = names.get(i);
        System.out.println(i + ": " + name);
    }

}
```

```dart
// Dart code
void main() {

    var names = <String>["Matti", "Teppo", "Johannes"];
    for(var i = 0; i < names.length; i++) {
        String name = names.elementAt(i);
        print('$i: $name');
    }

}
```

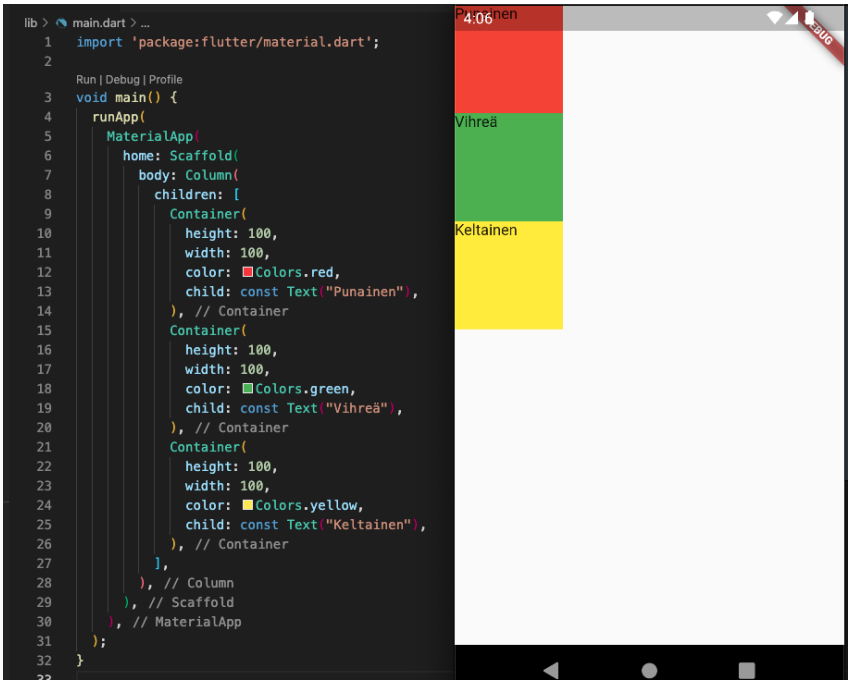
KUVA 1. Dart-koodi muistuttaa hyvin paljon Javaa

2.3 Flutter-komponentit

Flutterissa käyttöliittymä rakennetaan käyttämällä komponentteja (widget), jotka ovat Reactin inspiroima idea. Komponentit kertovat, miltä käyttöliittymän tulee näyttää nykyisen sovelluksen tilan perusteella. Kun sovelluksen tila muuttuu, ohjelmistokehys vertaa edellistä sovelluksen tilaa uuteen sovelluksen tilaan ja piirtää komponentin uudelleen muuttamalla mahdollisimman vähän komponenttipuuta (widget tree). Näin sovellukset pysyvät nopeina, eikä komponentteja joiden tila pysyy samana, piirretä turhaan uudelleen. Yksinkertaisimmillaan Flutter-sovellus kutsuu runApp-funktiota, ja antaa sille komponentin parametrina (kuva 2). runApp-funktion ottama parametri tulee olemaan komponenttipuun ensimmäinen komponentti. Kuvan 2 komponenttipuussa on kaksi komponenttia: Center-komponentti ja sen alapuolella Text-komponentti. Flutter pakottaa ensimmäisen, Center-komponentin, peittämään koko alueen. Center-komponentti ottaa parametriksi child-komponentin, joka nimensä mukaisesti tulee näkyville ruudun keskelle. Kuvan 2 esimerkissä on määritelty vielä erikseen textDirection. Yleensä juurikomponenttina käytetään MaterialApp-komponenttia, joka hoitaa tyylittelyt, eikä textDirection parametria tarvitse erikseen määritellä. (Flutter 2023b.)

KUVA 2. Minimaalinen Flutter sovellus (Flutter 2023b.)

Flutterissa on muutama peruskomponentti, mitä tarvitaan lähes jokaisessa sovelluksessa. Nämä komponentit ovat Text-komponentti, joka näyttää tekstiä sovelluksessa. Tekstiä voi tyylitellä muun muassa style-parametrillä. Row- ja Column-komponentit järjestävät sille syötetyt komponentit vierekkäin sekä horisontaalisessa suunnassa (Row) että vertikaalisessa suunnassa (Column). Nämä komponentit pohjautuvat CSS:n flexbox-ulkoasuun. Stack-komponttti järjestää sille syötetyt komponentit päällekkäin, määritellyssä järjestyksessä. Stack-komponentin kanssa voi käyttää esimerkiksi Positioned-komponenttia, jonka lapsikomponentille voi antaa tarkan sijainnin ruudulta. Stack-komponentti pohjautuu CSS:n absolute-ulkoasuun. Container-komponentti (kuva 3) luo neliön muotoisen laatikon. Laatikkoa voi muotoilla haluamakseen näköiseksi määrittelemällä muun muassa värin, reunat tai varjon. Containerille voi antaa myös marginaalit, täytteet (padding) ja muita kokoon liittyviä rajoitteita.

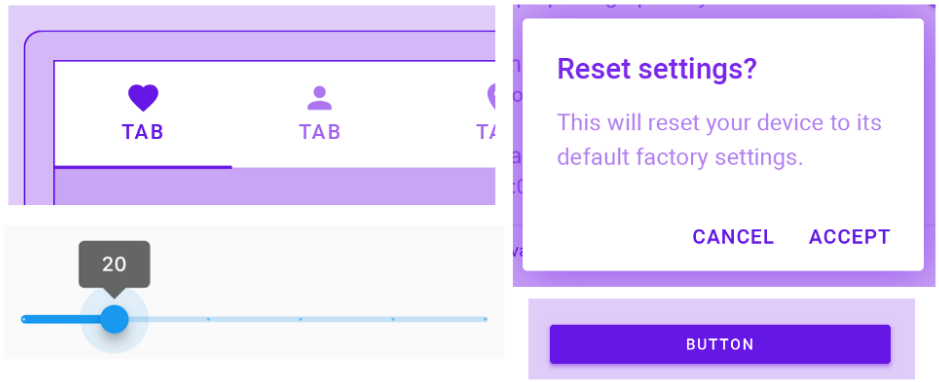


KUVA 3. Flutter sovellus käyttäen Column, Container ja Text komponentteja

2.4 Material Design komponentit

Material Design on Googlen kehittämä systeemi, joka auttaa kehittäjiä rakentamaan korkealaatuisia sovelluksia kaikille alustoille. Sen tavoitteena on tuoda samanlainen käyttökokemus riippumatta siitä, millä laitteella sovellusta käytetään. Material Design sisältää infoa muun muassa siitä, mitä värejä ja fontteja sovelluksessa tulisi käyttää. Tämän lisäksi Material Design kertoo, millä tavalla erilaiset näkymät tulisi suunnitella, eikä pelkästään miltä niiden pitäisi näyttää. Google ylläpitää Material Designille kattavaa dokumentaatiota sivuillaan. (Designers 2020.)

Flutterissa on myös paljon valmiita komponentteja, jotka noudattavat Material Design -ohjeita. Käyttämällä pelkästään näitä komponentteja voi jo saada valmiin, ammattimaisen ulkoasun pienellä vaivalla. Näihin komponentteihin kuuluvat muun muassa ElevatedButton, Slider, TapBar ja AlertDialog (kuva 4). (Flutter 2023c.)



KUVA 4. Flutterin valmiit ElevatedButton, Slider, TapBar ja AlertDialog komponentit

2.5 Koodieditori ja emulaattori

Kaksi suosituinta koodieditoria Flutter-kehitykseen ovat Android Studio ja Visual Studio Code. Android Studio on Googlen suosittelema koodieditori Android-sovelluskehitykseen, josta löytyy valmiiksi valtava määrä ominaisuuksia. Näistä kahdesta suositumpi on kuitenkin Visual Studio Code, jota tullaan käyttämään tässäkin projektissa. Visual Studio Code on ilmainen, kevyt koodieditori ja asentamalla siihen Flutter ja Dart -lisäosat se sopii erinomaisesti Flutter kehitykseen. (Lewis Cianci 2022.)

Emulaattori on työkalu, joka simuloi laitetta, tässä tapauksessa Android laitetta tietokoneella. Emulaattorien avulla sovellusta on helppo kehittää ja testata. Emulaattoriin voi valita puhelimen mallin ja Android version, mikä mahdollistaa sovelluksen testauksen useilla eri laitteilla. Testaaminen emulaattorissa on myös useasti nopeampaa kuin fyysisellä laitteella. (Android for Developers 2022.)

2.6 Tietojen tallentaminen

Sovelluksessa tullaan käyttämään Hiveä. Hive on Flutterille kehitetty relaatiomallista poikkeava tietokanta. Sen käyttö on erittäin yksinkertaista, ja se tarjoaa erittäin nopeita luku- ja

kirjoitusnopeuksia (kuva 5). Verrattuna SharedPreferences-kirjastoon, joka on Flutter-tiimin kehittämä relaatiomallista poikkeava tietokantakirjasto, avainarvo parien tallentaminen Hiven avulla on moninkertaisesti nopeampaa. Arvojen lukemisessa ei kuitenkaan ole isoa eroa. (Choi Simon 2019.)

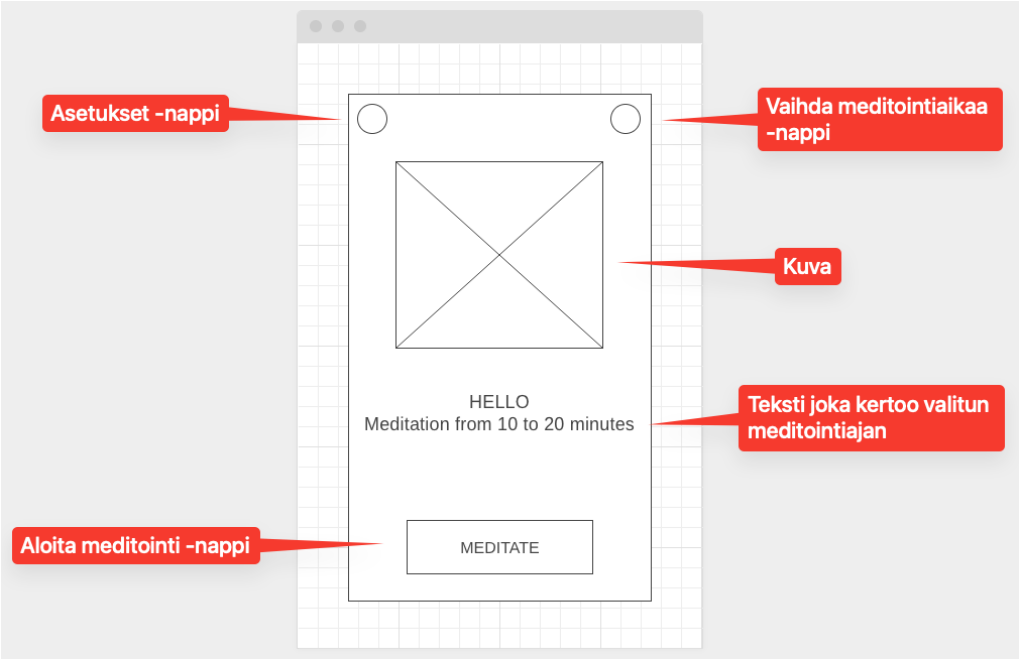
KUVA 5. Hiven nopeus verrattuna muihin relaatiomallista poikkeaviin tietokantoihin (Choi Simon 2019.)

3 SOVELLUKSEN KEHITTÄMISPROSESSI

Opinnäytetyön aiheena oli kehittää meditoinnissa avustava yksinkertainen ja helposti käytettävä sovellus. Sovellus tulee ensisijaisesti omaan käyttöön, joten mitään ennalta annettuja määrittelyjä sovellukselle ei ollut, vaan kokonaisuuteen kuului sovelluksen suunnittelu. Sovellus oli tarkoitus julkaista myös Google Play -sovelluskaupassa, mutta tämä jätettiin raportista pois ajan puutteen vuoksi.

3.1 Suunnittelu

Sovellus toimii siten, että käyttäjä painaa etusivulta (kuva 6) ”meditate”-painiketta, joka aloittaa meditoinnin. Koko sovelluksen ydinidea, jota en löytänyt mistään muusta valmiista sovelluksesta, on se, että meditointiaika on joka kerta eri – käyttäjä voi valita asetuksista minimiajan sekä maksimiajan. Kun meditointi alkaa, sovellus arpoo jonkin ajan näiden valmiiksi määriteltyjen aikojen väliltä.



KUVA 6: Rautalankamalli sovelluksen etusivusta

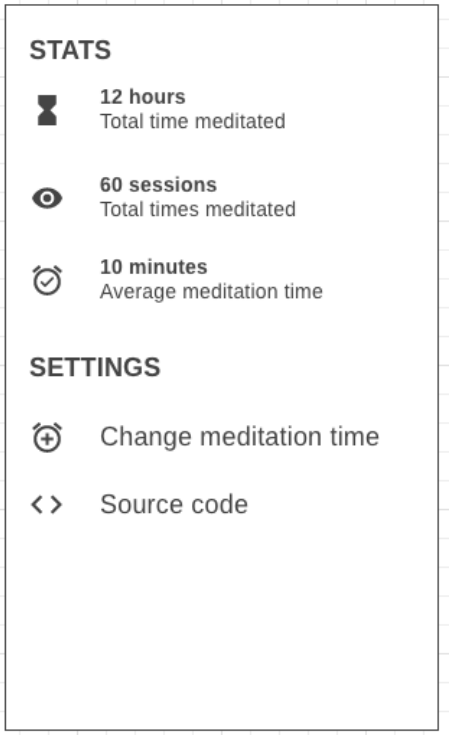
Itse meditointinäkymä (kuva 7) on erittäin yksinkertainen. Keskellä näkyy kulunut aika sekä ympyrän muotoinen indikaattori siitä, kuinka paljon aikaa on vielä jäljellä. Painamalla puhelimen takaisinnappia meditoinnin voi tarvittaessa laittaa tauolle tai keskeyttää kokonaan. En näe syytä käyttää tähän näkymään isoa osaa ajasta, sillä meditoidessa silmiä pidetään kiinni, eikä näkymää nähdä kauaa.

KUVA 7: Rautalankamalli meditointinäkymästä

Kun meditointi on loppunut, kuuluu lyhyt merkkiääni tiedoksi, että meditoinnin voi lopettaa. Merkkiääni ei kuitenkaan jää soimaan pidemmäksi aikaa, vaan häipyy nopeasti pois, jotta käyttäjä voi halutessaan jatkaa meditointia. Äänimerkin tuottamiseen voitaisiin hyödyntää puhelimen omaa hälytystä, mutta huonona puolena siinä on, että hälytyksen soidessa sovellus hälyttäisi herätyskellon tapaan niin kauan, että käyttäjä sammuttaa äänen. Tämä estää meditoinnin jatkamisen ilman taukoa, joten sovelluksessa tullaan käyttämään omaa merkkiääntä. Merkkiäänen on myös oltava tyydyttävä ja mukava, samankaltainen kuin jossain videopelissä saattaisi kuulua käyttäjän edistyessä seuraavalle tasolle.

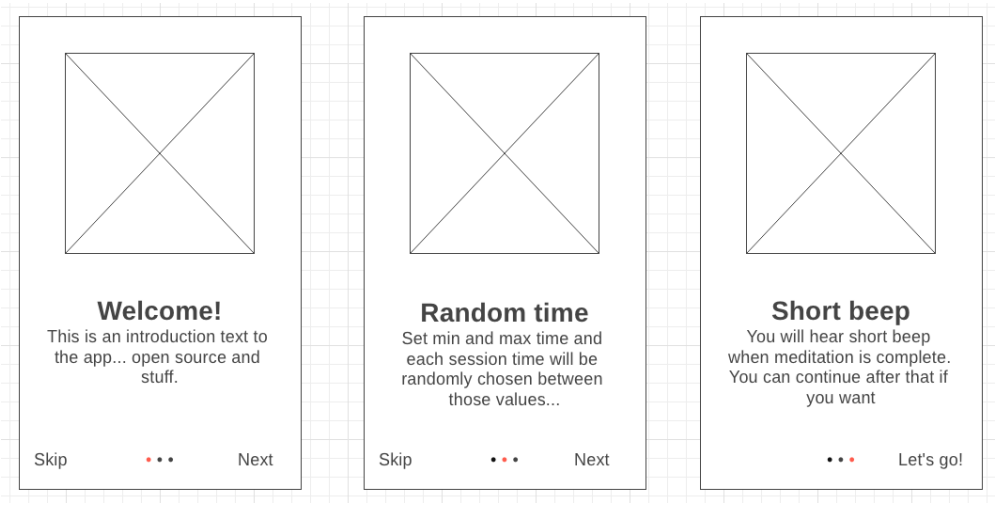
Sovellus kerää myös dataa käyttäjän meditoinneista. Asetukset-sivulta (kuva 8) käyttäjä voi nähdä kuinka kauan on meditoinut yhteensä, kuinka monta kertaa yhteensä sekä kuinka pitkä keskiverto

meditointikerta on ollut. Asetukset sivulta löytyy myös tarvittavat asetukset sekä linkki GitHub-repositorioon.



KUVA 8: Mock-up asetukset-sivusta

Viimeisimpänä sovelluksessa tulee olla selkeä ja helposti käytettävä käyttöliittymä. Ei turhia painikkeita tai tekstiä, vaan kaikilla elementeillä tulee olla selkeä tarkoitus. Siirtyessä näkymältä toiselle tulee olla sulava animaatio. Sovelluksen käynnistäessä ensimmäisen kerran käyttäjälle näkyy lyhyt opastus sovelluksen ominaisuuksista (kuva 9). Opastus on jaettu kolmelle eri sivulle, ettei ruudulle tule kerralla liikaa tekstiä.



KUVA 9: Rautalankamalli opastussivuista

3.2 Flutter-projektin luominen

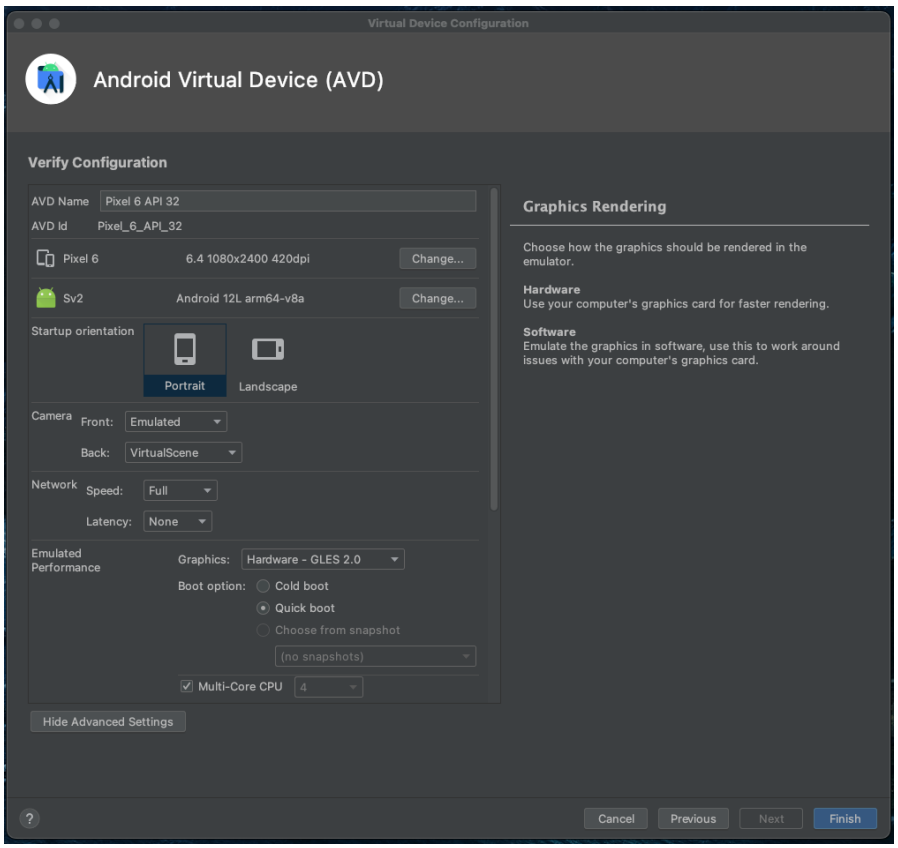
Sovelluksen luomiseen tarvittiin Flutter ohjelmistokehityspaketti (software development kit), jonka mukana tuli myös uusin Dart ohjelmistokehityspaketti. Tämän lisäksi käytettiin Visual Studio Code koodieditoria, Git-versionhallintajärjestelmää sekä Android-emulaattoria. Käytin Visual Studio Codea, koska se oli itselle tutumpi koodieditori kuin Android Studio. Sovellusta voi testata myös omalla iOS- tai Android-älypuhelimella, mutta käytin tässä projektissa emulaattoria. Näiden lisäksi Apple Silicon Mac-käyttäjät tarvitsevat Rosettan, mikä mahdollistaa Intel-prosessoreille suunniteltujen ohjelmien suorittamista. (Apple 2022.)

Flutter ohjelmistokehityspaketin, Visual Studio Coden sekä Rosettan voi asentaa ajamalla muutaman komennon terminaalissa (kuva 10). Näiden lisäksi Android Studio tulee asentaa, jotta Android emulaattorin saa käyttöön. Emulaattorin voi luoda myös ilman Android Studiota, mutta Android Studio helpottaa prosessia huomattavasti. Git-versionhallintajärjestelmä on jo valmiiksi asennettuna MacOS-tietokoneissa.

```
# Install Rosetta
sudo softwareupdate --install-rosetta --agree-to-license

# Get the Flutter SDK
cd ~/development
wget https://storage.googleapis.com/flutter_infra_release/releases/stable/macos
/flutter_macos_arm64_3.7.0-stable.zip
unzip ~/Downloads/flutter_macos_3.7.0-stable.zip

# Add Flutter to path
export PATH="$PATH:`pwd`/flutter/bin"

# Install VSCode & Android Studio
brew install --cask visual-studio-code
brew install --cask android-studio
```

KUVA 10. Rosettan, Flutter ohjelmistokehityspaketin, Visual Studio Coden ja Android Studion asennus terminaalissa

Tämän jälkeen Android Studiossa voi luoda uuden Android emulaattorin. Tämä tapahtuu menemällä AVD-valikkoon ja valitsemalla ”Create Virtual Device…”. Emulaattoriin voi valita haluamansa Android laitteen ja Android version. Viimeisellä sivulla (kuva 11) ennen emulaattorin luomista pitää vielä valita ”Hardware – GLES 2.0” asetus, jotta Flutter sovellukset pyörivät emulaattorilla parhaiten.



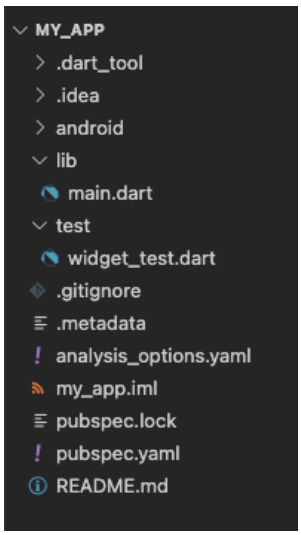
KUVA 11. Android emulaattorin luominen. Laitteeksi on valittu Pixel 6, Android versioksi 12L ja grafiikka-asetukseksi ”Hardware – GLES 2.0”

Tarvittavien työkalujen asennuksen jälkeen uusi projekti luodaan komennolla ”flutter create [sovelluksen nimi]”. Projektia luodessa ei tarvitse valita mitään asetuksia, vaan komento tekee projektin uuteen kansioon. Projektin voi aukaista Visual Studio Code -koodieditorissa komennolla ”code [sovelluksen nimi]” (kuva 12).

```
# Create Flutter project
flutter create my_app

# Open app in VSCode
code my_app
```

KUVA 12. Flutter projektin luominen terminaalissa.

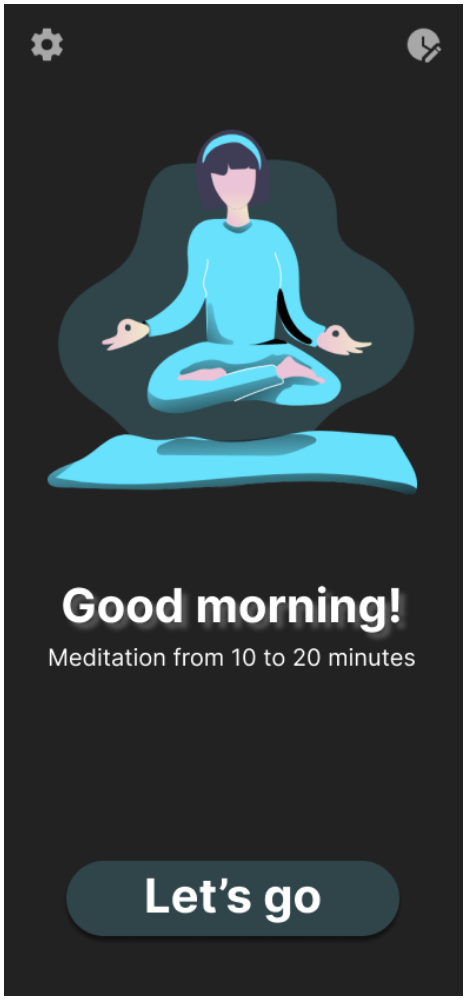
Projektin auetessa ensimmäistä kertaa Visual Studio Code pyytää käyttäjää asentamaan Flutter- ja Dart-lisäosat. Koodieditorissa pääsee näkemään mitä tiedostoja ja kansioita sovellukselle luotiin valmiiksi (kuva 13). Näistä tiedostoista main.dart on kaikista tärkein, sillä sovellus käynnistyy aina kutsumalla kyseisen tiedoston main-funktiota. Test-kansioon voi tarvittaessa kirjoittaa testejä, ja pubspec.yaml tiedostoon voi määritellä yleisiä projektin asetuksia sekä lisätä kolmannen osapuolen Dart kirjastoja.



KUVA 13. Flutter-projektin tiedostorakenne.

3.3 Etusivun luominen

Sovellukseen tehtiin aluksi etusivu. Aloitin etusivun tekemisen lisäämällä siihen kuvan, joka löytyi manyipixels-sivustolta. Kyseisen sivun illustraatiota voi käyttää maksutta missä tahansa projektissa, mainitsematta kuvan alkuperää. Lisäsin sivulle myös alustavat tekstit painikkeen alareunaan sekä kaksi pientä painiketta ylänurkkiin, joista jatkossa pääsee asetukset- ja ajanvaihto-näkymille (kuva 14.) Tässä vaiheessa tyylittelyt ja värit olivat vielä kesken. Halusin keskittyä sovelluksen logiikan ohjelmointiin ja miettiä ulkoasua myöhemmin.



KUVA 14. Sovelluksen alustava etusivu, ilman toiminnallisuuksia

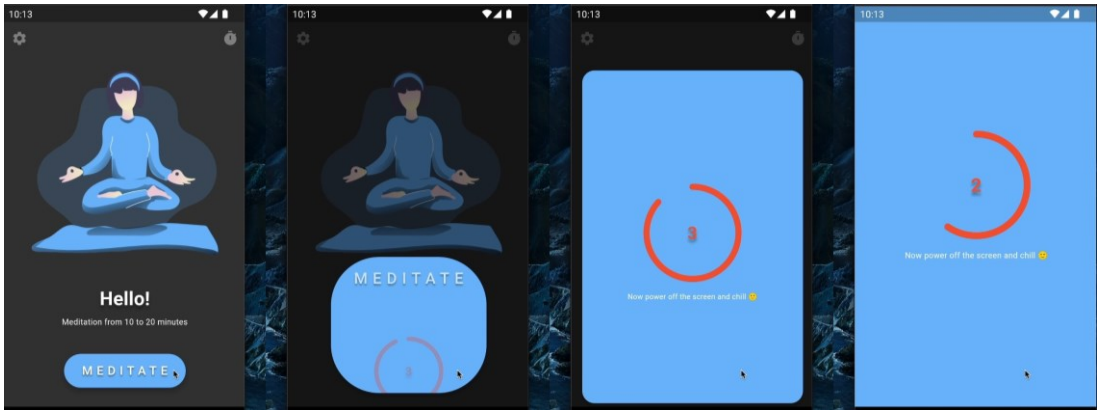
Tässä vaiheessa projektia oli käytössä ainoastaan Flutterin omia komponentteja (kuva 15). Ylhäällä olevan ikonit ovat osa Flutter Icons-pakettia, ja alhaalla oleva iso ”Let’s go”-painike on Flutterista löytyvä FloatingActionButton. Floating Action Button on osa Material Designia. Sitä on tarkoitus käyttää näkymän tärkeimpään, eniten käytettyyn toimintoon. (Material Design 2022.)

KUVA 15. Etusivun koodi, ilman toiminnallisuuksia

Myöhemmin Flutterin valmis FloatingActionButton-komponentti kuitenkin korvattiin omalla komponentilla (kuva 16), jotta meditointinäkymälle siirtymiseen saatiin sulava animaatio. Animaatio näyttää siltä, että kun painiketta painaa, painike laajenee koko ruudun kokoiseksi, ja toimii taustavärinä seuraavalle ruudulle (kuva 17). Animaatioon käytettiin Flutter-tiimin kehittämää animations-kirjastoa, mikä piti lisätä erikseen projektiin. Kirjastoja voi lisätä Flutter-projektiin määrittelemällä ne pubspec.yaml tiedostoon.

```dart
class CustomFAB extends StatelessWidget {
  const CustomFAB({super.key});

  @override
  Widget build(BuildContext context) {
    return Container(
      margin: const EdgeInsets.only(bottom: 12),
      child: OpenContainer(
        closedBuilder: (BuildContext ctx, VoidCallback openContainer) {
          // Animation doesn't look smooth with FloatingActionButton widget
          return const SizedBox(
            height: 50,
            width: 180,
            child: Center(
              child: Text(
                "MEDITATE",
                style: Styles.floatingButtonTextStyle,
              ), // Text
            ), // Center
          ); // SizedBox
        },
        closedShape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(99),
        ), // RoundedRectangleBorder
        closedColor: Styles.buttonColor,
        openColor: Styles.buttonColor,
        closedElevation: 6,
        openBuilder: (BuildContext ctx, VoidCallback _) {
          return const InProgressScreen();
        },
      ), // OpenContainer
    ); // Container
  }
}
```

KUVA 16. Oma Floating Action Button-komponentti



KUVA 17. Animaatio ”meditate”-painiketta painaessa

3.4 Meditointinäkymän luominen

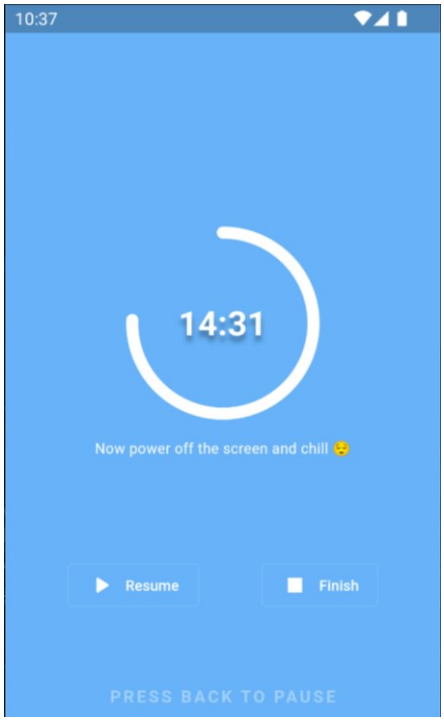
Meditointinäkymälle siirtyessä sovellus arpoo meditointiajan käyttäjän määrittelemien aikojen väliltä. Ennen kuin meditointi alkaa, ruudulla näkyy kolmen sekunnin punainen lähtölaskenta (kuva 17), ennen kuin meditoinnin ajastin lähtee käyntiin. Lähtölaskennan aikana käyttäjä voi painaa puhelimen takaisinpainiketta mikä peruuttaa meditoinnin. Kun lähtölaskenta on suoritettu loppuun, komponentti piilotetaan ja sen tilalla näytetään meditointiajastin-komponentti (kuva 18).

KUVA 18. hasStarted muuttujalla ohjataan sitä, näytetäänkö ruudulla MeditationTimer vai CountdownTimer-komponentti

Sekä lähtölaskentaan että medintointiajastimeen käytettiin circular_countdown_timer-kirjaston CircularCountDownTimer-komponenttia omilla tyylittelyillä (kuva 19). Meditoinnin aikana keskellä näkyy kulunut aika, sekä ympyrän muotoisesta indikaattorista voi päätellä suunnilleen kauanko aikaa on vielä jäljellä.

KUVA 19. MeditationTimer-komponentissa käytettiin circular_countdown_timer-kirjastoa

Meditoinnin aikana käyttäjä voi painaa puhelimen takaisinpainiketta, mikä keskeyttää meditoinnin. Näkyviin tulee kaksi uutta nappia: Resume ja Finish (kuva 20), jolla käyttäjä voi jatkaa meditointia tai keskeyttää sen kokonaan. Mikäli meditointi lopetetaan ennen kuin minuutti on kulunut, meditointia ei kirjata historiaan eikä sitä lasketa statistiikkaan mukaan. Statistiikasta lisää myöhemmin.



KUVA 20. Meditointinäkymä

Puhelimen takaisinpainikkeesta meditoinnin keskeyttäminen toteutettiin Flutterin omalla WillPopScope-komponentilla (kuva 21). Koodissa seurataan muuttujien avulla, onko meditointi käynnissä, ja jos on, niin takaisinpainiketta painaessa näytetään ”Resume ja ”Finish” -painikkeet. Normaalisti takaisinpainiketta painaessa sovellus navigoi taaksepäin. Mikäli meditointi on jo valmiiksi keskeytetty ja käyttäjä painaa uudelleen takaisinpainiketta, meditointi lopetetaan kokonaan ja navigoidaan etusivulle.

```
103
104   Future<bool> onWillPop() {
105     if (paused || !hasStarted || complete) {
106       return Future<bool>.value(true);
107     }
108
109     setState(() {
110       paused = true;
111       meditationController.pause();
112     });
113
114     return Future<bool>.value(false);
115   }
116
117   @override
118   Widget build(BuildContext context) {
119     return WillPopScope(
120       onWillPop: onWillPop,
121       child: Scaffold(
122         backgroundColor: Styles.buttonColor,
123         body: Stack(
124           children: [
125             SafeArea(
126               child: Center(
127                 child: Column(
128                   mainAxisAlignment: MainAxisAlignment.spaceBetween,
129                   children: [
```

KUVA 21. WillPopScope-komponentin käyttö meditointinäkymällä

Kun meditointi on suoritettu loppuun, käyttäjälle soitetaan merkkiääni, näytölle tulee onnitteluteksti ja ruudun ylälaidasta alkaa tippua serpentiiniä (kuva 22). Merkkiääni ladattiin Freesound-sivustolta, ja on täysin vapaasti käytettävä tiedosto.

KUVA 22. Serpentiini ja onnitteluteksti meditoinnin lopettaessa

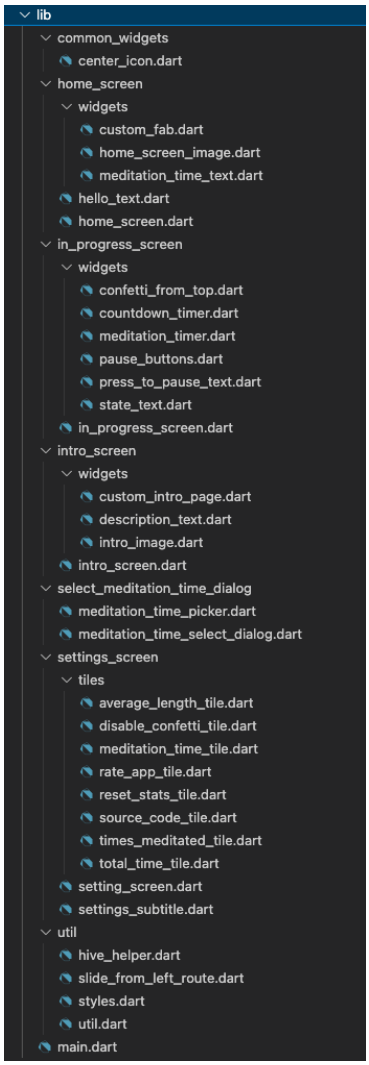
Serpentiinin näyttämiseen käytettiin confetti-kirjastoa (kuva 23). Komponenttiin pystyy määritellä paljon erilaisia asetuksia liittyen siihen miltä serpentiini näyttää, muun muassa kuinka paljon sitä on, mihin suuntaan se tippuu, kuinka nopeasti se tippuu ja minkä värisiä serpentiinit ovat. Kirjaston puutteellisen dokumentaation vuoksi serpentiini oli hieman haasteellista saada näyttämään hyvälle.

```
6   class ConfettiFromTop extends StatelessWidget {
7     final ConfettiController confettiController;
8
9     const ConfettiFromTop({super.key, required this.confettiController});
10
11    @override
12    Widget build(BuildContext context) {
13      return Align(
14        alignment: Alignment.topCenter,
15        child: ConfettiWidget(
16          confettiController: confettiController,
17          shouldLoop: true,
18          canvas: Size.infinite,
19
20          // Styling
21          blastDirection: pi / 2, // Top to bottom
22          numberOfParticles: 10,
23          minBlastForce: 1,
24          maxBlastForce: 10,
25          gravity: 0.09,
26        ), // ConfettiWidget
27      ); // Align
28    }
29  }
30
```

KUVA 23. Serpentiini-komponentin koodi omilla tyylittelyillä

3.5 Tiedostorakenne

Tässä vaiheessa projektia tiedostoja alkoi olla ja niin paljon, että päätin järjestellä tiedostorakenteen uusiksi. Tein lib-kansioon, missä kaikkien oleellisten tiedostojen tulee olla, useita alikansioita. Yksi näistä oli common_widgets mihin kuuluu kaikki yleiset, uudelleenkäytettävät komponentit. Util-kansioon tuli yleiset, useassa paikassa käytettävät funktiot joille ei löytynyt muuta loogista paikkaa. Näiden lisäksi jokaiselle näkymälle tehtiin oma kansio, minne tuli näkymän oma dart-tiedosto. Näiden näkymäkansioiden sisällä oli vielä widgets kansio (kuva 24) johon laitettiin näkymäkohtaiset komponentit, joita ei kuitenkaan haluttu pitää näkymän omassa tiedostossa. Komponentteja pilkottiin omiin tiedostoihin luettavuuden takia.



KUVA 24. Sovelluksen tiedostorakenne

Tiedostot järjestettiin tällä tavalla, jotta kaikki toisiinsa liittyvät oleelliset tiedostot löytyvät samasta paikasta. Jos muokkaan esimerkiksi etusivua, niin etusivuun liittyvät tiedostot on helppo paikantaa samasta kansiosta. Toinen yleinen tiedostorakenne Flutter-projekteissa on erotella kaikki komponentit yhteen kansioon ja kaikki näkymät yhteen kansioon. En kuitenkaan pidä tästä lähestymisestä sillä siinä laitetaan useita komponentteja, mitkä eivät liity mitenkään toisiinsa, samaan kansioon. Isossa projektissa tästä kansiosta tulee helposti sekava ja vaikeasti selattava.

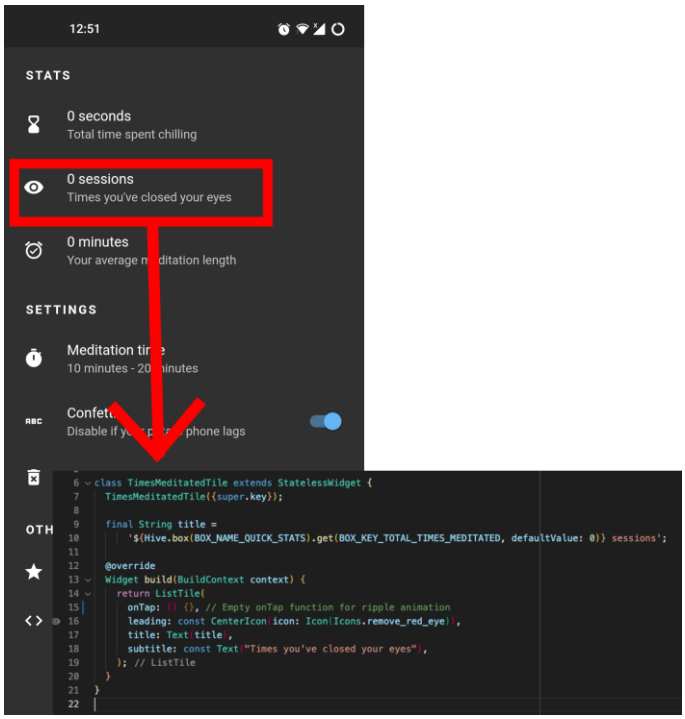
3.6 Asetukset-sivun luominen

Asetukset-sivulle laitettiin asetusten lisäksi statistiikkaa meditoinnista sekä painikkeet, joista käyttäjä pääsee suoraan sovelluksen Google Play -sivustolle sekä GitHub-repositorioon. Asetusten sekä statistiikan tallentamiseen käytettiin Hiveä. Hivessä avainarvo parit tallennetaan laatikkoihin, ja se pitää alustaa main.dart-tiedostossa olevassa main-funktiossa (kuva 25).

```
 9  void main() async {
10    await Hive.initFlutter();
11    await Hive.openBox(BOX_NAME_MEDITATION_TIME);
12    await Hive.openBox(BOX_NAME_QUICK_STATS);
13    await Hive.openBox(BOX_NAME_SETTINGS);
14
15    runApp(const MyApp());
16  }
```

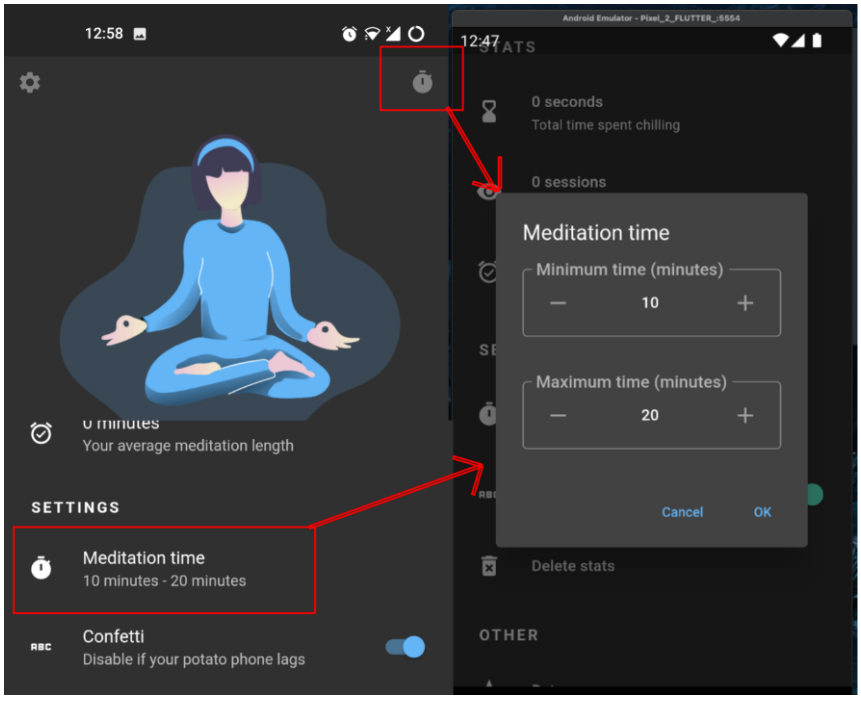
KUVA 25. Hiven alustus main-funktiossa

Pilkoin jokaisen yksittäisen asetuksen komponentin omaan tiedostoon. Yksittäisten asetuskomponenttien tekemiseen käytin Flutterin valmista ListTile-komponenttia (kuva 26). Asetuksista voi vaihtaa meditointiaikaa, ottaa serpentiinianimaation pois päältä sekä poistaa kaiken kerätyn statistiikan. Serpentiini-asetus laitettiin varalta vanhoja puhelimia varten, jotka eivät välttämättä jaksa suorittaa monimutkaisia animaatioita.



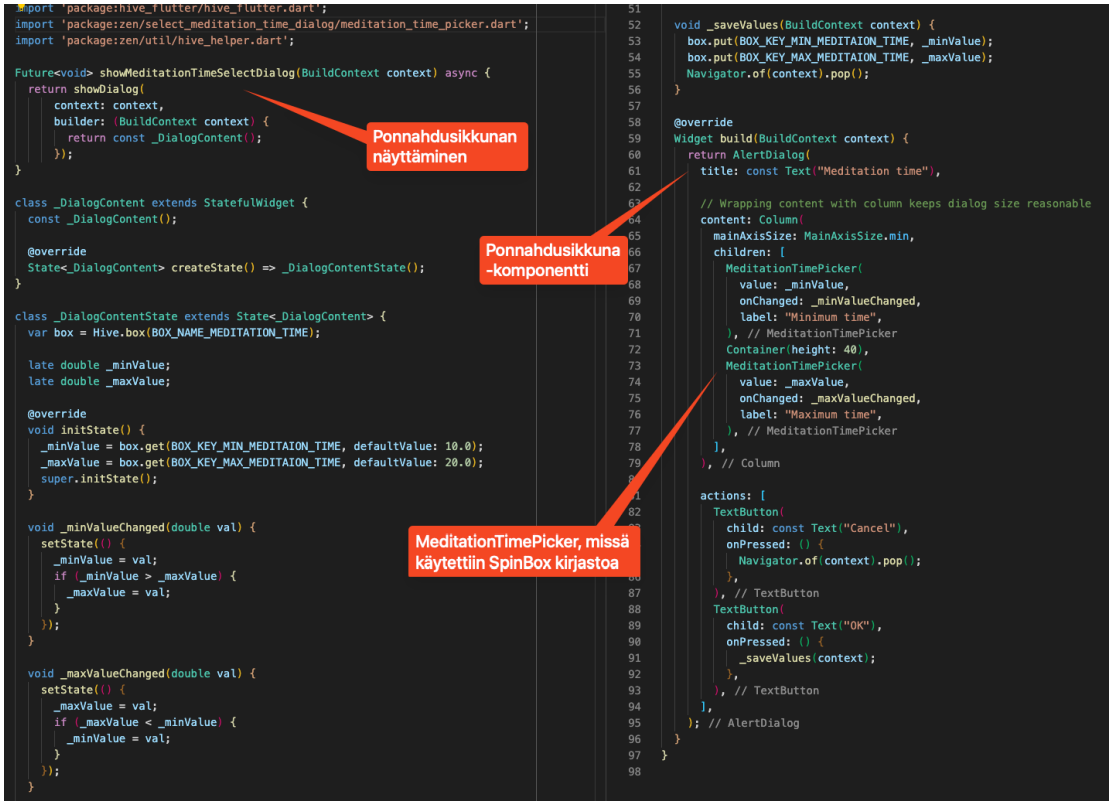
KUVA 26. Asetussivun komponentti, mikä näyttää kuinka monta kertaa käyttäjä on meditoinut

Meditation time-laattaa painaessa asiakkaalle avautuu ponnahdusikkuna (kuva 27), josta minimi- ja maksimiaikaa voi muuttaa. Tämä sama ponnahdusikkuna näytetään myös etusivun oikeassa yläkulmassa olevasta painikkeesta painaessa.



KUVA 27. Ponnahdusikkuna meditointiajan valitsemiseen

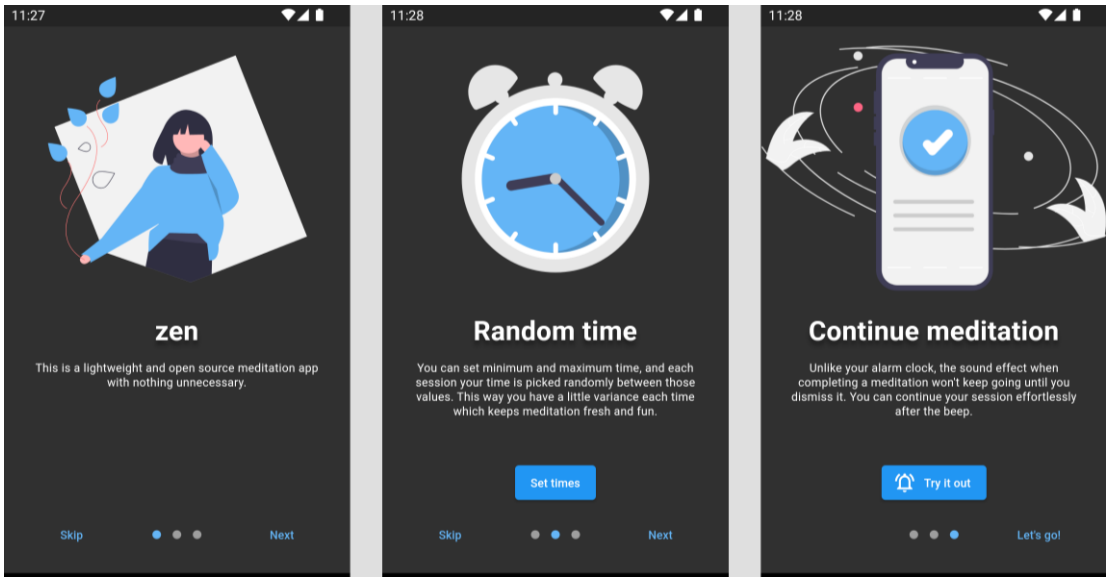
Ponnahdusikkunan näyttämiseen käytettiin showDialog-funktiota (kuva 28), jonka builder-parametrille annetaan haluttu komponentti. Aikojen valitsemiseen käytettiin valmista flutter_spinbox-kirjastoa. Ponnahdusikkunan sulkiessa valitut arvot tallennettiin sovelluksen muistiin, käyttäen Hive-kirjastoa.



KUVA 28. Ponnahdusikkunan koodi

3.7 Opastussivujen luominen

Opastussivun ohjeet jaettiin kolmeen osaan (kuva 29), jotta käyttäjälle ei tulisi kerralla näkyville liikaa tekstiä. Jokaisella osalla on oma tarkoitus. Ensimmäinen osa toivottaa käyttäjän tervatulleeksi. Toinen osa selittää sovelluksen idean, että käyttäjän pitää valita minimi- ja maksimiaika. Kolmas sivu kertoo, että meditointia voi halutessaan jatkaa merkkiäänen jälkeen.

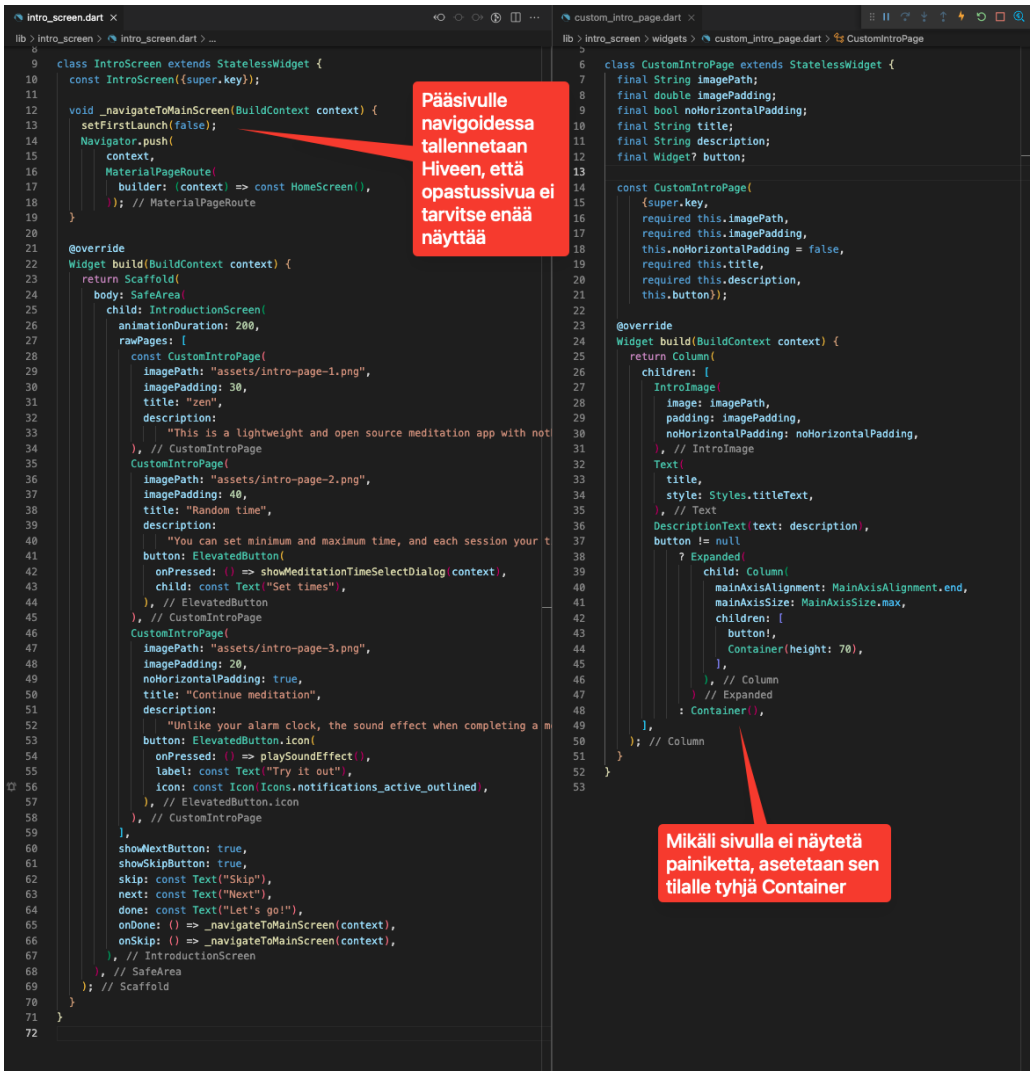


KUVA 29. Opastussivut vierekkäin

Myös opastussivuilla käytettiin vapaasti käytettäviä kuvia, tällä kertaa unDraw-sivustolta. Kuvat vaativat hieman muokkausta, että sain niistä haluamani näköisiä. Ensimmäistä kuvaa piti rajata, että kuvan henkilön sai kohdistettua horisontaalisessa suunnassa keskelle. Toisessa ja kolmannessa kuvassa oli alun perin henkilö, jonka poistin kokonaan sillä en keksinyt henkilöille sopivan värisiä vaatteita. Tummat vaatteet eivät erottuneet taustasta hyvin, ja siniset vaatteet veivät liikaa huomiota.

Toisen sivun ”Set times”-painike aktivoi saman dialogin, minkä saa esille asetukset-sivulta sekä kotisivulta (kuva 27). Viimeisellä sivulla ”Try it out”-painikkeesta voi kokeilla miltä merkkiääni kuulostaa, kun meditoinnin suorittaa loppuun. Opastussivut näytetään vain, kun sovellus käynnistetään ensimmäisen kerran. Opastuksen suoritettua tai ohitettua Skip-painikkeesta, Hiveen tallennetaan arvo, joka indikoi, että opastussivua ei tule enää näyttää.

Opastissivujen näyttämiseen käytin introduction_screen-kirjastoa. Opastussuvuille olisi voinut määritellä helposti sisällön käyttämällä kirjaston PageViewModel-komponenttia. En saanut kuitenkaan sivun 2 ja 3 painikkeita näkymään oikeassa kohdassa tämän avulla, joten kirjoitin oman CustomIntroPage-komponentin (kuva 30).



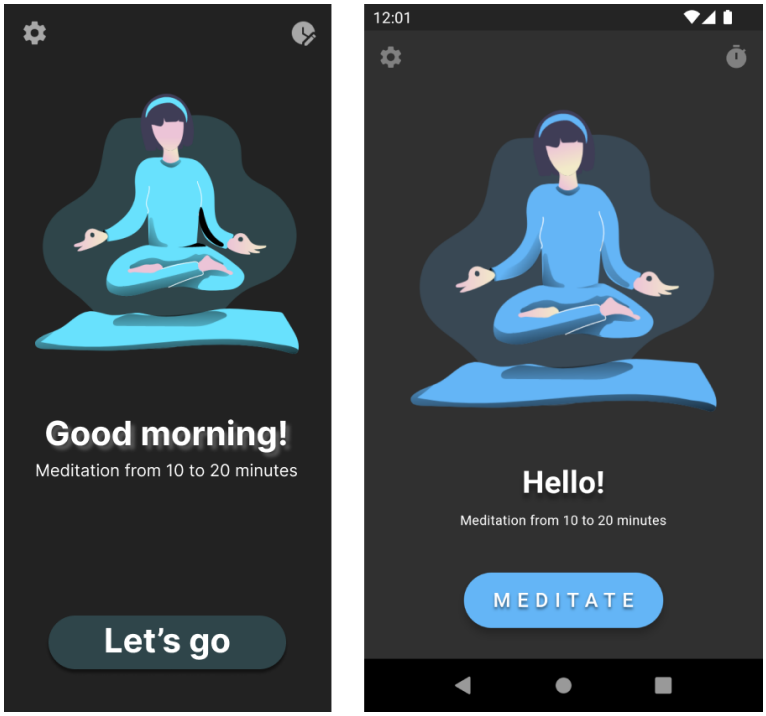
KUVA 30. Opastissivujen koodi käyttäen introduction_screen kirjastoa, sekä omaa CustomIntroPage-komponenttia

3.8 Värimaailman valitseminen

Päätin käyttää sovelluksessa tummaa teemaa, joka näyttää mielestäni vaaleaa teemaa paremmalta. Lisäksi käytän sovellusta pääsääntöisesti aamuisin. Tummat värit eivät rasita silmiä niin paljoa. Tumman teeman määrittely Flutterissa on erittäin helppoa (kuva 31). Parempi vaihtoehto olisi ollut noudattaa puhelimen asetuksissa määriteltyä väriteemaa. Tällä tavoin vaaleaa teemaa käyttäjät näkisivät sovelluksen vaalealla teemalla, ja tumman teeman käyttäjät näkisivät sovelluksen tummalla teemalla. En kuitenkaan toteuttanut tätä, sillä se olisi vaatinut värien miettimistä molemmille teemoille, sekä ylimääräistä logiikkaa sovellukseen.

KUVA 31. Tumman teeman määritteleminen Flutterissa

Sovelluksen pääväriksi oli aluksi tarkoitus tulla punainen. Päädyin kuitenkin lopulta siniseen väriin, sillä tutkimusten mukaan sinisellä värillä on rentouttavia ja rauhoittavia vaikutuksia. Koska meditointi on rentouttava ja rauhoittava aktiviteetti, ajattelin että sininen väri sopii sovellukseen punaista väriä paremmin. Sovelluksen alkuvaiheessa käytin turkoosia väriä. Sovelluksen edetessä ja asiaa miettiessä päädyin kuitenkin hieman tummempaan, vähemmän huomiota herättävään siniseen (kuva 32). (Al-Ayash Aseel, Kane Robert T., Smith Dianne & Green-Armytage Paul 2015.)



KUVA 32. Etusivun ulkoasu alkuvaiheessa verrattuna lopputulokseen.

4 POHDINTA

Suunnitteluvaihe projektissa jäi hieman lyhyeksi, ja tarkempi suunnitelma olisi voinut helpottaa sovelluksen toteuttamista. Tästä huolimatta sovelluksesta tuli hyvä, tiukasta aikataulusta huolimatta. Olen erittäin tyytyväinen lopputulokseen; sovellus näyttää hyvältä, pyörii sulavasti ja sisältää kivoja animaatioita. Myös opastus- ja päänäkymällä näkyvät kuvat lisäävät sovellukseen persoonallisuutta.

Sovelluksesta jätettiin joitain ominaisuuksia, kuten tarkempi statistiikan näyttäminen, pois kokonaan ajanpuutteen vuoksi. Sovelluksen kehittämistä on kuitenkin helppo jatkaa tästä eteenpäin ja lisätä uusia ominaisuuksia. Opinnäytetyön piti myös sisältää, kuinka sovellus julkaistaan Google Play -sovelluskaupassa. Tämäkin jäi raportista pois ajanpuutteen takia.

Vaikka olin käyttänyt Flutteria joskus aiemmin, vuosien tauon jälkeen se vaati hieman uudelleenopettelua. Projektia tehdessä muistin, kuinka hyvä kehittäjäkokemus Flutterissa on. Kehittäjä voi keskittyä täysin sovelluksen ohjelmointiin, kaikki toimii sulavasti ja valmiit Material komponentit tekevät ulkoasun toteuttamisesta erittäin vaivatonta.

Opinnäytetyön aikana ei ilmennyt isoja ohjelmointiin liittyviä ongelmia. Pääosin ohjelmointi sujui vaivatta ja edistyi ripeästi. Haasteita kuitenkin oli, esimerkiksi värimaailman valitseminen oli hankalaa ja vei paljon enemmän aikaa, mitä olin varannut. Tunnistan mobiilisovelluksista yleensä isoimmat suunnitteluvirheet, jotka huonontavat käyttäjäkokemusta. Hyvän ulkoasun suunnittelu, jonka käyttö on vaivatonta, kuitenkin on paljon haastavampaa kuin toisen ulkoasun analysointi. Tämä tietenkin paranee kokemuksen ja opiskelun myötä. Jatkossa aion käyttää enemmän aikaa ulkoasun suunnitteluun jo rautalankamalleja piirtäessä.

Takerruin myös sovellusta tehdessä liikaa yksityiskohtiin. Kun sovelluksen on suunnitellut itse, ja se tulee omaan käyttöön, takertuu helposti epäolennaisiin asioihin. Esimerkiksi meditoinnin päättyessä näkyvä serpentiinianimaatio vei paljon enemmän aikaa kuin siihen olisi ollut järkevä käyttää. Toisaalta nämä pienet, jopa turhat yksityiskohdat tekevät sovelluksesta uniikin ja loppuun hiotun oloisen.

Vaikka itse sovelluksen kehitys sujui pääpiirteittäin hyvin, niin vaikein osuus oli raportin kirjottaminen. Projektia tehdessä en ottanut tarpeeksi kuvakaappauksia enkä kirjoittanut omia ajatuksia muistiin, mikä teki raportin kirjoittamisesta entistä haastavampaa. En myöskään osannut ottaa huomioon, kuinka paljon asioita tulee selostaa raportissa. Jos tämän olisi osannut ottaa huomioon jo sovellusta kehittäessä, olisi matkan varrella raportoinut enemmän, mikä olisi helpottanut tämän raportin kirjoittamista.

LÄHTEET

Al-Ayash Aseel, Kane Robert T., Smith Dianne & Green-Armytage Paul 2015. The influence of color on student emotion, heart rate and performance in learning environments. Wiley Online Library. Hakupäivä 28.1.2023. https://onlinelibrary.wiley.com/doi/10.1002/col.21949.

Android for Developers 2022. Run apps on the Android Emulator. Android for Developers. Hakupäivä 3.2.2023. https://developer.android.com/studio/run/emulator#get-started.

Apple 2022. If you need to install Rosetta on your Mac. Apple. Hakupäivä 26.1.2023. https://support.apple.com/en-us/HT211861.

Belokrylova Alexander 2022. Why And How Java Continues To Be One Of The Most Popular Enterprise Coding Languages. Forbes. Hakupäivä 24.1.2023. https://www.forbes.com/sites/forbestechcouncil/2022/04/06/why-and-how-java-continues-to-be-one-of-the-most-popular-enterprise-coding-languages/.

Canorea Elena 2022. What Is Kotlin and What Is It Used For? Plain Concepts. Hakupäivä 24.1.2023. https://www.plainconcepts.com/kotlin-android/.

Chapman Cameron 2020. Why Use Material Design? Weighing the Pros and Cons. Designers. Hakupäivä 25.1.2023. https://www.toptal.com/designers/ui/why-use-material-design.

Choi Simon 2019. README.md. GitHub. Hakupäivä 27.1.2023. https://github.com/hivedb/hive.

Cianci Lewis 2022. Best IDEs for Flutter in 2022. LogRocket. Hakupäivä 3.2.2023. https://blog.logrocket.com/best-ides-flutter-2022/.

Cleveroad 2020. Why to Use Flutter for Building Cross-Platform Apps? Cleveroad. Hakupäivä 25.1.2023. https://www.cleveroad.com/blog/why-use-flutter/.

Flutter 2023a. Supported platforms. Flutter. Hakupäivä 24.1.2023. https://docs.flutter.dev/development/tools/sdk/release-notes/supported-platforms.

Flutter 2023b. Introduction to widgets. Flutter. Hakupäivä 24.1.2023. https://docs.flutter.dev/development/ui/widgets-intro.

Flutter 2023c. Material Components widgets. Flutter. Hakupäivä 24.1.2023. https://docs.flutter.dev/development/ui/widgets/material.

harkiran78 2022. Top Programming Languages for Android App Development. GeeksforGeeks. Hakupäivä 24.1.2023. https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/.

Javatpoint 2020. What is Dart Programming. Javatpoint. Hakupäivä 25.1.2023. https://www.javatpoint.com/flutter-dart-programming.

Material Design 2022. Buttons: floating action button. Material Design. Hakupäivä 27.1.2023. https://m2.material.io/components/buttons-floating-action-button.

Mohit Joshi 2023. Flutter vs React Native: A Comparison. BrowserStack. Hakupäivä 24.1.2023. https://www.browserstack.com/guide/flutter-vs-react-native.

Petrova Katerina 2022. The Six Most Popular Cross-Platform App Development Frameworks. Kotlin. Hakupäivä 24.1.2023. https://kotlinlang.org/docs/cross-platform-frameworks.html.

StatCounter 2023. Mobile Operating System Market Share Worldwide. StatCounter. Hakupäivä 5.2.2023. https://gs.statcounter.com/os-market-share/mobile/worldwide.