

FLUTTER SOVELLUSKEHITYKSESSÄ



Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus

kevät, 2023

Ville Lensu

Tietojenkäsittelyn koulutus

Tiivistelmä

Tekijä Ville Lensu

Vuosi 2023

Työn nimi Flutter sovelluskehityksessä

Ohjaaja Lasse Seppänen

TIIVISTELMÄ

Opinnäytetyön aiheena oli tutustua Flutter-ohjelmistonkehityspakettiin ja kehittää sitä käyttäen yksinkertainen mobiilisovellus, joka sisältää mobiilisovelluksissa usein käytettyjä elementtejä, kuten navigaation. Tavoitteena oli kartuttaa kokemusta ohjelmoinnista työelämää varten sekä samalla oppia käyttämään Flutteria ja ohjelmoimaan Dart-ohjelmointikielellä.

Opinnäytetyön tietopohja sisältää kaksi lukua, joista ensimmäisessä kerrotaan yleistä tietoa mobiilikehityksestä, kuten erilaisista mobiilisovellustyypeistä ja kahdesta suurimmasta mobiilikäyttöjärjestelmästä. Toinen luku käsittelee tarkemmin Flutteria; sen taustalla olevaa innovaatiota, sillä tehtyjä sovelluksia ja sen käyttämää Dart-ohjelmointikieltä. Opinnäytetyö on tyypiltään toiminnallinen ja sen pääpaino on tehdyn sovelluksen kehitysprosessissa.

Tehdyn kehitystyön tuloksena voidaan todeta, että Flutter on toimiva ohjelmistonkehityspaketti mobiilisovellusten tekemiseen, mutta aluksi sen käyttäminen voi olla haastavaa. Aikaisempi kokemus ohjelmoinnista on hyödyksi, ennen Flutterin käyttöönottoa tai Dart-ohjelmointikieleen tutustumista. Kehitystyössä saavutettiin asetetut tavoitteet.

Avainsanat Flutter, Mobiili, Sovellus, Kehitys

Sivut 31 sivua ja liitteitä 1 sivu

Degree Programme in Business Information Technology

Abstract

Author Ville Lensu

Year 2023

Subject Flutter in App Development

Supervisor Lasse Seppänen

ABSTRACT

The purpose of this thesis was to get acquainted with Flutter software development kit and use it to make a simple mobile application, that includes some common elements used in many mobile applications, such as navigation. The aim of this thesis was to get more programming experience for future, and at the same time to learn how to use Flutter and the Dart programming language.

The thesis has two chapters that contain background information about the development project. First of these two chapters has general knowledge about mobile development such as descriptions about different types of mobile applications and the two largest mobile operating systems. The second of these chapters takes a closer look at the Flutter itself; the innovation behind it, some known applications made with it and the Dart programming language it uses. The type of the thesis is practical and its focus centers around the process of making the mobile application using Flutter.

Based on the results of the app development process, Flutter is quite well functioning software development kit to be used in the making of mobile applications, although, using it can be difficult at first. Previous experience in programming is recommended, before using Flutter, or learning Dart programming language. The objectives set for the development work were completed.

Keywords Flutter, Mobile, Application, Development

Pages 31 pages and appendices 1 page

Sanasto

Flutter	Googlen kehittämä avoimen lähdekoodin mobiili ohjelmistonkehityspaketti (SDK).
Dart	Ohjelmointikieli, jota käytetään Flutterissa.
SDK	Software Development Kit eli ohjelmistonkehityspaketti. Sisältää valmiita työkaluja sovelluksen kehittämiseen.
UI	User Interface eli käyttöliittymä. Laitteen tai ohjelman osa, josta käyttäjä käyttää laitetta tai ohjelmaa. Esimerkiksi älypuhelimien ruutu.
Sovellus	Suoritettava ohjelma tai ohjelmien kokonaisuus. Esimerkiksi Skype, Whatsapp tai erilaiset pelit.
Widget	Widget tai pienoishjelma, jota usein ei tarvitse erikseen käynnistää kuten sovellusta. Esimerkiksi älypuhelimien kello, kalenteri ja sää -toiminnot. Flutterissa käsite on laajempi.
Avoin lähdekoodi	Lähdekoodi, joka on vapaasti saatavilla, muokattavissa ja uudelleenjaettavana.
Natiivi	Termi ohjelmille, jotka on suunniteltu alun perin toimimaan vain tietyllä käyttöjärjestelmällä.
Mobiililaitte	Pienikokoiset ja mukana kannettavat tietokonelaitteet, joissa on käyttöjärjestelmä. Esimerkiksi matkapuhelimet ja tabletit.
Android	Käyttöjärjestelmä, pääasiassa mobiililaitteille.

iOS	Applen kehittämä käyttöjärjestelmä pääasiassa mobiililaitteille.
VSCode	Visual Studio Code on editori, jota voi käyttää ohjelmointiin monella eri ohjelmointikiellä.
Emulaattori	Ohjelma, joka on suunniteltu käyttäytymään samalla tavalla kuin jokin toinen ohjelma tai laite. Esimerkiksi puhelimia imitoivat ohjelmat, joita käytetään mobiilikehityksessä.
Crossplatform	Monella alustalla tai käyttöjärjestelmällä toimiva ohjelma.
HTML	HyperText Markup Language, verkkosivujen määrittelykieli, jolla määritellään kuinka verkkoselaimet näyttävät verkkosivun.
HTML5	HTML:n viides versio. Sisältää parannuksia aikaisemmista versioista.
Objective-C	Yleiskäyttöinen olio-ohjelmointikieli, jota käytetään pääasiassa Applen käyttöjärjestelmien ja niille tehtyjen sovelluksien kehitystyössä.

Sisälllys

1	Johdanto	1
2	Mobiilikehitys	2
2.1	Mobiilisovellustyypit	2
2.2	Android ja iOS.....	4
3	Flutter	6
3.1	Yleistä Flutterista	6
3.2	Dart-ohjelmointikieli	6
3.3	Widgetit.....	7
3.4	Flutterilla tehtyjä sovelluksia	9
4	Kehittämistyön tarkoitus	10
4.1	Tausta.....	10
4.2	Tavoitteet	10
5	Sovelluksen kehitys	11
5.1	Kehitysympäristön rakentaminen.....	11
5.2	Kehitystyö.....	12
5.2.1	Navigaatio	14
5.2.2	Tiedostoon tallennus.....	17
5.2.3	Tiedostosta luku	23
6	Johtopäätökset ja pohdinta.....	26
6.1	Lopputulos.....	26
6.2	Tehdyn sovelluksen jatkokehitysmahdollisuuksia	27
7	Yhteenveto	28
	Lähteet.....	29

**Kuvat, ohjelmakoodit ja taulukot (nämä pitää päivittää kuten sisällyskin,
poista tarpeettomat)**

Kuva 1 Kaavio statejen käytöstä datan mukaan. (Flutter Statet, n.d.).....	8
Kuva 2 Uuden Flutter-projektin valmiin demon näkymä emulaattorissa.	12
Kuva 3 Emulaattorinäkymä ruudun alalaidassa sijaitsevasta navigaatiopalkista.	15
Kuva 4 Emulaattorinäkymä toisen sivun molemmasta välilehdestä ja niiden välisestä navigaatiosta.....	19
Kuva 5 Tallennusvälilehden perus-, kirjoitus- ja ilmoituskenttänäkymä emulaattorissa.	21
Kuva 6 Emulaattorinäkymä tiedoston sisällön lukemisesta.	25
Ohjelmakoodi 1 Luodun uuden sivun ohjelmakoodi.	13
Ohjelmakoodi 2 Navigaatiopalkin ohjelmakoodi käyttöliittymässä.....	15
Ohjelmakoodi 3 MyHomePage-luokassa olevan navigaatiopalkin logiikka.	16
Ohjelmakoodi 4 Ruutunäkymän asettaminen ja selectedIndex-muuttujan yhdistäminen sivuihin. Kommentoituna ensimmäinen toteutustapa.	17
Ohjelmakoodi 5 Välilehtivalikon ja sivun välilehtien sisällön ohjelmakoodi.....	18
Ohjelmakoodi 6 Tekstikentän toteutus ruutunäkymään.	20
Ohjelmakoodi 7 Tallennuspainikkeen toteutus.....	20

Ohjelmakoodi 8 Funktioita, joita käytetään tiedostoon tallentamisessa.	22
Ohjelmakoodi 9 TekstiStorage-luokkaan lisätty funktio, joka hakee tiedoston sisällön.	23
Ohjelmakoodi 10 Välilehden toteutus, jossa sijaitsee tiedoston sisällön näyttäminen.	23
Ohjelmakoodi 11 tiedostonSisalto-muuttujan arvon päivittävä funktio. Lisäksi initState-funktio, joka suorittaa päivitysfunktion sovelluksen käynnistyksen yhteydessä.....	24

Liitteet

Liite 1	Aineistonhallintasuunnitelma
---------	------------------------------

1 Johdanto

Opinnäytetyö käsittelee Flutter-ohjelmistonkehityspakettia ja sillä yksinkertaisen sovelluksen tekoa. Flutter on Googlen kehittämä avoimen lähdekoodin UI ohjelmistonkehityspaketti (SDK), josta löytyy valmiita työkaluja mobiilisovellusten kehitykseen. Flutterin käyttämä ohjelmointikieli on Dart. Opinnäytetyön täysi ymmärtäminen vaatii aikaisemmin hankittua tietopohjaa ohjelmoinnista.

Opinnäytetyö sisältää teoriaosuuden, joka pohjustaa käytännön osuudessa tehtävää sovelluskehitystä. Teoriaosuudessa annetaan taustatietoa mobiilikehityksestä ja Flutterista.

Käytännön osuus koostuu yksinkertaisen mobiilisovelluksen kehitysprosessista, sekä Flutterin käyttöönottamisesta ja tarvittavista asennuksista. Tarkastellaan kehittämistä Flutterilla, ja kuinka eri ominaisuuksien lisääminen sovellukseen tapahtuu.

Lopuksi kerrotaan tuloksista ja käytännön osuudessa kohdatuista ongelmista, sekä yleinen yhteenveto opinnäytetyöstä. Lopussa myös pohditaan, kuinka tehtyä sovellusta olisi mahdollista jatkokehittää.

Tutkimuskysymykset:

- Mikä on Flutter?
- Miten sovelluksen navigointi toteutetaan Flutterilla?
- Miten sovellukseen toteutetaan tiedostoon tallentaminen ja lukeminen Flutterilla?

2 Mobiilikehitys

Tässä luvussa käsitellään yleisesti mobiilisovelluksia. Luku alkaa lyhyellä selvityksellä mitä mobiilikehitys on. Sen jälkeen luvussa kerrotaan erilaisista yleisimmistä mobiilisovellustyypeistä ja jatketaan kertomalla kahdesta suurimmasta mobiilialustasta, Androidista ja iOS:ista, yleistä tietoa.

Mobiilikehitys on pähkinänkuoressa kehitystyötä mille tahansa mobiililaitteelle. Mobiililaitteiksi voidaan laskea monia erilaisia laitteita, joissa toimii mobiilikäyttöjärjestelmä, kuten Android tai iOS. Yleisimpiä mobiililaitteita ovat älypuhelimet ja tabletit, mutta myös erilaiset puettavat laitteet, kuten älykellot, ovat mobiililaitteita. Mobiilikehitys ei kuitenkaan tarkoita kehittämistä yksinomaan vain näille laitteille, vaan se voi olla pieni osa suurempaa kokonaisuutta, kuten verkkosivun tekemistä mobiililaitteilla paremmin saavutettavaksi. (Somnez, 2016)

Tämän hetken suurimmat mobiilikehitysalustat ovat Applen iOS- ja Googlen tukema Android-käyttöjärjestelmä. Androidia käyttäviä laitteita löytyy huomattavasti enemmän kuin laitteita, jotka käyttävät iOSia. Tämä selittyy sillä, että Androidin lähdekoodi on avoin, minkä johdosta kuka vain voi käyttää sitä. Android on suunniteltu toimimaan erilaisissa laitteissa, ja sitä käyttävät useat eri laitevalmistajat käyttöjärjestelmänä laitteissaan. iOS sen sijaan on suunniteltu toimimaan ainoastaan Applen valmistamissa laitteissa. (Somnez, 2016)

2.1 Mobiilisovellustyypit

Mobiilisovelluksia on kehitetty erilaisilla ohjelmistokehyksillä. Mobiilisovellusten eri tyyppejä ovat esimerkiksi natiivi-, HTML5- ja hybridimobiilisovellukset sekä progressiiviset verkkosovellukset. (R, 2021)

Natiivit mobiilisovellukset on kehitetty tietylle alustalle, joka on kirjoitettu yhdellä ohjelmointikielellä, ja tietylle käyttöjärjestelmälle. Tällaisia ovat esimerkiksi Android- tai iOS mobiilisovellukset. Koska natiivit mobiilisovellukset on tehty toimimaan tietyllä alustalla tai

käyttöjärjestelmällä, ne ovat nopeampia, varmatoimisempia ja pystyvät tarjoamaan ominaisuuksia silloinkin, kun laite ei ole yhteydessä verkkoon. (R, 2021)

HTML5-mobiilisovellukset ovat verkkosovelluksia, jotka on kehitetty älypuhelimille, tableteille ja muille kädessä pidettäville laitteille käyttäen HTML5 verkkosisällön standardia. HTML5-sovellukset on ohjelmoitu toimimaan useilla erilaisilla laitteilla verkon välityksellä, eli ne ovat myös yksi esimerkki monialustaisesta kehityksestä. Koska HTML5 on hyväksytty verkkosisällön standardi, sillä tehdyt mobiilisovellukset ovat yhteensopivia tavallisten tietokoneiden verkkoselainten kanssa. HTML5-sovelluksilla on myös varjopuolensa; koska ne ovat vahvasti sidottuna verkkoyhteyteen ja toimivat sen välityksellä, yhteyksien kohdatessa ongelmia sovellukset eivät välttämättä toimi ollenkaan. Lisäksi ne eivät ole yhtä nopeita tai tehokkaita kuin esimerkiksi natiivit mobiilisovellukset. (TechTarget, 2015)

Hybridimobiilisovellukset koostuvat taustakoodista ja natiiveista katseluohjelman eli käyttöliittymän ominaisuuksista. Hybridisovelluksia ovat jotkut maailman johtavista sovelluksista, kuten Gmail, Twitter ja Amazon. Kehittäjät käyttävät erilaisia laajennuksia, joiden avulla he pystyvät hyödyntämään eri alustojen natiiveja ominaisuuksia. Lisäksi hybridisovellukset ovat nopeahkoja kehittää, ja niiden ylläpitäminen on yksinkertaisempaa kuin monen muun sovellustyyppin. (R, 2021)

Progressiiviset verkkosovellukset toimivat verkkoselaimella, mutta käyttäytyvät kuin natiivit mobiilisovellukset. Tämä tarkoittaa, että ne ovat sekä nopeita että halvempia ja helpompia kehittää kuin natiivit mobiilisovellukset, eikä niitä tarvitse asentaa laitteelle. (R, 2021)

Progressiiviset verkkosovellukset ovat verkkosivuston tavoin toimivia kehittyviä sovelluksia. Kyseessä ei ole uusi teknologia tai standardi, vaan termi progressiivinen verkkosovellus kuvaa tiettyä tapaa suunnitella ja kehittää verkkosovelluksia. Verkkosovellus yksinkertaisesti tarkoittaa sovellusta, joka toimii kuten verkkosivusto. Progressiivisuus tarkoittaa, että tällaisten sovellusten käyttäjäkokemus paranee vähitellen verkkoselainten saatessa uusia ominaisuuksia ja päivityksiä. Progressiiviset verkkosovellukset toimivat vanhoillakin verkkoselaimilla, mutta kykenevät hyödyntämään uusia teknologioita, jos käyttäjän

verkkoselain tukee niitä. Progressiiviset verkkosovellukset toimivat jo lähes yhtä hyvin kuin natiivit sovellukset, mutta esteenä laajemmalle käytölle on tuen puute teknologioille, joita progressiiviset verkkosovellukset hyödyntävät. (Hakulinen, 2018)

2.2 Android ja iOS

Android on Googlen kehittämä mobiilikäyttöjärjestelmä pääasiassa kosketusnäytöllä varustetuille laitteille, älypuhelimille ja tableteille. Sen suunnittelu mahdollistaa mobiililaitteiden käyttämisen intuitiivisesti sormiliikkeillä, joita ihmiset yleisesti käyttävät, kuten pyyhkäisy, napautus tai puristusliike. Google käyttää yksilöllisesti suunniteltuja Androidin ohjelmistoja myös televisioissa, autoissa ja rannekelloissa. (Chen, 2021)

Android käyttöliittymän kehitti ensimmäisenä Android, Inc. -yhtiö, ennen kuin Google osti sen vuonna 2005. Pian hankkimisen jälkeen, vuonna 2007, Google julkisti yhtiön ensimmäisen kaupallisesti saatavilla olevan Android-laitteen, joka saapui markkinoille 2008. Siitä lähtien ohjelmisto- ja sovelluskehittäjät ovat pystyneet käyttämään Android-teknologiaa mobiilisovellusten kehittämiseen. Androidille suunnattuja mobiilisovelluksia voin hankkia sovelluskaupoista kuten Google Playsta. Koska Android on Googlen tuote, käyttäjät voivat lisäksi yhdistää mobiililaitteitaan Googlen tarjoamiin muihin tuotteisiin, kuten pilvipalveluihin, sähköposteihin ja videopalveluihin. (Chen, 2021)

Androidin lähdekoodi on julkaistu avoimena lähdekoodina, jotta se auttaisi edistämään standardien avaamista mobiililaitteissa. Vaikka Android on julkaistu avoimen lähdekoodin muodossa, sitä käyttävät laitteet sisältävät laitevalmistajien omia ohjelmistoja tullessaan markkinoille. (Chen, 2021)

Vaikka Android on käyttäjille hyvä vaihtoehto toisten mobiilikäyttöjärjestelmien sijaan, silläkin on edelleen rajoituksia. Kehittäjäpuolella monimutkaisten käyttäjäkokemuksien ja rajapintojen ohjelmointi on usein haastavaa, ja vaatii enemmän tukeutumista Javaan kuin Objective-C:hen. Käyttäjäpuolella erilaisilla sovelluksilla, joita on tarjolla Androidille, on tapana olla tehty matalampien standardien mukaan kuin toisten käyttöjärjestelmien

sovelluskaupoissa. Toisin sanoin, näillä sovelluksilla on heikompi turvallisuus ja käyttäjät voivat joutua todennäköisemmin tietomurron kohteeksi. (Chen, 2021)

Apple iOS on käyttöjärjestelmä iPhoneille, iPadeille ja muille Applen mobiililaitteille. Se perustuu Mac OS:iin, joka on käyttöjärjestelmänä Applen Mac-tietokoneissa. Apple iOS on suunniteltu helppoon ja saumattomaan verkostoitumiseen usean eri Applen tuotteen välillä. Apple iOS on maailman toiseksi suosituin mobiilikäyttöjärjestelmä. Kesäkuussa 2021 sillä oli 26,3 % osa puhelinmarkkinoista, ja Androidilla oli 73,3 % osa. (Kenton, 2021)

iOSin ensimmäinen versio julkaistiin kesäkuussa 2007, kun iPhone saapui ensimmäisen kerran markkinoille. iPhone Operating System, josta iOS on lyhennys, pohjautuu Unix-käyttöjärjestelmään ja se on käytössä jokaisessa Applen mobiililaitteessa. iOS-nimitys ei ollut virallisesti käytössä ennen vuotta 2008, jolloin Apple julkaisi iPhoneen ohjelmistonkehityspaketin, täten mahdollistaen sovellusten tekemisen alustalle kenen tahansa toimesta. (Kenton, 2021)

iPhoneen suosiota perustellaan usein sen käyttöjärjestelmän käyttäjäkeskeisellä suunnittelulla ja tehokkuudella. Lähes 218 miljoonaa iPhonea oli myyty vuoden 2018 loppuun mennessä, mikä teki laitteesta suosituimman julkaistun tuotteen koskaan. Joidenkin arvioiden mukaan iOS-tuotteet ovat tuottaneet Applelle yli triljoona dollaria tuloa. (Kenton, 2021)

3 Flutter

Luvussa kerrotaan mikä on Flutter ohjelmistokehityspaketti ja siihen liittyvää yleistä tietoa. Kerrotaan yleisellä tasolla Flutterin käyttämästä Dart-ohjelmointikielestä. Tarkastellaan tarkemmin mitä ovat Flutterin käyttämät widgetit ja niihin liittyvät statet. Kerrotaan lisäksi suosittuja sovelluksia, joita on tehty Flutterilla.

3.1 Yleistä Flutterista

Flutter on Googlen kehittämä avoimeen lähdekoodiin perustuva mobiili ohjelmistokehityspaketti. Sillä on mahdollista luoda sekä Android- että iOS-käyttöjärjestelmillä toimivia sovelluksia samanaikaisesti käyttäen vain yhtä koodipohjaa. Ohjelmointi Flutterilla tapahtuu Dart-ohjelmointikielellä. Flutter oli kehityksen beetavaiheessa jo vuonna 2015, mutta sen virallinen julkaisu tapahtui joulukuussa 2018. (Concise Software, 2019)

Flutter-ohjelmistokehitys julkistettiin ensimmäistä kertaa vuonna 2015 Googlen toimesta, ja se toimi Android-käyttöjärjestelmällä. Flutterin ensimmäinen vakaa version julkaistiin joulukuussa 2018. Vuoden 2020 toukokuussa Flutterin versio 1.17 julkaistiin, jossa oli mukana Metal API -liitäntä, jonka myötä käytettävyys parani iOS-alustoilla. Flutter on ohjelmoitu käyttäen C-, C++ - ja Dart -ohjelmointikieliä ja se käyttää Googlen Skia-grafiikkamoottoria käyttöliittymän renderöintiin. Skiaa on käytetty myös monissa muissa tunnetuissa tuotteissa, kuten Googlen Chrome-selaimessa, Chrome-käyttöjärjestelmässä, Mozilla Firefox-selaimessa ja Androidissa. (Velykyy, 2020)

3.2 Dart-ohjelmointikieli

Dart-ohjelmointikieli esiteltiin suurelle yleisölle monta vuotta ennen Flutteria. Googlen kehittämää ja vuonna 2013 julkaistua Dart 1.0 versiota suunniteltiin alunperin korvaamaan JavaScriptin käyttö selaimissa. Dart on tyypiltään olio-ohjelmointikieli ja syntaksiltaan

samankaltainen kuin C++ -, Java- ja JavaScript -ohjelmointikielet, joten teoriassa näitä kieliä käyttävien kehittäjien olisi vaivatonta siirtyä Dartiin. Googlen markkinoinnista huolimatta, Dartin suosio ohjelmointikielenä ei kuitenkaan aluksi kasvanut ja ohjelmistokehittäjät jatkoivat JavaScriptin käyttöä. Dartin suosio alkoi kuitenkin kasvamaan vuonna 2018, kun versio 2.0 julkaistiin alkuvuodesta ja erityisesti loppuvuodesta Flutterin julkaisun jälkeen. (Bolton, 2019)

Flutter toimii käyttämällä Dart-virtuaalikonetta Windows, Linux ja macOS -käyttöjärjestelmillä. Dart-virtuaalikone käyttää nopeatoimista koodikokoelmaa, jonka avulla se pystyy päivittämään kehittäjän tekemät muutokset vain kyseiseen tiedostoon sen sijaan, että koko ohjelmaa tarvitsisi jatkuvasti päivittää. Tämän ominaisuuden avulla kehittäjän on mahdollista nähdä tekemänsä muutokset välittömästi ilman uudelleenkäynnistystä, säästäen runsaasti aikaa. Dart-virtuaalikone käyttää lisäksi ahead-of-time (AOT) -kokoamista, jolla Dart-ohjelmointikieltä käännetään alustasta riippuen sopivammaksi kieleksi, mahdollistaen paremman suorituskyvyn mobiililaitteilla. (Velykyy, 2020)

3.3 Widgetit

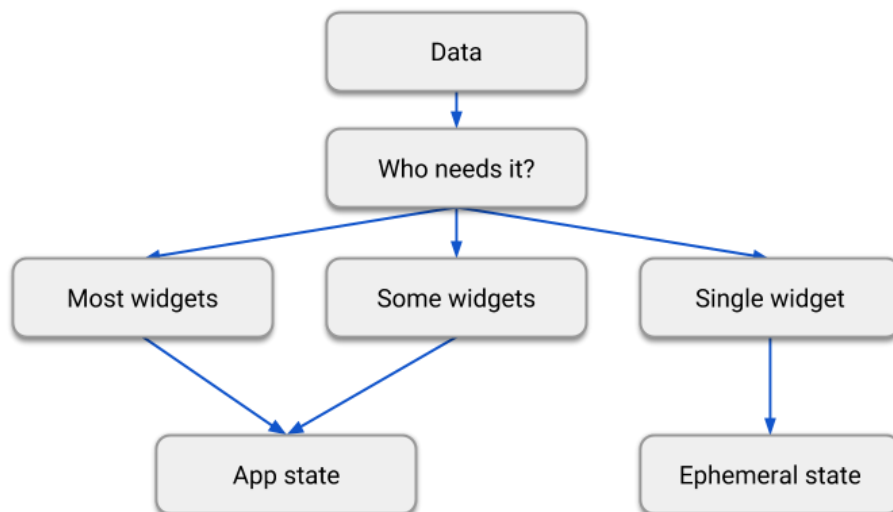
Flutterin pohjana oleva innovaatio on, että sovellusten koko käyttöliittymä olisi mahdollista luoda yhdistelemällä erilaisia widgettejä eli pienoishjelmia. Flutterissa tällaiset widgetit voivat olla mitä tahansa objekteja sovelluksissa, kuten painikkeita, taustakuvia tai erilaisia ei-toiminnallisia UI:n ulkoasuun liittyviä elementtejä. Sovellusten kehittäjien on mahdollista luoda Flutteria käyttäen mukautettuja widgettejä, joita voidaan yhdistää jo olemassa oleviin widgetteihin. (NIX United, 2020)

Flutterin tarjoamia widgettejä on kahta eri tyyppiä: stateless ja stateful. Widget, joka on tyypiltään stateless, ei muutu sovellusta käytettäessä. Luonnin jälkeen stateless widgetin sisältämä data ja asetukset pysyvät samoina koko sovelluksen elinkaaren ajan. Stateful widget sen sijaan voi muuttua sovelluksen käyttämisen yhteydessä. Tällaiset muutokset voivat liittyä esimerkiksi käyttäjän toimiin, jotka muuttavat widgetin sisältämää dataa tai

asetuksia. Stateful widgetissä tapahtuvat muutokset voivat myös olla periytyviä toiselta widgetiltä. (Choo, 2020)

Flutterilla tehdyssä sovelluksessa voi olla kahta erityyppistä statea. Yleistykseenä state voidaan määritellä kaikeksi muistiin tallennetuksi dataksi, kun sovellus on käynnissä. Statet jaotellaan ephemeral- eli lyhytaikaisiksi- tai paikallisiksi stateiksi ja app stateiksi. Ephemeral state on rajoitettu usein vain yhteen stateful widgettiin, ja tällainen on esimerkiksi navigaatiopalkin sisäinen state, jossa tapahtuu muutos siirryttäessä toiselle sivulle. App state, jota välillä kutsutaan myös jaetuksi stateksi, on monimutkaisempi kuin ephemeral state. Sen sisältämät datamuutokset voi jakaa moneen eri sovelluksen osaan ja asettaa pysymään muistissa käyttökertojen välillä. Erilaisia esimerkkejä app statesta ovat käyttäjän muuttamat asetukset, sisäänkirjautumistiedot tai ostoskärri verkkokauppa sovelluksessa. Kuva 1 kaaviossa näkyy yleistys siitä, kuinka ephemeral- ja app staten käyttö jakautuu sovelluksessa. (Flutter, n.d.-a)

Kuva 1 Kaavio statejen käytöstä datan mukaan. (Flutter Statet, n.d.-b)



3.4 Flutterilla tehtyjä sovelluksia

Monet yritykset ovat hyödyntäneet Flutterin ominaisuuksia tekemissään sovelluksissa. Flutter on suosittu vaihtoehto yritysten sovelluskehityksessä sen hyvän suorituskyvyn, kehittäjäystävällisyyden ja monella alustalla toimivan yhden koodipohjan takia. Monen pienemmän yrityksen lisäksi Flutteria ovat käyttäneet myös monet suuret ja tunnetut yritykset, kuten Google, SpaceX, BMW ja Toyota. (Khomutova, 2022)

Google, Flutterin kehittäjä, on yksi suurimmista yrityksistä, joka on sitä käyttänyt sovellusten kehitykseen. Google Ads ja Google Pay ovat käytettyjä esimerkkejä sovelluksista, jotka on tehty Flutterilla. Näiden lisäksi Googlen Stadia-pelialusta on myös tehty Flutterilla. SpaceX Go ja My BMW -mobiilisovellukset ovat esimerkkejä muista suurten yhtiöiden tekemistä Flutter-sovelluksista. Erikseen tehdyn sovelluksen sijaan, Toyota on hyödyntänyt Flutteria autojensa sisäisissä järjestelmissä. (Khomutova, 2022)

4 Kehittämistyön tarkoitus

Kehittämistyön tarkoituksena on tutustua Flutterin ja Dartin käyttämiseen mobiilisovelluksen kehittämisessä. Aikaisempiin opiskeluihin ei ole kuulunut kumpikaan. Tavoitteena on hankkia kokemusta näistä ohjelmista ja opetella samalla uuden ohjelmointikielen käyttöä.

4.1 Tausta

Opinnäytetyön tekijällä on aikaisempaa kokemusta ohjelmoinnista opiskelujen pohjalta. Mobiilisovelluksen kehittäminen on ollut osa aikaisempia opintoja React Nativea käyttäen. Ohjelmointikielistä aikaisempaa kokemusta on C#:sta, Javasta ja Javascriptistä. Työkaluista ja ohjelmista, joita kehitystyössä käytetään, tekijällä on aikaisempaa kokemusta Android Studion tarjoamista emulaattoreista ja VSCode:n käyttämisestä editorina. Flutterista ja Dart-ohjelmointikielestä ei opinnäytetyön tekijällä ole aikaisempaa kokemusta.

4.2 Tavoitteet

Tavoitteena on luoda yksinkertainen sovellus, johon on toteutettu yleisimpiä mobiilisovelluksissa käytettyjä komponentteja. Sovellusta voi käyttää pohjana, jota on mahdollista jatkokehittää monella tapaa. Tavoitteena on myös kokemuksen kartuttaminen yleisen mobiilikehittämisen suhteen sekä tutustua Flutterin ja Dart-ohjelmointikielen käyttöön.

5 Sovelluksen kehitys

Tässä osiossa käydään läpi yksinkertaisen Android-mobiilisovelluksen kehittäminen Flutteria ja Dartia käyttäen. Sovelluksen tavoitteena on sisältää muutama välilehti, sekä navigointi näiden välillä. Kehitys alkaa tarvittavien ohjelmien asennuksilla ja käyttöönottamisella, sekä lyhyen ohjauksen läpikäynnillä. Sovelluksen rakentaminen alkaa kahden sivun luonnilla, jonka jälkeen käydään läpi navigoinnin lisääminen näiden sivujen välille. Navigaation luomisen jälkeen toiselle luodulle sivulle lisätään myös välilehtinavigaatio. Välilehtiä tehdään kaksi, joista toiseen luodaan mahdollisuus tallentaa käyttäjän syöte tiedostoon. Toisella välilehdellä luetaan tiedostoon tallennettu sisältö.

5.1 Kehitysympäristön rakentaminen

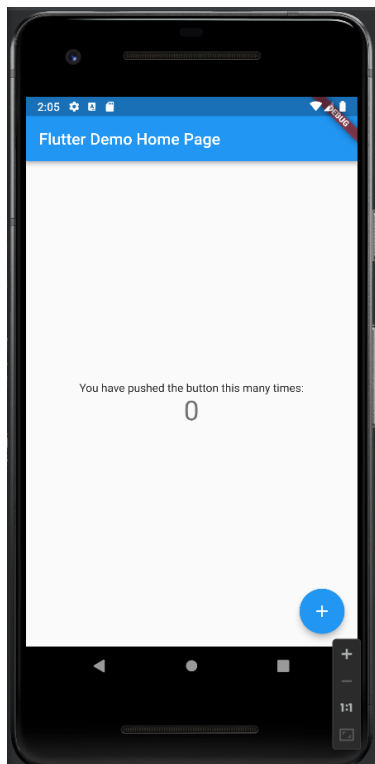
Kehitysprojehti alkaa tarvittavien ohjelmien lataamisella, asentamisella ja tutustumisella Flutteriin käytännössä. Ohjeistuksena kehitysympäristön rakentamiseen käytetään Flutterin dokumentaationsivuston englanninkielisiä ohjeita. Getting Started -osiosta löytyvien ohjeiden avulla kehitysympäristön rakennus onnistui hyvin. Ohjeet alkavat Flutter-ohjelmistonkehityspaketin lataamisella ja asentamisella. Lisäksi ohjeistuksessa käsiteltiin kehityksessä käytettävien VScode – ohjelmointieditorin ja Android Studion tarjoaman emulaattorin käyttöönottaminen askel kerrallaan. Edellä mainituista ohjelmista jälkimmäisen, Android Studion, täysi asennus vaaditaan. (Flutter, n.d.-c)

Kehitysympäristön rakentamisen jälkeen edellä mainitut sivut ohjaavat sivuille, jossa opastetaan tekemään yksinkertainen sovellus Flutterilla. Ohjauksen aikana kerrotaan tarkemmin erilaisista Flutterin toiminnoista ja ominaisuuksista. Ohjauksessa käydään läpi sovelluksenkehityksen edetessä perusasioita, joita Flutterilla voi tehdä, ja tutustutetaan ohjattava Dart-kielen syntaksiin. (Hracek, n.d.)

5.2 Kehitystyö

Sovelluksen kehitystyö alkaa uuden projektin luonnilla. Flutter luo automaattisesti Kuva 2 näkyvän toimintavalmiin demoprojektin, jossa on laskuri napin painamiskertoja varten. Koodipohjan seassa on runsaasti Flutterin perusominaisuuksia selittäviä kommentointeja, jotka poistetaan selkeyden vuoksi ennen kehityksen aloittamista. Demoprojektin voi suorittaa heti, ja tällä voi varmistua, että kehityksessä käytettävät ohjelmat toimivat oikein. Käynnistämällä demo-ohjelman heti, voidaan ottaa myös hyöty irti Flutterin ”Hot Reload” -toiminnosta, joka päivittää emulaattorin näkymän viiveettömästi ohjelmoijan tallentaessa koodiin tehdyt suoritettavat muutokset. ”Hot Reload” -toiminto ei käynnistä emulaattorin ohjelmaa uudestaan, joten jos demo-ohjelman toiminnallisuutta oli testattu napauttamalla painiketta, laskurin tallentama painallusten määrä ei nollaannu.

Kuva 2 Uuden Flutter-projektin valmiin demon näkymä emulaattorissa.



Kehityksen ensimmäisenä tavoitteena on luoda mahdollisuus navigoida sivujen välillä. Ennen navigoinnin rakentamista, aloitetaan siirtämällä aloitussivu ja sen sisältämä laskuri pois

valmiiksi luodusta MyHomePage-luokasta, ja tehdään se omaksi luokakseen. Luokalle annetaan nimeksi Sivu1 ja siitä tehdään stateful widget, jotta sivun elementtien, kuten laskurin lukeman, on mahdollista muuttua nappia painettaessa. Myös laskurin käyttämä funktio ja muuttuja siirretään tämän uuden luokan sisään, jotta toiminnallisuus säilyy. Siirron yhteydessä muutetaan näkyvät tekstielementit suomenkielisiksi ja yksinkertaistetaan ohjelmakoodia poistamalla title-muuttuja MyHomePage-luokasta, joka helpottaa luokkien kutsumista myöhemmin. MyApp-luokasta, joka toimii sovelluksen root-widgettinä, löytyy määritettynä home, jolla asetetaan, mikä on sovelluksen käyttämä oletusnäkyvä. Tämän voi vaihtaa kutsumaan Sivu1-luokkaa, jos haluaa varmistaa, että sen siirto omaksi luokakseen on onnistunut.

Seuraavaksi luodaan toisen sivun pohja omana luokkana ja nimetään se Sivu2:ksi. Tässä vaiheessa se ei vielä sisällä muuttuvaa sisältöä, joten siitä tehdään stateless widget. Kehityksen myöhemmässä vaiheessa on mahdollista muuttaa myös tämä sivu stateful widgetiksi, käyttämällä Flutterin Refactor-toimintoa. Toinen sivu ei vielä näy, ellei vaihda MyApp-luokasta oletusnäkyvää, mutta se on valmiina testaamista varten. Sivu2:n ohjelmakoodi on tässä vaiheessa vielä yksinkertainen ja sitä voi tarkastella Ohjelmakoodi 1. Scaffold toimii kehiksenä, jonka sisään muut visuaalisen käyttöliittymän osat määritellään. Sivun yläalaidassa on appBar-niminen elementti, jossa on tekstinä sivun otsikko. Sivu sisältää Center-widgetin avulla keskitetyn Row-widgetin, jonka sisällä on ”Tämä on toinen sivu” - lausahdus. Row-widgettiin määritetty ominaisuus mainAxisAlignment keskittää Row-widgetin sisältämän lausahduksen sivusuunnassa, jonka tuloksena teksti on täysin sivun keskellä.

Ohjelmakoodi 1 Luodun uuden sivun ohjelmakoodi.

```
class Sivu2 extends StatelessWidget {
  const Sivu2({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Toinen Sivu'),
      ),
      body: Center(
```

```

        child: Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: const <Widget>[
            Text('Tämä on toinen sivu'),
          ],
        ),
      ),
    );
  }
}

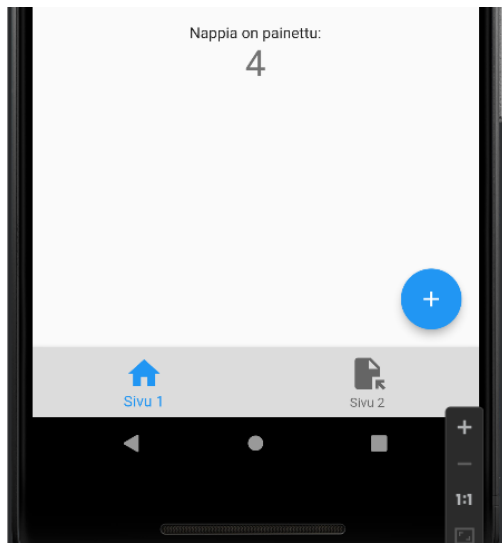
```

On tärkeää muistaa, että Flutteria käyttäessä koko sovelluksen käyttöliittymä koostuu widgeteistä. Sekä aloitussivu että toinen sivu ovat molemmat widgettejä, ja näiden sisällä olevat graafiset elementit ovat myös widgettejä. Aloitussivun tekeminen omaksi luokakseen ja toisen sivun pohjan luominen tässä vaiheessa tulevat helpottamaan navigoinnin tekoa ja testaamista, sekä selkeyttävät projektin rakennetta.

5.2.1 Navigaatio

Toteutettavaksi navigaatioksi valitaan ruudun alalaidassa oleva navigaatiopalkki. Palkista tulee löytyä kuvakkeet, joiden avulla siirrytään sivulta toiselle. Ruudun alalaidassa oleva navigaatiopalkki on valmis `bottomNavigationBar`-niminen widget Flutterissa, joten sen luominen käyttöliittymään on yksinkertaista. Toiminnallisuuden lisääminen navigaatiopalkkiin on haastavampaa, jotta sen avulla voidaan siirtyä sivulta toiselle. Navigaatiopalkki ja sen logiikka tehdään Flutterin valmiiksi luomaan `MyHomePage`-nimiseen ja `stateful`-tyyppiseen luokkaan, josta aiemmin siirrettiin aloitussivu laskureineen omaksi luokakseen.

Kuva 3 Emulaattorinäkömä ruudun alalaidassa sijaitsevasta navigaatiopalkista.



BottomNavigationBar-widgetin sisältämät objektit ovat painikkeita ja ne ovat listamuodossa. Jokainen listassa oleva objekti vaatii vähintään kuvakkeen määrittelyksi. Selkeyden vuoksi objekteille määritellään kuvakkeiden lisäksi myös nimike, joka vastaa aikaisemmin luotuja Sivu1- ja Sivu2-luokkia. Myös kuvakkeiden kokoa kasvatettiin ja navigaatiopalkille lisättiin taustaväri, jotta se erottuisi paremmin muusta näkymästä. Tätä voi huomioida Kuva 3 ja Ohjelmakoodi 2.

Ohjelmakoodi 2 Navigaatiopalkin ohjelmakoodi käyttöliittymässä.

```
bottomNavigationBar: BottomNavigationBar(
  backgroundColor: const Color.fromARGB(255, 220, 220, 220),
  iconSize: 35,
  items: const <BottomNavigationBarItem>[
    BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Sivu
1'),
    BottomNavigationBarItem(icon: Icon(Icons.file_open), label:
'Sivu 2'),
  ],
  currentIndex: selectedIndex,
  onTap: _onItemTapped,
)
```

Navigaation toiminnallisuuden saavuttamiseksi aloitetaan tekemällä pages-niminen lista-widget, joka sisältää objekteina Sivu1- ja Sivu2-luokkien kutsut. Luodaan selectedIndex-niminen integer-muuttuja. Tätä muuttujaa hyödynnetään pages-listan objektien index-

arvojen ja navigaatiopalkin sisältämien objektien, jotka ovat myös listamuodossa, index-arvojen yhdistämiseen. Tämä mahdollistaa määrittelemään, mikä sivu näkyy navigaatiopalkin kuvakkeita painettaessa. Navigaatiopalkin logiikkaa voi tarkastella Ohjelmakoodi 3. (Singh, 2020)

Ohjelmakoodi 3 MyHomePage-luokassa olevan navigaatiopalkin logiikka.

```
class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key});

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  List<Widget> pages = <Widget>[const Sivu1(), const Sivu2()];

  int selectedIndex = 0;

  void _onItemTapped(int index) {
    setState(() {
      selectedIndex = index;
    });
  }
}
```

Seuraava vaihe, joka tarvitaan navigaation toiminnallisuuden saavuttamiseen, on asettaa navigaatiopalkin currentIndex-ominaisuuden arvoksi aiemmin luotu selectedIndex-muuttuja, joka tällöin muuttaa arvoaan riippuen siitä kumpaa navigaation kuvaketta painetaan. Lisätään funktio, joka suoritetaan navigaatiopalkin kuvaketta painettaessa. Tämä funktio päivittää MyHomePage-luokan staten selectedIndex-muuttujan uudella arvolla, mahdollistaen navigaatiopalkin toimivuuden. MyApp-luokassa asetetaan oletusnäkymäksi MyHomePage-luokka, jolla saadaan navigaatiopalkki näkyviin emulaattorin ruudulle. Navigaatiopalkki on tässä vaiheessa täysin toiminnallinen, vaikka sivut eivät vielä näy. (Singh, 2020)

Navigaation toteutuksen viimeisenä vaiheena, jotta saadaan sivut näkyviin, lisätään MyHomePage-luokan Scaffold-kehiksen sisällöksi body-widget navigaatiopalkin lisäksi. Tällä widgetillä määritetään kehiksen pääsisältö. Sisällöksi määritetään aiemmin luotu pages-lista, josta näytetään selectedIndex-muuttujan arvoa vastaava sivu. Tällöin sovellus lataa sivun uudestaan navigaation kuvaketta painettaessa, minkä vuoksi esimerkiksi Sivu1-luokan laskuri

nollautuu aina sivunvaihdon yhteydessä. Parempana ratkaisuna body-widgetin sisällöksi asetetaan indexedStack-widget, jolle määritetään sisällöksi sama pages-lista selectedIndex-muuttujan mukaan. IndexedStack lataa molemmat sivut samanaikaisesti, mutta näyttää sen, johon selectedIndex-muuttujan arvo viittaa. Tämä mahdollistaa sivunäkymän vaihtamisen navigaatiosta ilman sivujen uudelleen lataamista, jolloin sivujen omat statet, kuten laskurin tallentamat painallukset, säilyvät. Molemmat toteutukset ovat näkyvissä Ohjelmakoodi 4. (Singh, 2020)

Ohjelmakoodi 4 Ruutunäkymän asettaminen ja selectedIndex-muuttujan yhdistäminen sivuihin. Kommentoituna ensimmäinen toteutustapa.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    // body: pages[selectedIndex],
    body: IndexedStack(
      index: selectedIndex,
      children: pages,
    )
  );
}
```

5.2.2 Tiedostoon tallennus

Toteutetaan sovellukseen mahdollisuus tallentaa tiedostoon käyttäjän syöttämää tekstiä, ja lukea tiedoston sisältö. Sovelluksen aikaisemmin tehtyyn toiseen sivuun lisätään toiminnallisuutena kaksi välilehteä. Välilehtivalikko sijaitsee ruudun ylä laidassa ja se sisältää kuvakkeet, joilla valitaan haluttu välilehtinäköymä. Ensimmäiselle välilehdelle sijoitetaan tekstikenttä, johon voi syöttää tekstiä. Syötetyn tekstin voi tallentaa tekstitiedostoon nappia painamalla. Toiselle välilehdelle asetetaan näkymään tiedoston sisältö.

Aloitetaan muuttamalla Sivu2-luokka stateful widgetiksi. Refactor toimintoa käyttäen Flutter muuntaa luokan stateful tyyppiseksi ja luo automaattisesti sille oman staten. Sivulle tehdään välilehtinavigointi, joka sijoittuu sivunäkymän ylälaidaan sivun otsikon alle. Luodaan DefaultTabController-widget ja sivun sisältö mukaan lukien scaffold-widget asetetaan sen sisälle. Length-arvolla määritetään tulevien välilehtien määrä. AppBar-widgettiin määritetään sivun otsikon alle TabBar-widget, joka luo käyttöliittymään sivusuuntaisen

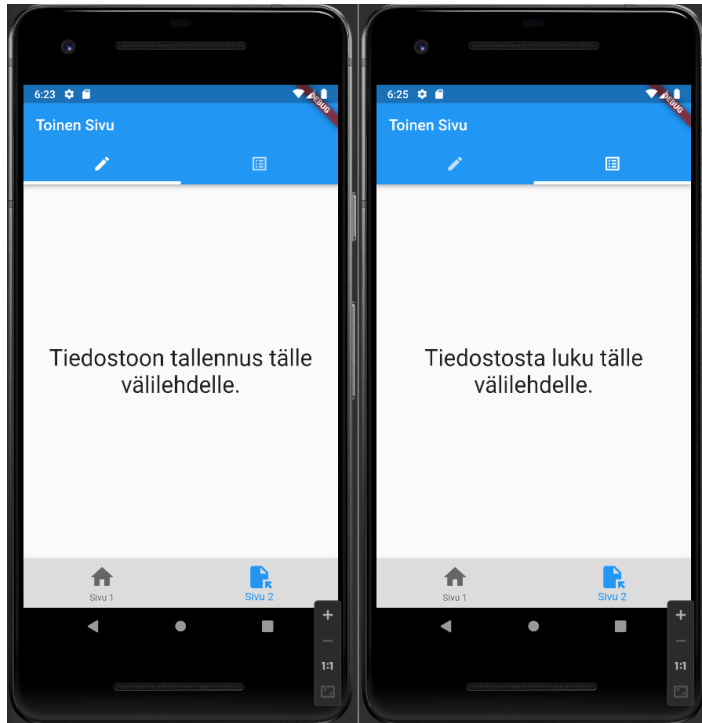
välilehtinavigaation. Tämä navigaatio sisältää kaksi välilehteä, joilla on oma kuvakkeensa. Välilehtivalinnan havainnointia helpottamaan `indicatorWeight`-ominaisuudelle asetetaan oletusarvoa suurempi arvo. (Flutter, n.d.-d)

Ohjelmakoodi 5 Välilehtivalikon ja sivun välilehtien sisällön ohjelmakoodi.

```
@override
Widget build(BuildContext context) {
  return DefaultTabController(
    length: 2,
    child: Scaffold(
      appBar: AppBar(
        title: const Text('Toinen Sivu'),
        bottom: const TabBar(
          indicatorWeight: 4,
          tabs: <Widget>[
            Tab(
              icon: Icon(Icons.edit),
            ),
            Tab(
              icon: Icon(Icons.list_alt),
            )
          ],
        ),
      ),
      body: const TabBarView(
        children: <Widget>[
          Center(
            child: Text(
              'Tiedostoon tallennus tälle välilehdelle.',
              style: TextStyle(fontSize: 30),
              textAlign: TextAlign.center,
            ),
          ),
          Center(
            child: Text(
              'Tiedostosta luku tälle välilehdelle.',
              style: TextStyle(fontSize: 30),
              textAlign: TextAlign.center,
            ),
          ),
        ],
      ),
    ),
  );
}
```

Välilehtivalikon luonnin jälkeen luodaan välilehdille pohja. Määritetään `body`-widget ja sen sisällöksi asetetaan `TabBarView`, joka sisältää listana välilehdet. Ohjelmakoodi 5 molemmat välilehdet sisältävät väliaikaisena sisältönä `Center`-widgetillä keskitettynä tekstiä, jonka fonttikokoa on suurennettu Kuva 4 havainnoinnin helpottamiseksi. (Flutter, n.d.-d)

Kuva 4 Emulaattorinäkymä toisen sivun molemmasta välilehdestä ja niiden välisestä navigaatiosta.



Välilehdelle, johon käyttäjän on mahdollista syöttää tietoa ja tallentaa se tiedostoon, luodaan TextField-widget ja painike. Kuva 4 näkyvä tallennusvälilehden väliaikainen sisältö poistetaan ja tuleva tekstikenttä sekä tallennukseen käytettävä painike tehdään Center-widgetin sijaan Column-widgetin sisään. Column-widgetin sisältö keskitetään ja TextField-widget luodaan Padding-widgetin sisään, jolla saadaan irrotettua tekstikentän reunat sekä muista elementeistä että ruutunäkymän ulkoreunoista. TextField-widgetille määritetään maxLines-ominaisuus, jonka arvoksi annetaan null, jolloin tekstikentän on mahdollista kasvaa suuremmaksi syötettävän tekstin mukaan. TextField-widgetille määritellään myös decoration-ominaisuus, jossa tarkennetaan tekstikentän reunat, nimike ja vihjeteksti. Tekstikentän luonnin jälkeen tehdään painike ElevatedButton-widgetillä, joka sijoittuu ruutunäkymässä tekstikentän alapuolelle. Tähän painikkeeseen liitetään tilapäisesti yksinkertainen print-funktio testaamaan sen toimintaa, joka myöhemmin korvataan tiedostoon tallentamisella. Tekstikentän ja välilehden toteutusta voi tarkastella Ohjelmakoodi 6. (Javatpoint, n.d.)

Ohjelmakoodi 6 Tekstikentän toteutus ruutunäkymään.

```
Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    const Padding(
      padding: EdgeInsets.all(15),
      child: TextField(
        controller: tekstiKontrolleri,
        maxLines: null,
        decoration: InputDecoration(
          border: OutlineInputBorder(),
          labelText: 'Kirjoita tähän',
          hintText: 'Tekstikenttä kasvaa tarvittaessa.',
          suffixIcon: IconButton(
            onPressed: tekstiKontrolleri.clear,
            icon: const Icon(Icons.clear),
          ),
        ),
      ),
    ElevatedButton(
      onPressed: tallennusNappi,
      child: const Text('Tallenna'),
    ),
  ],
),
```

Käyttöliittymän visuaalisten elementtien luomisen jälkeen niihin lisätään toiminnallisuutta. Tekstikenttään on mahdollista kirjoittaa tekstiä, mutta tämä teksti ei tallennu vielä mihinkään. Tekstikentän sisältämän arvon saamiseksi lisätään luokkaan `TextEditingController`, joka liitetään aiemmin luotuun `TextField`-widgettiin määrittelemällä sille `controller`-ominaisuus. Tämä kontrolleri päivittää arvoaan tekstikentän arvon mukaan ja sen avulla on mahdollista myös tyhjentää tekstikentän sisältö. `TextField`-widgettiin lisätään pieni kuvakepainike, jota painamalla tyhjennetään kontrollerin sisältö, joka tyhjentää samalla myös tekstikentän sisällön.

Ohjelmakoodi 7 Tallennuspainikkeen toteutus.

```
ElevatedButton(
  onPressed: () {
    if (tekstiKontrolleri.text != "") {
      tallennusNappi();
      showDialog(
        context: context,
        builder: (context) {
          return AlertDialog(
            content: Text(
              'Tallennettiin: ${tekstiKontrolleri.text}',
            ),
          );
        },
      );
    }
  },
)
```

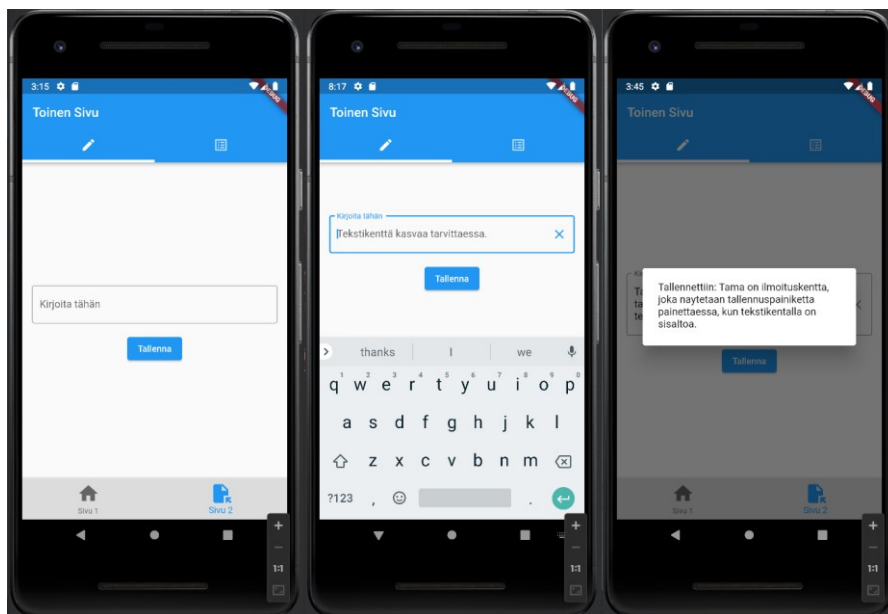
```

    );
  },
);
}
},
child: const Text('Tallenna'),
)

```

Kontrolleria hyödynnetään myös aiemmin tehdyssä tallennuspainikkeessa, johon lisätään ilmoituskenttä painamisen yhteyteen. Ohjelmakoodi 7 näkyy tallennusvälilehden sisältämä ElevatedButton-widget kokonaisuudessaan. Painettaessa tallennuspainiketta suoritetaan yksinkertainen if-lause, jolla tarkistetaan, että tekstikenttä ei ole tyhjä vertaamalla sitä kontrollerin sisältämän tekstin arvoon. Sovellus ilmoittaa AlertDialog-widgetillä, mitä tallennettiin käyttäen kontrollerin sisältämän tekstin arvoa. Kuva 5 voidaan havainnoida tallennusvälilehden eri näkymiä emulaattorissa sekä ilmoitusta, joka ilmaantuu painettaessa tallennusnappulaa. (Flutter, n.d.-e)

Kuva 5 Tallennusvälilehden perus-, kirjoitus- ja ilmoituskenttänäkymä emulaattorissa.



Tiedostoon tallentamisen seuraava vaihe on saada sovellus tallentamaan käyttäjän syöttämä teksti tekstitiedostoon. Tämä vaatii kolmen import-paketin lisäämistä sovellukseen; dart:async, dart:io ja path provider. Näistä kolmesta path provider -paketti vaatii myös

riippuvuuden lisäämisen sovelluksen pubspec.yaml-tiedostoon, jossa määritetään tarkemmin mitä path provider -paketin versiota käytetään. (Flutter, n.d.-f)

Tarvittavien pakettien lisäämisen jälkeen sovellukseen luodaan uusi luokka nimeltään TekstiStorage. Luokan sisään tehdään funktiot, joilla määritetään tiedoston polku, tehdään tiedostoon tarkempi viittaus ja kirjoitetaan dataa tiedostoon. Sivu2-luokkaan ja sen kutsuun navigoinnissa määritetään storage-parametri, jolle annetaan arvoksi luotu TekstiStorage-luokka. Aiemmin sovellukseen lisätty path provider -paketti mahdollistaa alustasta riippumattoman tavan päästä käyttämään yleisiä sijainteja laitteen tiedostojärjestelmässä. Sillä luodaan hakemistoon tiedostosijainti, jonne vain sovelluksen kautta pystyy luomaan tiedoston tai tekemään siihen muokkauksia. Ohjelmakoodi 8 näkyy TekstiStorage-luokan sisältämät funktiot. Ensimmäisessä funktiossa käytetään path provider -paketin automaattista tiedostosijainnin luontia hyväksi ja tallennetaan sijainnin polku. Toisessa funktiossa tarkennetaan mitä tiedostoa käytetään aiemmin määritellyssä polussa. Kolmas writeInput-funktio vastaanottaa tietoa ja kirjoittaa sen tiedostoon. (Flutter, n.d.-f)

Ohjelmakoodi 8 Funktioita, joita käytetään tiedostoon tallentamisessa.

```
class TekstiStorage {

  Future<String> get _localPath async {
    final directory = await getApplicationDocumentsDirectory();
    return directory.path;
  }

  Future<File> get _localFile async {
    final path = await _localPath;
    return File('$path/TallennettuFlutterTeksti.txt');
  }

  Future<File> writeInput(String kontrollerinTeksti) async {
    final file = await _localFile;
    return file.writeAsString(kontrollerinTeksti);
  }
}
```

Tallennusvälilehden painikkeen aikaisempi print-funktio muutetaan lähettämään kontrollerin sisältämä teksti writeInput-funktiolle, jolloin tiedostoon tallentaminen on mahdollista sovelluksessa. Koska kehitysprojektissa käytetään Android-emulaattoria, tallennuksia vastaanottava tekstitiedosto sijaitsee emulaattorin hakemistossa.

5.2.3 Tiedostosta luku

Sivu2-luokan kehitys jatkuu luomalla toiselle välilehtinäkyville mahdollisuus lukea tiedostoon tallennettu sisältö. Tätä varten TekstiStorage-luokkaan luodaan valmiiksi funktio, jolla haetaan aikaisemmin tehdyn tiedoston sisältö. Ohjelmakoodi 9 näkyvä readInput-funktio lukee tiedoston sisällön ja palauttaa sen contents-muuttujana. (Flutter, n.d.-f)

Ohjelmakoodi 9 TekstiStorage-luokkaan lisätty funktio, joka hakee tiedoston sisällön.

```
Future<String> readInput() async {
  final file = await _localFile;

  final contents = await file.readAsString();

  return (contents);
}
```

Luodaan toisen välilehden ulkoasu, jonne tulee näkyviin tiedoston sisältö. Välilehden aikaisempi Center-widget korvataan Column-widgetillä, jonka sisältö keskitetään. Column-widgetin sisällöksi asetetaan kaksi Text-widgettiä ja ElevatedButton-painike. Näiden välissä on kaksi SizedBox-widgettiä erottamassa elementtejä toisistaan, jotta ruutunäkymän tulkitseminen olisi helpompaa. Myös ensimmäinen Text-widget on apuna ruutunäkymän tulkitsemisessa, eikä sisällä muuttuvaa tekstisisältöä. Toinen Text-widget sisältää tiedostosta luettavan sisällön, ja sitä on korostettu asettamalla tekstin fonttikokoa suuremmaksi. Painikkeen, joka sivulle lisää, on tarkoitus mahdollistaa tiedostosta luetun sisällön päivitys, jos tiedostoon on tallennettu uutta dataa sovelluksen viimeisimmän käynnistytksen jälkeen. Ohjelmakoodi 10 voidaan havainnoida tarkemmin lukuvälilehden rakennetta.

Ohjelmakoodi 10 Välilehden toteutus, jossa sijaitsee tiedoston sisällön näyttäminen.

```
Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    const Text(
      'Tiedostosta haettu sisältö:',
      style: TextStyle(fontSize: 15),
    ),
    const SizedBox(
      height: 20,
    ),
    Text(
```

```

        tiedostonSisalto,
        style: const TextStyle(fontSize: 20),
      ),
      const SizedBox(
        height: 30,
      ),
      ElevatedButton(
        onPressed: paivitaNappi,
        child: const Text('Päivitä'),
      ),
    ],
  ),
),

```

Sivu2-luokkaan luodaan uusi string-tyyppinen tiedostonSisalto-muuttuja, jonka oletusarvo on tyhjä. Sivunäkymään lisätty painike asetetaan suorittamaan funktio, jolla päivitetään tiedostonSisalto-muuttujan arvo. Funktio suorittaa aikaisemmin luodun readInput-funktion, jonka palautuksena saatu arvo asetetaan tiedostonSisalto-muuttujan uudeksi arvoksi. Ohjelmakoodi 11 on lisätty Sivu2-luokkaan myös initState-funktio, joka suorittaa saman tiedostonSisalto-muuttujan arvon päivittävän funktion, kuin välilehden painike. initState-funktio suoritetaan heti, kun widget, jonka sisällä se on, aktivoituu sovelluksen käynnistämisen jälkeen. Tämän avulla saadaan näkyviin viimeisin tiedostoon tallennettu syöte heti sivunäkymän avautuessa, kun sovellus on käynnistetty uudelleen. (Flutter, n.d.-f)

Ohjelmakoodi 11 tiedostonSisalto-muuttujan arvon päivittävä funktio. Lisäksi initState-funktio, joka suorittaa päivitysfunktion sovelluksen käynnistykseen yhteydessä.

```

String tiedostonSisalto = "";

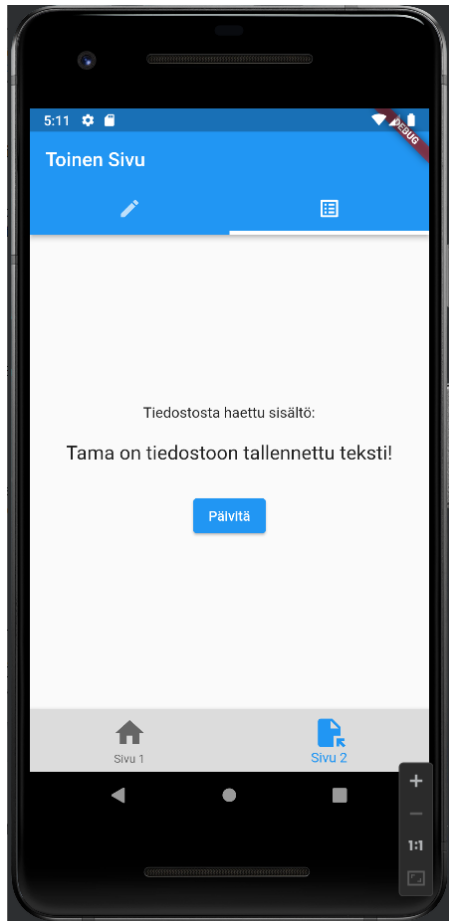
@override
void initState() {
  super.initState();
  paivitaNappi();
}

Future<void> paivitaNappi() async {
  widget.storage.readInput().then((value) {
    setState(() {
      tiedostonSisalto = value;
    });
  });
}

```

Sovellus ei tallenna käyttäjän syötteitä yhdeksi listaksi, vaan korvaa aikaisemman syötteen uudella. Kuva 6 voidaan tarkastella tiedoston sisältöä näyttävää välilehtinäköymää emulaattorilla.

Kuva 6 Emulaattorinäkömä tiedoston sisällön lukemisesta.



6 Johtopäätökset ja pohdinta

Tässä luvussa tarkastellaan tehdyn sovelluksen kehitystyötä ja lopputulosta. Arvioidaan Flutterin käyttämistä ja sillä sovelluksen kehittämistä. Lopuksi pohditaan potentiaalisia mahdollisuuksia jatkokehittää tehtyä sovellusta.

6.1 Lopputulos

Kehitystyön lopputuloksena on yksinkertainen mobiilisovellus. Sovelluksessa on kaksi sivua, joiden välillä on navigaatio ruutunäkymän alalaidassa. Ensimmäinen sivuista sisältää Flutterin demo-ohjelman napinpainamislaskurin ja sitä käytetään sovelluksen kotisivuna. Toiselle sivulle on toteutettu välilehtinavigointi ruutunäkymän ylälaitaan, tiedostoon tekstin tallentaminen ja tiedostosta tallennetun tekstin lukeminen. Ensimmäisellä välilehdellä on tekstikenttä, johon käyttäjä voi kirjoittaa tekstiä ja tallentaa sen tiedostoon. Toisella välilehdellä näytetään tiedostoon tallennettu sisältö ja se sisältää myös mahdollisuuden päivittää välilehden näkymä.

Tutkimuskysymyksiin vastaaminen onnistui hyvin opinnäytetyöprosessin aikana. Aloittamalla opinnäytetyön tekemisen teoriaosuudella ja selvittämällä mikä Flutter on, sai tietopohjaa käytännönsä sovelluksen tekemiseen. Tietopohjan kirjoituksen aikana vaikeuksia aiheutti yleisesti aiheen haastavuus ja opinnäytetyöntekijän rajattu tietopohja aiheesta. Sovelluksen kehityksen aikana suurin osa ongelmista liittyi vähäiseen aikaisemmin hankittuun ohjelmointikokemukseen ja Dart-ohjelmointikieleen. Suurimpia haasteita sovelluksen kehityksessä olivat loogiset ongelmat ja Dartin syntaksin ymmärtäminen. Esimerkiksi navigaation toteutuksen yhteydessä joutui etsimään syntaksi- ja logiikkaongelmiin ratkaisua monelta eri verkkosivulta. Sovelluksen sisältämät statet vaativat myös lisäselvitystä prosessin aikana. Ongelmanratkaisu verkon avulla oli haasteellisempaa verrattuna esimerkiksi React Nativeen tai Javascriptiin, sillä Flutter on teknologiana uudempi.

Sovelluksen kehittäminen Flutterilla onnistui kuitenkin hyvin, jopa vähemmän kokeneelta ohjelmoijalta. Tarvittavien ohjelmien käyttöönottoaminen ja kehitysympäristön rakennus

onnistuivat ongelmitta seuraamalla Flutterin verkkosivuilta löytyviä ohjeita. Flutter vaikuttaa helposti lähestyttävältä ja nopealta teknologialta tehdä usealla alustalla toimiva sovellus. Valmiisiin widgetteihin perustuva kehitys oli yllättävän sujuvaa, jopa ilman aikaisempaa kokemusta Flutterin käyttämisestä.

6.2 Tehdyn sovelluksen jatkokehitysmahdollisuuksia

Opinnäytetyönä tehdyssä sovelluksessa on toiminnallisuutta ja peruselementtejä, joita usein käytetään mobiilisovelluksissa. Sovelluksessa on kuitenkin monia erilaisia mahdollisuuksia jatkokehitykselle. Jatkokehitystä varten ensimmäisenä toimenä ohjelmointikoodi olisi hyvä jakaa useampaan eri tiedostoon ja liittää ne toisiinsa import-komennoilla. Tällä muutoksella projektin hallinta ja sen rakenteen havainnointi olisi huomattavasti helpompaa, kuin koko ohjelmointikoodin kirjoittaminen yhteen tiedostoon. Widgeteille, joita projektissa käytetään useasti, olisi myös mahdollista tehdä oma tiedosto, johon määritellään niille valmiit mukautetut asetukset. Tämä helpottaisi ja nopeuttaisi ohjelman käyttöliittymän tekemistä, kun jokaista samanlaista widgettiä ei tarvitsisi erikseen ohjelmoida tai mukauttaa yhteneväksi muun sovelluksen kanssa.

Jatkokehityksessä olisi hyvä testata sovellusta useammalla eri alustalla, kuten selaimella ja iOS-käyttöjärjestelmällä varustetulla emulaattorilla. Sovellukseen voisi lisätä eri alustoja varten mahdollisuuden mukautua näytön koon mukaan. Hyödyntämällä ja lisäämällä app ataten olisi mahdollista lisätä useita erilaisia käyttäjän toimiin pohjautuvia muutoksia, kuten vaihdon tumman ja vaalean teeman välillä tai sisäänkirjautumisen.

Sovelluksen ulkoasua jatkokehityksessä voisi parantaa animaatioilla, joiden lisäämiseen löytyy valmius joistakin widgeteistä, kuten navigaatiopalkista. Aloitussivun demolaskurin olisi mahdollista korvata jollakin toisella toiminnallisuudella. Toteutettua tiedostoon tallentamista olisi myös mahdollista parantaa vaihtamalla tallennusmuoto listaksi, jolloin olisi mahdollista tallentaa useampia eri syötteitä. Tätä ominaisuutta voisi jatkaa lisäämällä mahdollisuuden poistaa tallennettuja syötteitä tiedostosta. Paikallisen tiedoston sijaan olisi myös mahdollista hyödyntää pilveen tallentamista.

7 Yhteenveto

Opinnäytetyön kirjoitusprosessi venyi suunniteltua pitemmäksi, aiheuttaen hieman ongelmia yhteneväsyyden ja kirjoitetun tekstin laadun kanssa. Tutkimuskysymyksiin on pyritty vastaamaan opinnäytetyössä, vaikka niiden muotoilu on muuttunut prosessin aikana. Kokonaisuutena opinnäytetyö onnistui kuitenkin hyvin ja asetetut tavoitteet saavutettiin.

Käytännön osana tehdyn sovelluksen kehittäminen oli alussa haastavaa uuden ohjelmointikielen vuoksi. Kehitys kuitenkin helpottui samassa suhteessa kuin ohjelmointikielen osaaminen kasvoi, ja Flutterin sivuilla tarjottu ohjaus ensimmäisen sovelluksen kehitykseen oli erittäin hyödyllinen alkuun pääsemisessä. Flutteriin tutustuminen oli mielenkiintoista ja tehdyn kehitystyön perusteella se on toimiva työkalupaketti mobiilisovellusten kehittämiseen.

Opinnäytetyön yhteydessä ohjelmointikokemus kasvoi runsaasti. Yleinen tietämys mobiilikehityksestä kasvoi ja opin Flutterin käytön alkeet. Lisäksi opin alkeet Dart-ohjelmointikielestä, joka oli myös täysin uusi kieli itselleni. Ohjelmointiin liittyvästä ongelmanratkaisusta tuli myös lisää kokemusta, samoin kuin verkosta löytyvän dokumentaation hakemisesta.

Tehtyä sovellusta voi tulevaisuudessa käyttää pohjana muihin Flutter sovelluksiin tai jatkokehittää monella tapaa paremmaksi, jos haluan jatkaa kehitystä Flutterin parissa tai parantaa Dart-osaamistani. Kehitystyöstä ja erityisesti sen mukanaan tuomasta ohjelmointikokemuksesta on toivottavasti hyötyä myös opiskelujen jälkeen työelämässä.

Lähteet

Bolton, D. (27.3.2019). *The Fall and Rise of Dart, Google's 'JavaScript Killer'*. Haettu 25.1.2023 osoitteesta <https://www.dice.com/career-advice/fall-rise-dart-google-javascript-killer>

Chen, J. (3.2.2021). *Android Operating System*. Haettu 30.9.2021 osoitteesta <https://www.investopedia.com/terms/a/android-operating-system.asp>

Choo, J. (25.5.2020). *The Art of Flutter: Widget Explained*. Haettu 23.1.2023 osoitteesta <https://medium.com/flutter/g/the-art-of-flutter-widget-explained-6f176eb3daa9>

Concise Software. (26.8.2019). *What is Flutter? Here is everything you should know*. Haettu 27.1.2021 osoitteesta <https://medium.com/@concisesoftware/what-is-flutter-here-is-everything-you-should-know-faed3836253f>

Darji, P. (26.5.2021). *How to build a bottom navigation bar in Flutter*. Haettu 17.1.2023 osoitteesta <https://blog.logrocket.com/how-to-build-a-bottom-navigation-bar-in-flutter/>

Flutter. (n.d.-a). *Differentiate between ephemeral state and app state*. Haettu 2.2.2023 osoitteesta <https://docs.flutter.dev/development/data-and-backend/state-mgmt/ephemeral-vs-app>

Flutter. (n.d.-b). *Differentiate between ephemeral state and app state* [kuva]. Haettu 2.2.2023 osoitteesta <https://docs.flutter.dev/development/data-and-backend/state-mgmt/ephemeral-vs-app>

Flutter. (n.d.-c). *Install*. Haettu 5.1.2023 osoitteesta <https://docs.flutter.dev/get-started/install>

Flutter. (n.d.-d). *TabBar class*. Haettu 6.2.2023 osoitteesta <https://api.flutter.dev/flutter/material/TabBar-class.html>

Flutter. (n.d.-e). *Retrieve the value of a text field*. Haettu 7.2.2023 osoitteesta

<https://docs.flutter.dev/cookbook/forms/retrieve-input>

Flutter. (n.d.-f). *Read and write files*. Haettu 13.2.2023 osoitteesta

<https://docs.flutter.dev/cookbook/persistence/reading-writing-files>

Hakulinen, R. (16.2.2018). *What is a Progressive Web App (PWA) and why should I care?*

Haettu 29.9.2021 osoitteesta <https://gofore.com/en/progressive-web-app-pwa-i-care/>

Hracek, F. (n.d., viimeisin muokkaus 14.1.2023). *Your first Flutter app*. Haettu 6.1.2023

osoitteesta <https://codelabs.developers.google.com/codelabs/flutter-codelab-first#0>

Javatpoint. (n.d.). *Flutter TextField*. Haettu 7.2.2023 osoitteesta

<https://www.javatpoint.com/flutter-textfield>

Kenton, W. (10.8.2021). *Apple iOS*. Haettu 5.10.2021 osoitteesta

<https://www.investopedia.com/terms/a/apple-ios.asp>

Khomutova, S. (21.10.2022). *Top 24 famous apps built with Flutter Framework*. Haettu

23.1.2023 osoitteesta <https://apexive.com/post/top-apps-built-with-flutter-framework>

NIX United. (3.8.2020). *What is Flutter and why use Flutter for app development*. Haettu

29.1.2021 osoitteesta <https://nix-united.com/blog/the-pros-and-cons-of-flutter-in-mobile-application-development/>

R, G. (28.7.2021). *Mobile Application Development: Your Ultimate Guide for 2021*. Fingent.

Haettu 22.9.2021 osoitteesta <https://www.fingent.com/blog/mobile-application-development-your-ultimate-guide-for-2021/>

Singh, N. (7.12.2020). *FLUTTER : SAVE PAGE STATE WHEN USING BOTTOM NAV*. Haettu

1.2.2023 osoitteesta <https://www.nstack.in/blog/flutter-save-page-state-when-using-bottom-nav/>

Somnez, J. (5.12.2016). *What is Mobile Development?* Haettu 15.9.2021 osoitteesta <https://simpleprogrammer.com/what-is-mobile-development/>

TechTarget Contributor. (8.2015). *HTML5 mobile app*. Haettu 23.9.2021 osoitteesta <https://searchmobilecomputing.techtarget.com/definition/HTML5-mobile-app>

Velykyy, A. (22.6.2020). *Flutter: a full introduction to the framework*. Haettu 27.10.2021 osoitteesta <https://www.axon.dev/blog/flutter-a-full-introduction-to-the-framework>

Liite 1: Aineistonhallintasuunnitelma

Opinnäytetyötä tallennetaan Microsoft Word -asiakirjaan ja siihen liittyvää kehitysprojektin ohjelmointikoodia tallennetaan DART-tiedostoon. Kehitysprojektin aikana tallennetaan projektin ohjelmointikoodia myös kahteen erilliseen tekstitiedostoon. Tekstitiedostot sisältävät ohjelmointikoodia projektin eri vaiheista, joita käytetään sekä varmuuskopioina että alustana, jonka kautta siirretään ohjelmointikoodiesimerkkejä opinnäytetyöhön. Kehitysprojektia, opinnäytetyötä ja tekstitiedostoja säilytetään tekijän tietokoneen C-aseamalla yhdessä kansiossa, ja siitä tehdään säännöllisesti varmuuskopio erilliselle muistitikulle. Samasta kansioista löytyvät myös opinnäytetyössä käytetyt kuvat, ja muu opinnäytetyöhön tai kehitysprojektiin liittyvä materiaali. Kehitysprojektia, opinnäytetyötä ja muuta niihin liittyvää materiaalia säilytetään tekijän C-aseamalla ja muistitikulla ainakin vuoden verran opinnäytetyön valmistumisesta.

Opinnäytetyön aineiston ja tulokset omistaa opinnäytetyön tekijä.