Tampereen ammattikorkeakoulu

Mobiilisovelluksen kehittäminen osana tuotekehitystä

Jenni Ylisirniö

OPINNÄYTETYÖ
Joulukuu 2024

Tietotekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

YLISIRNIÖ, JENNI
Mobiilisovelluksen kehittäminen osana tuotekehitystä

Opinnäytetyö 42 sivua, joista liitteitä 0 sivua
Joulukuu 2024

Opinnäytetyön tavoitteena oli tarkastella mobiilisovelluskehitystä osana tuotekehitysprosessia. Työssä käsiteltiin mobiilisovelluskehityksen perusteita, eri teknologioita ja suunnitteluperiaatteita, jotka ovat olennaisessa roolissa onnistuneen mobiilisovelluksen kehittämisessä.

Työssä tarkasteltiin, miten olemassa oleva web-sovellus voidaan muuttaa mobiilisovellukseksi, millaisia suunnitteluvaiheita ja teknisiä ratkaisuja prosessi vaatii sekä millaisia erityispiirteitä mobiiliympäristö asettaa sovelluskehitykselle.

Opinnäytetyön tuloksena syntyi selkeä suunnitelma ja dokumentaatio siitä, miten web-sovellus voidaan muuttaa mobiilisovellukseksi ottaen huomioon tuotekehityksen eri vaiheet, tekniset vaatimukset ja käyttäjäkokemuksen merkitys. Opinnäytetyö antaa kattavan käsityksen mobiilisovelluskehityksestä osana tuotekehitystä, ja se tuo esiin käytännön näkökulmia sekä teoreettista ymmärrystä aiheesta.

Asiasanat: mobiilisovellus, tuotekehitys

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

YLISIRNIÖ, JENNI:
Mobile Application Development as Part of Product Development

Bachelor's thesis 42 pages, appendices 0 pages
December 2024

The objective of this thesis was to examine mobile application development as part of the product development process. The work covered the fundamentals of mobile application development, various technologies, and design principles that play a crucial role in developing a successful mobile application.

The thesis explored how an existing web application can be transformed into a mobile application, the design stages and the technical solutions required by this process, and the specific challenges that the mobile environment imposes on mobile application development.

The outcome of the thesis was a clear plan and documentation on how a web application can be converted into a mobile application, taking into account the different phases of product development, technical requirements, and the importance of user experience. The thesis provides a comprehensive understanding of mobile application development as part of product development and highlights the practical perspectives as well as the theoretical insights on the subject.

Key words: mobile app, product development

SISÄLLYS

LYHENTEET JA TERMIT

SaaS	Software as a Service on ohjelmistojen jakelumalli
GPS	Global Positioning System eli maailmanlaajuinen paikallistamisjärjestelmä on satelliittipaikannusjärjestelmä
haptiikka	tuntoaistiin liittyvät ominaisuudet mobiilisovelluksessa, yleensä erilaisia värinöitä
push-ilmoitus	ilmoitus, jonka mobiilisovellus lähettää päätelaitteen näytölle
PWA	progressive web app eli progressiivinen verkkosovellus
UI	user interface eli käyttöliittymä
UX	user experience eli käyttökokemus tai käyttäjäkokemus
API	application programming interface eli ohjelmointirajapinta
Gt	gigatavu on 1024 megatavua. Tavu eli bitti on tietotekniikassa käytettävä mittayksikkö tallennuskapasiteetille
Mt	megatavu on 1024 kilotavua. Tavu eli bitti on tietotekniikassa käytettävä mittayksikkö tallennuskapasiteetille.
GDPR	General Data Protection Regulation eli Yleinen tietosuoja-asetus on Euroopan Unionin asetus, joka säätelee tietosuojaa koskevaa lainsäädäntöä
OTA	Over-the-Air -päivitys tarkoittaa päivitystä jota ei ladata sovelluskaupasta
bugi	ohjelmointivirhe, joka aiheuttaa virhetilanteen käyttäjälle
debuggaus	virheenjäljitys, ohjelmointivirheen etsintää
MFA	Multi-Factor Authentication eli monivaiheinen tunnistautuminen
QA	Quality Assurance eli laadunvarmistus

1 JOHDANTO

Mobiililaitteiden käyttö on yleistynyt viime vuosina voimakkaasti. Suomessa 92% väestöstä vuonna 2022 käytti älypuhelinta. Älypuhelimia käytetään laajasti eri tarkoituksiin kuten puheluiden soittamiseen, viestittelyyn, sosiaalisen median selailuun ja päivittelyyn, televisio-ohjelmien ja elokuvien katseluun, musiikin kuunteluun, ostosten tekemiseen, sekä mobiilipelaamiseen. (FiCom, 2024)

Tässä opinnäytetyössä tutustutaan tuotekehityksen eri vaiheisiin mobiilisovelluskehityksen näkökulmasta. Opinnäytetyö on tehty toimeksiantona Loikka Design Oy:lle, joka on suomalainen kotihoidon ohjelmistoyritys. Loikka Design Oy kehittää Nursebuddy -ohjelmistoa SaaS-jakelumallillla kotihoitoa tarjoavien yritysten tarpeisiin. Nursebuddy -tuoteperheeseen kuuluu tällä hetkellä web-käyttöliittymä hoitoyritysten esihenkilöille, hoitajille ja omaisille. Hoitajien käyttöön on kehitetty erillinen mobiilisovellus, jolla on helppo seurata ja raportoida hoitotyöhön liittyviä asioita. (Nursebuddy, 2024)

Tämän opinnäytetyön tavoitteena on suunnitella Omaisnäkymästä uusi mobiilisovellus Nursebuddy -tuoteperheeseen. Omaisnäkymästä kotihoitoa saavan omaiset sekä muut yhteistyötahot voivat nähdä rajattuja tietoja hoitoa saavasta henkilöstä, ja seurata tiettyjä hoidon elementtejä. Omaisnäkymä on tällä hetkellä toteutettu verkkosovelluksena, mutta Omaisnäkymän mobiilisovellukselle olisi kerätyn käyttäjäpalautteen mukaan tarvetta.

2 MOBIILISOVELLUSKEHITYKSEN PERUSTEET

2.1 Mobiilisovellusten ja web-sovellusten erot

Mobiilisovellus on ohjelmisto, joka on luotu käytettäväksi pienissä langattomissa laskentalaitteissa. Tällaisia laitteita ovat nykyisin esimerkiksi älypuhelimet, tabletit ja älykellot. Mobiilisovellus ladataan laitteeseen laitteen omasta sovelluskaupasta.

Verkkosivu (engl. *website*) on kokoelma staattista sisältöä, kuten tekstiä, kuvia ja videoita. Verkkosivun sisältö esitetään verkkoselaimessa. Verkkosivut voivat sisältää kevyitä interaktiivisia elementtejä, mutta niiden pääpaino on sisällön esittämisessä. Verkkosivujen tarkoituksena on yleensä tarjota tietoa laajalle käyttäjäkunnalle ilman merkittävää vuorovaikutusta. Verkkosivuja ovat esimerkiksi yrityksen kotisivut, uutissivustot ja blogit.

Web-sovellus (engl. *web application*) eli verkkosovellus on suunniteltu käyttäjän ja sovelluksen väliseen vuorovaikutukseen. Verkkosovellus tarjoaa yleensä monipuolisempia toimintoja, kuten lomakkeiden täyttämistä, tiedon käsittelyä ja reaaliaikaista päivitystä. Verkkosovellukset voivat sisältää monimutkaisia logiikkarakenteita ja prosessointia, jotka mahdollistavat monipuoliset ja dynaamiset palvelut. Verkkosovelluksia ovat esimerkiksi verkkopankit, verkkokaupat, Facebook ja Gmail.

Verkkosivut ja verkkosovellukset on luotu käytettäväksi selainympäristössä millä tahansa laitteella, jossa on internetselain ja jolla on toimiva yhteys internettiin. Internetselaimia ovat esimerkiksi Google Chrome, Mozilla Firefox, Safari ja Microsoft Edge (Kansallinen Senioriliitto ry).

Web-sovelluksen ja mobiilisovelluksen keskeinen ero on käyttökokemuksessa. Mobiilisovelluksissa voidaan hyödyntää mobiililaitteen ominaisuuksia, kuten kameraa, GPS:ää ja haptiikkaa. Mobiilisovelluksessa ilmoitusten lähettäminen käyttäjälle on huomattavasti helpompaa push-ilmoitusten ja in-app-ilmoitusten muodossa. (Liini Agency, 2023)

Mobiilisovelluskehityksen historia on ollut lyhyt mutta sitäkin nopeampi. Ensimmäinen älypuhelimeksi luokiteltava laite, IBM Simon, julkaistiin vuonna 1994. Simon -mobiililaitteelle kehitetty maailman ensimmäinen mobiilisovellus nimeltä Simon toimi ainoastaan Simon -laitteella. Mobiililaitteiden kehityksen käännekohta oli, kun Apple Inc julkaisi ensimmäisen iPhone:n vuonna 2007. Sen jälkeen markkinoita ovat valloittaneet muut mobiililaitteita valmistavat brändit kuten Samsung, Huawei ja OnePlus. Mobiililaitteiden kehityksen myötä myös mobiilisovelluskehitys on kehittynyt valtavasti lyhyen ajan sisällä. (Sunila Goray, 2023)

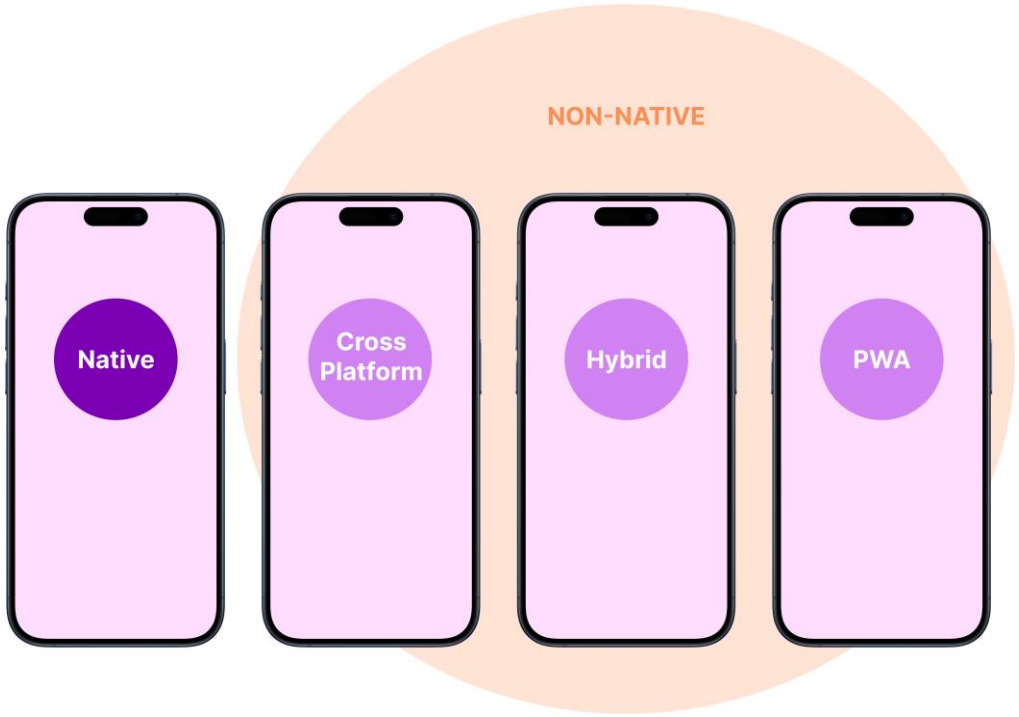
2.2 Mobiilisovelluskehityksen teknologiat

Mobiililaitteen käyttöjärjestelmällä tarkoitetaan ohjelmistoa, joka hallitsee mobiililaitteen muistia ja verkkoyhteyksiä, ja jolla ajetaan mobiililaitteelle asennettavia sovelluksia (Geeks for Geeks, 2024). Mobiililaitteille kehitettyjä käyttöjärjestelmiä on olemassa paljon, mutta kaikista yleisimpiä käyttöjärjestelmiä ovat Googlen kehittämä Android sekä Apple Inc:in kehittämä iOS. Noin 70% väestöstä Suomessa käyttää Andoird-käyttöjärjestelmän mobiilipuhelimia, kun taas noin 30% käyttää iOS-käyttöjärjestelmän laitteita (Statista, 2024). Samanlainen trendi on nähtävissä muissakin länsimaissa.

Kolmanneksi käytetyin mobiilikäyttöjärjestelmä nykyisin on Huawein kehittämä HarmonyOS. Suurin osa HarmonyOS-käyttöjärjestelmän mobiililaitteiden käyttäjistä on kiinalaisia. Globaaleilla markkinoilla on tällä hetkellä yli 900 miljoonaa laitetta, jotka käyttävät HarmonyOS-käyttöjärjestelmää. (Global Times, 2024)

Mobiilisovelluskehityksessä natiiviksi mobiilisovellukseksi kutsutaan sellaista mobiilisovellusta, joka on kehitetty suoraan tietylle mobiilikäyttöjärjestelmälle. Natiivi Android-sovellus on kirjoitettu Java- tai Kotlin-ohjelmointikielellä. Natiivi iOS-sovellus puolestaan on kirjoitettu Swift-ohjelmointikielellä. Natiivi mobiilisovellus kääntää natiivin lähdekoodin suoraan binäärikoodiksi, ja suorittaa binäärikoodin.

Ei-natiivi (engl. *non-native*) mobiilisovellus tarkoittaa yleisesti sellaista mobiilisovellusta, jonka lähdekoodia mobiilisovelluksen käyttöjärjestelmä ei voi suoraan kääntää binäärikoodiksi. Galatioton (Galatioto Chris, 2023) mukaan ei-natiiveja mobiilisovelluksia oli vuonna 2023 olemassa kolmen eri tyyppisiä (kuva1). Usein nämä termit sekoitetaan keskenään etenkin internetin keskustelupalstoilla.



KUVA 1. Mobiilisovelluksien tyypit luokiteltuna natiiviin ja ei-natiiveihin sovelluksiin.

Alustariippumaton mobiilisovellus (engl. *cross-platform app*) on ei-natiivi mobiilisovellus, joka on ohjelmoitu alustariippumattomien mobiilisovellusten rakentamiseen tarkoitetulla kehityskehyksellä. Tunnetuimpia kehityskehyksiä ovat React Native, Flutter ja Xamarin. Alustariippumattomat sovellukset käyvät läpi monimutkaisemman kääntämisprosessin, sillä niiden tarkoituksena on tuottaa ei-natiivista lähdekoodista natiivikoodia sekä iOS- että Android-mobiilikäyttöjärjestelmille. Kääntämisprosessi riippuu hieman käytetystä kehityskehyksestä. Esimerkiksi React Native -mobiilisovelluksessa käytetään JavaScriptiä, joka siltayhteyden kautta muunnetaan natiivikomponenteiksi JavaScript-ajoaikaa käyttäen React Nativen virallisen dokumentaation mukaan (Communication between native and React Native, 2024). Vaikka JavaScript ei

itsessään siis käänny natiivikoodiksi, React Native muuntaa JavaScript-komponentit natiiveiksi käyttöliittymäelementeiksi.

Alustariippumattomassa mobiilisovelluksessa voidaan hyödyntää lähes kaikkia mobiililaitteeseen sisäänrakennettuja ominaisuuksia, vaikkakin niiden käyttöön tarvitaan usein kolmannen osapuolen tarjoamia kirjastoja. Alustariippumattoman mobiilisovelluksen suorituskyky on muihin ei-natiiveihin sovelluksiin verrattuna paras, mutta ei kuitenkaan aivan yhtä hyvä kuin natiivin mobiilisovelluksen.

Hybridi-sovellus (engl. *hybrid app*) on ei-natiivi mobiilisovellus, joka on kirjoitettu web-sovelluskehitykseen tarkoitetuilla kielillä ja kirjastoilla. Hybridi-sovellus ei käännä lähdekoodia ollenkaan natiivikoodiksi, vaan hybridisovellus paketoidaan natiivisovelluksen kehykseen. Natiivisovelluksen kehys, WebView, on ikään kuin mini-verkkoselain, johon mobiilisovelluksen sisältö renderöidään. Tämän vuoksi hybridi-sovellus on aina pitkälti riippuvainen internet-yhteydestä.

Ionic ja Apache Cordova ovat tunnetuimpia työkaluja, joilla hybridi-sovelluksia kehitetään (Medium, 2020). Hybridi-sovelluksessa voidaan hyödyntää joitakin mobiililaitteen ominaisuuksia kuten GPS:ää, kameraa ja push-ilmoituksia, mutta esimerkiksi sormenjäljen tunnistusta, kompassia ja mobiililaitteen yhteystietoja ei voida hyödyntää. Hybridi-sovelluksen suorituskyky on heikompi kuin natiivin mobiilisovelluksen ja alustariippumattoman mobiilisovelluksen.

PWA-sovellus (engl. *Progressive Web App*) eli progressiivinen verkkosovellus on ei-natiivi mobiilisovellus, joka on hybridi-sovelluksen nuorempi serkku. PWA-sovellukset ovat pohjimmiltaan web-sovelluksia, jotka ajetaan suoraan verkkopalvelimelta tai välimuistista Service Worker -tekniikkaa hyödyntäen. Service Worker -tekniikalla voi jäljitellä joitakin natiivisovelluksen ominaisuuksia. PWA-sovelluksia on mahdollista luoda erilaisilla kolmansien osapuolien low-code ja no-code työkaluilla, kuten esimerkiksi Shopify:lla ja Bubble:lla. PWA-sovelluksen suorituskyky riippuu käytetystä selaimesta. Joitakin PWA-sovelluksia voi julkaista Google Play -sovelluskaupassa mutta ei App Storessa.

2.3 Käyttäjäkokemuksen merkitys mobiilisovelluksissa

Markkinoilla menestyvä mobiilisovellus koukuttaa käyttäjänsä, koska mobiilisovelluksen käyttö on miellyttävää, sujuvaa ja palkitsevaa. Käyttäjä palaa sovellukseen, kun sen käyttöliittymä on intuitiivinen, käyttö helppoa ja sovellus toimii moitteettomasti kaikissa tilanteissa. Mobiilisovelluksen käyttäjäkokemuksen suunnittelussa on otettava huomioon monenlaisia eri tekijöitä.

Mobiilisovelluksen suorituskyky on avainasemassa käyttäjäkokemuksen kannalta. Hitaasti latautuvat sivut ja näkymät sekä äkilliset kaatumiset ovat käyttäjälle äärimmäisen turhauttavia, jonka seurauksena käyttäjä saattaa päättää, ettei halua käyttää sovellusta enää ikinä uudelleen. Mobiilisovelluksen suorituskykyä optimoimalla varmistetaan, että sovellus latautuu nopeasti, vastaa käyttäjän toimintoihin ilman viiveitä ja toimii vakaasti kaikissa tilanteissa ja ympäristöissä.

Helppokäyttöisyys ja selkeys ovat myös olennaisia ominaisuuksia hyvän mobiilikäyttäjäkokemuksen luomisessa. Klikattavien alueiden on oltava riittävän suuria, jotta niitä on helppo käyttää pienemmilläkin kosketusnäytöillä. Mobiilisovelluksen viestinnän tulee olla täsmällistä ja johdonmukaista: esimerkiksi pelkkä ”jatka”-nappi ei välttämättä riitä ohjaamaan käyttäjää oikeaan suuntaan, kun taas ”jatka maksamaan” antaa selkeämmän käsityksen toiminnon tarkoituksesta. Selkeä ja tarkka viestintä vähentää käyttäjän tekemien virheiden mahdollisuutta, ja tekee käytöstä loogisempaa.

Käyttäjäkokemuksen kannalta käyttäjäpolut (engl. *user flows*) ovat tärkeitä, sillä ne ohjaavat käyttäjää sovelluksen eri toimintojen läpi mahdollisimman suoraviivaisesti. UI/UX- suunnittelijan tehtävänä on varmistaa, että käyttäjä löytää tarvitsemansa toiminnot helposti ja ymmärtää aina seuraavan askeleen. Hyvin suunnitellut käyttäjäpolut lisäävät mobiilisovelluksen käytön mukavuutta ja vähentävät hämmennystä.

Saavutettavuusominaisuudet tekevät mobiilisovelluksesta käytettävän ihan kaikille esimerkiksi toimintarajoitteista tai kulttuuritaustasta huolimatta.

Näkörajoitteisia käyttäjiä voi tukea suuremmilla ja dynaamisilla fonttikoilla. Mobiilisovellukseen on mahdollista luoda tumma tila (engl. *dark mode*), joka auttaa valoherkkiä käyttäjiä. Sokeille käyttäjille puolestaan on tärkeää, että mobiilisovellus on yhteensopiva erilaisten ruudunlukuohjelmien kanssa. Kognitiivisesti rajoittuneita käyttäjiä voi huomioida esimerkiksi välttämällä välkkyviä animaatioita mobiilisovelluksessa. (Kauriola, 2021)

Mobiilisovelluksen kulttuurinen saavutettavuus tarkoittaa esimerkiksi, että mobiilisovelluksen käyttöliittymä on saatavilla kohdemaassa käytetyllä kielellä. Mobiilisovelluksen käyttökokemus tulisi olla tasavertaista eri kielille. Kulttuurista saavutettavuutta mobiilisovelluksessa on kielen lisäksi esimerkiksi se, että päivämäärät ja kellonajat näytetään kulttuurikohtaisessa formaatissa. Värit symboloivat myös eri asioita riippuen kulttuurista. Länsimaissa esimerkiksi valkoinen väri symboloi puhtautta, viattomuutta ja rauhaa. Itä-Aasiassa valkoinen väri puolestaan liitetään kuolemaan ja suruun. Jos mobiilisovelluksessa käytetään valkoista teemaväriä juhlahetkiin, kuten vaikka syntymäpäiväilmoituksiin ja onnittelubannereihin, se saattaa viestiä jotain aivan muuta joissakin kulttuureissa.

Tietoturva on olennainen osa käyttäjäkokemusta, vaikka sitä ei aina nähdä perinteisenä UX-elementtinä. Tietoturva liittyy käyttäjäkokemukseen siksi, että turvallisuus ja yksityisyys ovat käyttäjille tärkeitä ja vaikuttavat suoraan siihen, miten luottavaisesti ja sujuvasti he käyttävät mobiilisovellusta. Turvallisuuden tunteen puute voi huonontaa käyttäjäkokemusta merkittävästi, kun taas hyvä tietoturva ja sen huomaamaton toteutus parantavat käyttökokemusta. Tietoturvaan panostaminen parantaa paitsi käyttäjäkokemusta myös yrityksen ja tuotteen mainetta ja uskottavuutta. (Peris.ai, 2024)

Lähtökohtaisesti mobiilisovelluksen kaikki ominaisuudet tulisi olla samanlaisia mobiilikäyttöjärjestelmästä riippumatta, jotta mobiilisovellus ei aseta käyttäjiä epätasa-arvoiseen asemaan käytetyn mobiililaitteen takia. Tässä on kuitenkin pieniä poikkeuksia. Apple julkaisi iOS 16 käyttöjärjestelmäpäivityksen yhteydessä ominaisuuden nimeltä ”Live Activities” (Apple Support, 2024), jota voidaan hyödyntää joissakin iOS-mobiilisovelluksissa. Tälle ominaisuudelle ei ole vielä olemassa täysin vastaavaa toiminnallisuutta Android-käyttöjärjestelmän

mobiililaitteissa eikä mobiilisovelluksissa. Internetistä löytyy vaihtoehtoisia tapoja simuloida ”Live Activities” ominaisuutta Android-laitteilla, kuten esimerkiksi käyttämällä RemoteViews-komponenttia, jota voidaan hyödyntää ainakin Flutter-kehityksessä (Fontenele, 2024).

3 KÄYTTÖLIITTYMÄN SUUNNITTELU MOBIILIALUSTOILLE

3.1 Käyttäjäroolien ymmärtäminen ja käyttöliittymän vaatimukset

Mobiilisovelluksen suunnittelu alkaa käyttäjäroolien ymmärtämisestä. Käyttäjäroolit ovat abstraktioita sovelluksen todellisista käyttäjistä, ja ne kuvaavat heidän tarpeitaan, tavoitteitaan ja odotuksiaan. Käyttäjäroolien ymmärtäminen auttaa varmistamaan, että sovellus tarjoaa oikeita toiminnallisuuksia oikeille käyttäjille.

Käyttäjäroolimatriisi (kuva 2) on hyödyllinen työkalu erityisesti, jos sovelluksessa on useita eri käyttäjätyyppejä tai monimutkaisia toiminnallisuuksia. Matriisissa kuvataan käyttäjäryhmien keskeiset tavoitteet, mahdolliset rajoitteet ja heidän prioriteettinsa. Käyttäjäroolimatriisi auttaa kehitystiimiä priorisoimaan sovelluksen toiminnallisuuksia ja keskittymään niihin ominaisuuksiin, jotka tuottavat eniten arvoa eri käyttäjäryhmille.

	opiskelija	opettaja	huoltaja	opinto-ohjaaja
oma edistyminen	✅	✅	✅	✅
oppimateriaali	✅	✅	❌	❌
kurssien hallinta	❌	✅	❌	❌
arvioinnit ja palaute	✅	✅	❌	❌
viestintä ja keskustelut	✅	✅	✅	✅
opiskelijan suoritustiedot	❌	✅	✅	✅
käyttäjien hallinta	❌	❌	❌	❌
analytiikka/ raportit	❌	✅	✅	✅

KUVA 2. Esimerkki käyttäjäroolimatriisista, jossa esitellään sovelluksen ominaisuudet ja käyttäjäroolit.

Kun käyttäjärooleista on saatu perustavanlaaatuinen käsitys, seuraavaksi on määriteltävä käyttöliittymän vaatimukset. Mobiilisovelluksen ensimmäiseen prototyyppiin kannattaa valita vain kaikista kriittisimmät ominaisuudet, jotka

tukevat sovelluksen ydintoiminnallisuutta. Käyttöliittymän ominaisuuksien priorisoinnissa voidaan käyttää esimerkiksi MoSCoW -menetelmää, joka auttaa selkeästi määrittelemään, mitkä ominaisuudet ovat välttämättömiä ja mitkä voivat odottaa (MoSCoW Priorization Technique in Product Managment, 2024). MoSCoW -menetelmä tulee sanoista ”must have”, ”should have”, ”could have” ja won’t have”, ja tarkoittaa vapaasti suomennettuna ”täytyy olla”, ”pitäisi olla”, ”voisi olla”, ”ei tule olemaan”. Muita priorisointimenetelmiä ovat esimerkiksi RICE ja Kano.

Yksi yleisimmistä ongelmista mobiilisovelluskehityksessä on niin sanottu ”scope creep”, eli projektin laajeneminen suunnitteluvaiheen jälkeen. Tämän ilmiön välttämiseksi käyttöliittymän vaatimukset tulisi dokumentoida huolellisesti, ja kommunikoida niistä selkeästi kaikille eri sidosryhmille. Käyttöliittymän dokumentaatio toimii viitekehyksenä koko kehitysprosessin ajan, ja sen avulla voidaan myös seurata, pysyykö projekti alkuperäisissä tavoitteissaan. (Martins, 2024)

3.2 Komponenttipohjainen suunnittelu

Käyttöliittymäkomponentilla tarkoitetaan sovelluksen visuaalisia elementtejä, kuten nappuloita, syöttökenttiä, listoista ja taulukoista koostuvia osia, valikoita ja muita näkymien rakennuspalikoita. Käyttöliittymäkomponentit ovat itsenäisiä yksiköitä, jotka voivat toimia itsenäisesti, mutta yhdessä ne luovat toimivan käyttöliittymän.

UI-suunnittelijan on hyvä jo sovelluksen suunnitteluvaiheessa jakaa mobiilisovellus pienempiin ja hallittavampiin osiin, sillä se helpottaa ohjelmistokehittäjän työtä ja parantaa sovelluksen skaalautuvuutta. Komponenttipohjainen lähestymistapa tuo etuja myös sovelluksen ylläpidon ja laajennettavuuden kannalta, sillä yksittäisiä komponentteja voidaan helposti päivittää, vaihtaa tai uudelleenkäyttää ilman, että koko sovelluksen toiminnallisuuteen tai käyttöliittymään tarvitsisi tehdä suuria muutoksia. Näin ohjelmistokehittäjä voi keskittyä yksittäisten komponenttien toteuttamiseen ja testaukseen ilman, että sovelluksen kokonaisuus hajoaa. Lisäksi komponenttipohjainen lähestymistapa tukee hyvää ohjelmistosuunnittelua, kuten

yksikkökehitystä ja iteratiivista kehitystä, sekä parantaa yhteistyötä suunnittelijoiden ja kehittäjien välillä. (Figma, n.d)

Lähtökohtaisesti mobiilisovelluksen tulisi toimia ja näyttää hyvältä niin älypuhelimilla kuin tableteilla. Tätä kutsutaan responsiivisuudeksi. Käyttöliittymäkomponenttien suunnitteluvaiheessa olisi hyvä ottaa huomioon miltä eri käyttöliittymäkomponentit näyttävät eri kokoisilla näytöillä. Moni suunnittelutyökalu, kuten Figma, tukee responsiivista suunnittelua. Lisäksi käyttöliittymäsuunnittelun yhteydessä on päätettävä, tuleeko mobiilisovelluksen tukea horisontaalisia näkymiä, kun mobiililaite käännetään vaakasuuntaan.

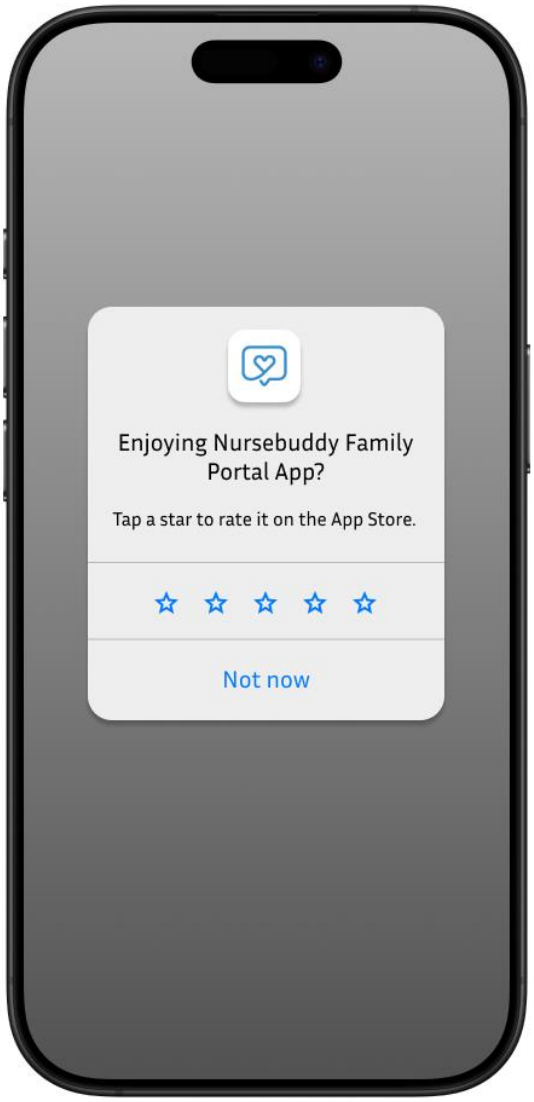
3.3 Käyttäjäpalautteen kerääminen

Käytettävyystestaus ja käyttäjäpalautteen systemaattinen kerääminen ovat keskeisiä vaiheita mobiilisovelluksen käyttöliittymän suunnitteluvaiheessa. Käytettävyystestauksen tavoitteena on varmistaa, että sovellus vastaa käyttäjien tarpeita ja odotuksia sekä tarjoaa sujuvaan käyttökokemuksen. Testaamalla sovellusta todellisten käyttäjien kanssa voidaan tunnistaa mahdolliset ongelmakohdat, kuten käyttöliittymän epäloogisuudet, hankalat navigointipolut tai muut epäselvät toiminnot. Näiden havaintojen pohjalta suunnittelua voidaan hioa paremmin käyttäjälähtöiseksi.

Käyttäjäpalautteen kerääminen täydentää käytettävyystestausta tarjoamalla syvällisempiä näkemyksiä käyttäjäkokemuksista. Käyttäjäpalautteen kerääminen voi tapahtua esimerkiksi kyselylomakkeiden, haastatteluiden tai sovellukseen integroitujen palautetyökalujen avulla. Käyttäjäpalautteen keräämiseen mobiilisovelluskessa on tarjolla monia menetelmiä, jotka voidaan jakaa passiivisiin ja aktiivisiin lähestymistapoihin.

Passiiviset menetelmät viittaavat sovelluksen sisäiseen analytiikkaan, jotka keräävät tietoa käyttäjien toiminnasta taustalla ilman heidän aktiivista osallistumistaan. Sovelluksen sisäisellä analytiikalla voidaan seurata esimerkiksi mobiilisovelluksen suosituimpia ominaisuuksia, käyttäjäpolkuja ja sovelluksesta poistumisen hetkiä.

Aktiivisiin menetelmiin kuuluvat esimerkiksi sovelluksen sisäiset kyselyt, palautelomakkeet ja suorat arvostelupyynnöt (kuva 3), jotka antavat käyttäjille mahdollisuuden ilmaista mielipiteensä ja ehdotuksensa. Push-ilmoitusten avulla voidaan ohjata käyttäjiä antamaan palautetta tiettyjen toimintojen tai päivitysten jälkeen. Aktiivisen käyttäjäpalautteen keräämisen suunnittelussa on päätettävä, kuinka usein käyttäjää kehotetaan antamaan palautetta, kuinka usein palautelomakkeen kysymyksiä vaihdetaan, sekä kuinka vastauksia analysoidaan.



KUVA 3. Suora arvostelupyyntö kuuluu aktiivisiin menetelmiin käyttäjäpalautteen keräämissä.

Hyvin toteutettu käytettävyystestaus ja käyttäjäpalautteen hyödyntäminen tukevat mobiilisovelluskehitystä tarjoamalla tieto- ja tutkimuspohjaa iteratiiviseen suunnitteluprosessiin. Käyttäjien tarpeiden ja mieltymysten ymmärtäminen auttaa kehitystiimiä tekemään perusteltuja päätöksiä, jotka johtavat paremmin toimivaan ja käyttäjäystävälliseen lopputuotteeseen. Siten varmistetaan, että mobiilisovellus ei ole pelkästään teknisesti toimiva, vaan myös käytännöllinen ja miellyttävä käyttää.

4 TEKNINEN SUUNNITELMA

4.1 Valitut teknologiat ja kehitystyökalut

Nursebuddyn Omaisnäkymä halutaan toteuttaa ei-natiivina alustariippumattomana sovelluksena, koska pienellä kehitystiimillä ei ole resursseja kehittää ja ylläpitää natiivisovelluksia useille eri alustoille. Cross-platform-ratkaisu tarjoaa kustannustehokkaan ja aikaa säästävän vaihtoehdon, koska lähdekoodi kirjoitetaan yhdellä ohjelmointikielellä, jolloin samaa koodipohjaa voidaan käyttää sekä iOS- että Android-käyttöjärjestelmille.

Alustariippumaton lähestymistapa säästää merkittävästi kehitysaikaa ja -kustannuksia. Koska lähdekoodi on yhteinen eri alustoille, uuden toiminnallisuuden tai korjausten lisääminen on yksinkertaisempaa ja nopeampaa, eikä vaadi erillisiä kehitysprosesseja eri mobiilikäyttöjärjestelmille. Tämä myös helpottaa ylläpitoa ja mahdollisten bugien korjaamista, kun muutokset voi tehdä keskitetysti yhteen koodipohjaan.

Cross-platform-teknologia mahdollistaa nopeamman markkinoille pääsyn, sillä kehityssyklit ovat lyhyempiä kuin natiivisovelluksissa. Nopeampi julkaisu voi antaa etumatkaa kilpailijoihin nähden ja mahdollistaa käyttäjäpalautteen keräämisen aikaisessa vaiheessa, mikä auttaa sovelluksen jatkuvassa kehittämisessä.

Nursebuddyn Omaisnäkymän kaltaisessa alustariippumattomassa mobiilisovelluksessa kehysvalinta on keskeinen päätös, joka vaikuttaa suoraan kehitysprosessiin, suorituskykyyn ja lopputuotteen laatuun. Tällä hetkellä suosituimmat cross-platform kehityskehykset ovat React Native ja Flutter, joista molemmat mahdollistavat sovellusten rakentamisen sekä iOS- että Android-alustoille yhden koodipohjan avulla.

React Native on Facebookin kehittämä kehityskehys, joka perustuu JavaScript-kieleen ja React-kirjastoon (React Native, n.d.). Se on luonnollinen teknologiavalinta, jos kehitystiimillä on jo kokemusta Reactista, sillä sen avulla

voidaan hyödyntää verkkosovelluksista tuttuja käyttöliittymäkomponentteja mobiilikehityksessä. Tämä tuo yhtenäisen ilmeen ja tuntuman tuoteperheen eri sovelluksiin ja nopeuttaa kehitystä. React Native luottaa natiivikomponentteihin, mikä tarkoittaa, että käyttöliittymän osat mukautuvat eri käyttöjärjestelmien tyyliin. Tämä tekee sovelluksesta natiivinoloisen sekä iOS- että Andoird-alustoilla.

Flutter puolestaan on Googlen kehittämä cross-platform-kehys, joka käyttää Dart-ohjelmointikieltä (Flutter, n.d.). Flutter on optimoitu erityisesti mobiilisovelluskehitystä varten ja poikkeaa React Nativesta siinä, että se ei käytä natiivikomponentteja, vaan piirtää käyttöliittymänsä omien widget-komponenttiensa avulla. Tämä mahdollistaa täysin yhtenäisen ulkoasun kaikilla alustoilla. Flutterin etuna on sen tarjoama erinomainen suorituskyky, sillä se piirtää käyttöliittymän suoraan, eikä siihen tarvita siltayhteyden kaltaisia välikerroksia. Tämä tekee Flutterista sopivan vaihtoehdon visuaalisesti vaativille sovelluksille, joissa tarvitaan monimutkaisia animaatioita ja tasaista suorituskykyä.

Suorituskyvyn ja käyttöliittymän yhtenäisyyden lisäksi kehysvalintaan vaikuttaa yhteisön tuki. React Native hyötyy laajasta JavaScript- ja React-kehittäjien yhteisöstä, mikä mahdollistaa monien valmiiden kirjastojen ja moduulien hyödyntämisen. Flutterin yhteisö on pienempi, mutta kehittyvä, ja se tarjoaa valmiita työkaluja, jotka tukevat esimerkiksi web- ja desktop-käyttöä. Flutter on oiva valinta, jos halutaan rakentaa sovellus, joka laajenee helposti myös muihin kanaviin kuin mobiiliin.

React Native on hyvä valinta, kun kehitystiimillä on jo JavaScript-osaamista tai kun tavoitteena on käyttää samoja komponentteja verkkosovelluksen ja mobiilisovelluksen välillä. Kehyksen valinnassa on otettava huomioon tiimin osaaminen, projektin vaatimukset ja se, mitä alustoja halutaan tukea nyt ja tulevaisuudessa.

Expo on avoimeen lähdekoodiin perustuva React Native -kehitysalusta, joka on suunniteltu nopeuttamaan mobiilisovellusten kehittämistä (Expo, 2024). Expon avulla React Native -kehityksen aloittaminen on helppoa, koska se ei edellytä monimutkaisia natiivikehitysympäristöjä, kuten Xcodea tai Android Studiota.

Expo tarjoaa laajan kokoelman käyttövalmiita komponentteja ja API-kirjastoja, joiden avulla voidaan hyödyntää esimerkiksi kameran ja sijaintipalveluiden kaltaisia mobiilitoimintoja ilman monimutkaisia natiivi-integraatioita.

Expo tukee reaaliaikaista kehitystä hyödyntäen hot reload -ominaisuutta, jonka ansiosta kehittäjät näkevät sovellukseen tehdyt muutokset välittömästi ilman tarvetta käynnistää sitä joka kerta uudelleen. Hot reload nopeuttaa iterointia ja helpottaa käyttöliittymän visuaalisten muutosten testaamista reaaliaikaisesti. (Development and production modes, 2024)

Expon vahvuuksia on myös sen Over-the-Air (OTA) -päivitysten tuki, jonka ansiosta sovelluksen JavaScript-koodi ja resurssit voidaan päivittää suoraan käyttäjien laitteille ilman, että sovellusta täytyy julkaista uudelleen App Storessa tai Google Playssa. OTA-päivitykset mahdollistavat nopean reagoinnin esimerkiksi bugikorjauksiin ja parannuksiin, mikä tekee sovelluksen ylläpidosta joustavaa ja kustannustehokasta. (Send over-the-air updates, 2024)

Jos sovellus vaatii sellaisia natiivitoimintoja, joita Expon rajallisempi ympäristö ei tue, Expo-projekti oli ennen mahdollista ejectata eli siirtää React Native CLI -ympäristöön, jolloin kehittäjillä oli täysi pääsy natiivikoodiin. Ejectaus vei projektin pois Expon hallitsemasta ympäristöstä, jolloin kehittäjän täytyi huolehtia itse natiivikehityksestä ja rakentamisesta, mikä saattaa lisätä kompleksisuutta ja kehitysaikaa. Expo eject on vanhentunut, eikä sitä suositella enää käytettävän. Expo eject on korvattu Expo Prebuildilla, mutta Prebuild toimii lähes täysin identtisellä tavalla ejectaukseen nähden (Prebuild, 2024).

Jos tavoitteeena on kehittää mobiilisovellus sekä iOS-laitteille että Android-laitteille, mobiilisovellus on helpointa ohjelmoida Applen Mac-tietokoneella, koska macOS on ainoa käyttöjärjestelmä, joka antaa ohjelmistokehittäjän asentaa ja käyttää Xcode-ohjelmistoa. Xcode-ohjelmisto tarjoaa työkalut iOS, macOS, wathcOS ja tvOS sovellusten kehittämiseen. Näitä kehitystyökaluja ohjelmistokehittäjä tarvitsee voidakseen luoda, testata ja julkaista sovelluksen Applen laitteille. Mac-tietokoneelle saa ladattua Android Studion ja sitä kautta erilaisia Android-emulaattoreita. Mac-tietokone on myös riittävän tehokas eri kehitystyökalujen ja ohjelmistojen pyörittämiseen.

Jos Mac-tietokone ei ole vaihtoehto esimerkiksi tiukemman budjetin takia, seuraavaksi paras vaihtoehto on Linux-käyttöjärjestelmän tietokone. Linux on riittävän tehokas pyörittääkseen mobiilisovellukseen tarvittavia kehitystyökaluja, kuten Android Studiota. Linuxilla iOS-kehittäminen on kuitenkin hieman haastavampaa, koska iOS-emulaattoria ei suoraan Linuxille saa asennettua. Linuxilla on mahdollista vuokrata macOS pilvipalvelin, ottaa käyttöön virtuaalikone, tai yhdistää Linux-tietokone Mac-tietokoneeseen lopullista iOS-mobiilisovelluksen testausta ja julkaisua varten.

4.2 Koodin rakenne ja modulaarisuus

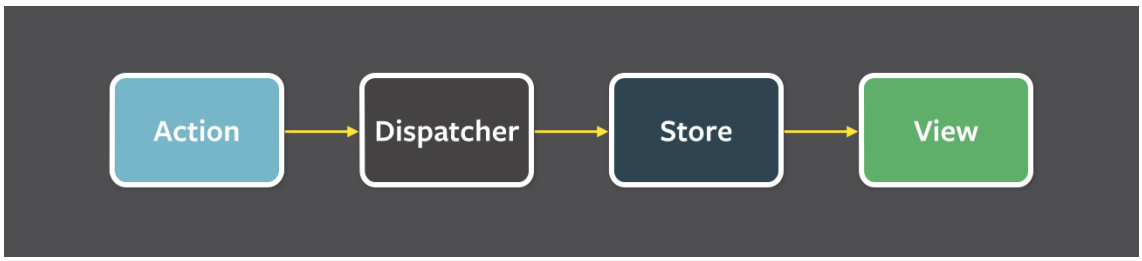
Sovelluksen modulaarisuudella tarkoitetaan ohjelmistokoodin rakenteen jakamista pienempiin, itsenäisiin osiin, eli moduleihin, jotka voivat toimia itsenäisesti mutta myös osana suurempaa järjestelmää. Modulaarinen koodi on jaettu loogisiin osiin, jotka ovat erillisiä mutta yhteensopivia.

Modulaarisuuden etuja ovat muun muassa koodin uudelleenkäytettävyys, koska yksittäisiä moduuleja voi käyttää useassa eri paikassa ilman, että koko järjestelmää tarvitsee muokata. Modulaarisuus helpottaa koodin hallintaa, sillä ohjelmoija voi keskittyä tiettyyn osaan ilman, että tarvitsee ymmärtää koko järjestelmän toimintaa. Modulaarisuus tukee parempaa virheenkorjausta ja testauskäytäntöjä, sillä pienempiä osia on helpompi testata erikseen ja varmistaa, että ne toimivat oikein ennen niiden yhdistämistä suurempaan kokonaisuuteen. Modulaarinen koodi parantaa tiimityöskentelyä, koska useat ohjelmistokehittäjät voivat työskennellä itsenäisesti eri moduulien parissa ilman, että heidän tarvitsee huolehtia muiden tekemistä muutoksista.

Tyypillisesti esimerkiksi käyttöliittymäkomponentit jaetaan koodissa omiksi, uudestaan käytettäviksi moduleiksi. Jos koodissa on paljon logiikkaa suorittavia funktioita, on suositeltavaa eristää ne erilleen esimerkiksi käyttöliittymäkomponenteista, jotta niiden uudelleenkäyttö on mahdollisimman yksinkertaista jatkossa. Ohjelmistoalalla on kahdenlaista koulukuntaa modulaarisuudesta, joka koodin lisäksi ulottuu aina tekniseen dokumentaatioon asti. Osa suosii ”WET”-periaatetta (Write Everything Twice) eli että koodi/ tekninen teksti saa osittain toistaa itseään, varsinkin jos toisto tapahtuu vain

harvoin eli noin 2-3 kertaa eri paikoissa. Toinen periaate on tiukempi, nimeltään ”DRY” (Don’t Repeat Yourself), jonka mukaan samaa koodia/ teknistä tekstiä ei tulisi missään tapauksissa kirjoittaa uudelleen eri yhteyteen (Hunt ja Thomas, 1999). Jokainen ohjelmistokehittäjä itse on lopulta vastuussa päättäessään omista periaatteistaan. Jos useamman ohjelmistokehittäjän projektiin valitaan jokin ohjelmointiperiaate, siitä on hyvä viestiä projektiryhmän sisällä riittävän selkeästi.

Tilanhallintamalli tarkoittaa sovelluskehityksessä tapaa ja periaatteita, joiden sovelluksen sisäistä tilaa hallitaan. Sovelluksen ”tila” (engl. *state*) viittaa kaikkiin tietoihin, joita sovellus tarvitsee toimiakseen ja joita se voi muuttaa, kuten käyttäjän syöttämät tiedot, sovelluksen näkymän tila, autentikointi- ja käyttäjäprofiilitiedot sekä käyttöliittymän eri tilat kuten latausvaihe ja virhetila. Flux on Facebookin kehittämä arkkitehtuurinen tilanhallintamalli, joka määrittelee tavan, jolla sovelluksen tilan hallinta ja muuttaminen tulisi tapahtua. Flux-mallissa tilan hallinta on yksisuuntainen ja perustuu neljään osaan (Flux, 2023).



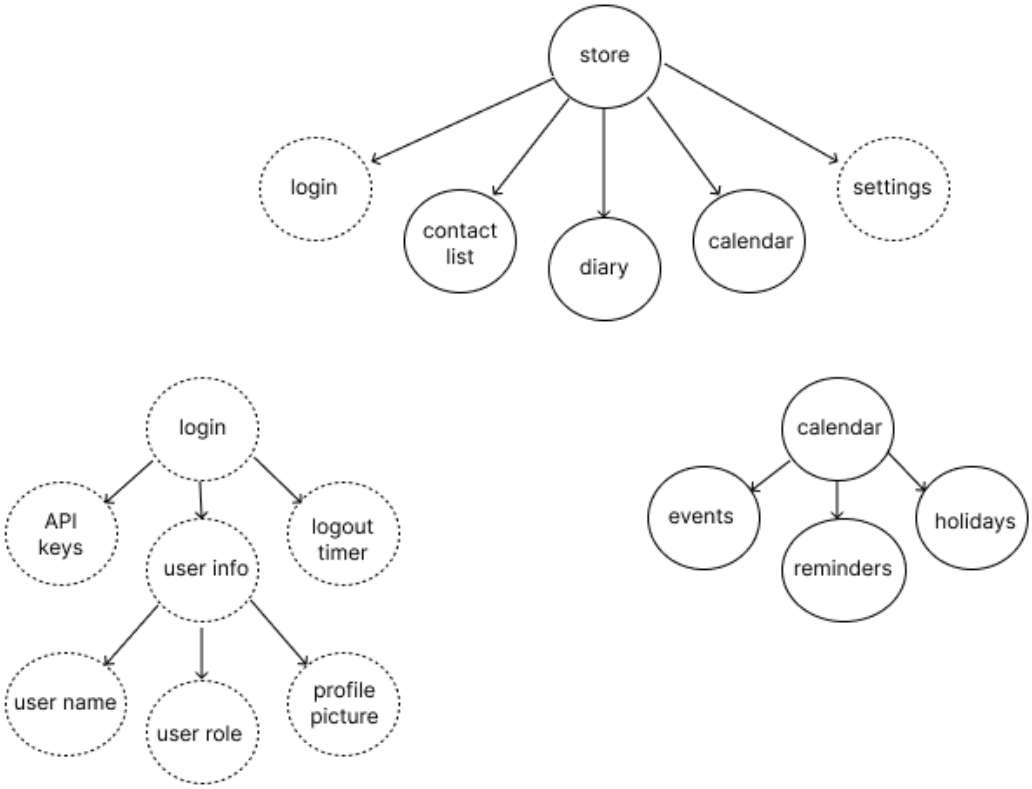
KUVA 4. Flux-mallin neljä osaa ovat ”Action”, ”Dispatcher”, ”Store” ja ”View” (Flux – In-Depth Overview, 2023)

Sovelluksen toiminta alkaa, kun käyttäjä suorittaa jonkin toiminnon, kuten painaa nappia mobiilisovelluksessa. Tämä toiminto laukaisee kuvassa 4 sinisellä taustavärillä esitetyn ”action”-objektin, joka kuvaa tapahtuman, kuten ”kirjaudu sisään”. ”Action” ei suoran muuta sovelluksen tilaa, vaan sen tehtävänä on välittää tietoa siitä, mitä pitäisi tapahtua.

”Action” lähetetään edelleen kuvan 4 tumman harmaalla taustavärillä esitetylle ”dispatcher”-komponentille, joka toimii eräänlaisena keskuskoordinaattorina. ”Dispatcher” välittää ”action”-objektin kaikille rekisteröidyille ”store”-komponenteille. ”Store” on sovelluksen tila- ja liiketoimintalogiikan keskus. Se

vastaanottaa ”action”-objektin, päivittää tarvittaessa sisäistä tilaansa ja käynnistää mahdollisia muutoksia sen perusteella. Kun ”store”-tila muuttuu, kuvan 4 vihreällä taustavärillä korostettu ”view”-komponentti, joka on sidottu kyseiseen ”store”-tilaan, päivittää käyttöliittymän vastaamaan uutta tilannetta. ”View” ei muokkaa ”store”-tilaa suoraan, vaan ainoastaan esittää sen visuaalisesti käyttäjälleen.

Redux on Flux-arkkitehtuurin pohjalta kehitetty kirjasto, joka tekee tilanhallinnasta hieman Fluxia yksinkertaisempaa. Fluxissa voi olla useita storeja, mutta Reduxissa on lähtökohtaisesti vain yksi store, johon tallennetaan koko sovelluksen tila (Geeks for Geeks, 2023). Sovelluksen sisäistä Redux-tilaa voi suunnitella esimerkiksi hahmottelemalla ”tilapuun” (engl. *state tree*) (Lusan Das, 2018). Kuvassa 5 käyttöliittymän näkymäkohtaiset tilat on ympyröity tasaisella viivalla Globaalit, koko ohjelmiston käytössä olevat tilat, on ympyröity katkoviivalla. Yksityiskohtaisesti rakennettu suunnitelma Redux-tilapuusta voi esittää myös objekti-rakenteita.



KUVA 5. Esimerkki Redux-tilapuusta.

4.3 API-kehityksen suunnittelu

API-kehityksellä tarkoitetaan ohjelmointirajapintojen suunnittelua, toteuttamista ja ylläpitoa. API:t mahdollistavat eri ohjelmistojen välisen viestinnän standardisoidulla tavalla. Esimerkiksi mobiilisovellukset hyödyntävät API-rajapintoja, jotta ne voivat kommunikoida pilvipalveluiden, tietokantojen ja muiden taustajärjestelmien kanssa.

Nursebuddy on SaaS-sovellus. Nursebuddy on rakennettu hyödyntäen Amazon Web Servicesin (AWS) tarjoamaa pilvi-infrastruktuuria. AWS on yksi maailman johtavista pilvipalvelualustoista, joka tarjoaa kattavan valikoiman erilaisia palveluita, kuten tietokantoja, tallennustilaa, laskentakapasiteettia ja analytiikkatyökaluja. AWS avulla yritykset voivat rakentaa skaalautuvia, turvallisia ja kustannustehokkaita järjestelmiä ilman tarvetta omaan fyysiseen infrastruktuuriin.

Mikropalvelu-pohjainen arkkitehtuuri ja monoliittinen arkkitehtuuri ovat kaksi erilaista tapaa rakentaa ohjelmistoja. Monoliittinen arkkitehtuuri tarkoittaa, että koko sovellus rakennetaan yhtenä suurena kokonaisuutena. Kaikki ominaisuudet ja toiminnot ovat osa yhtä ja samaa koodikantaa. Vaikka monoliittinen arkkitehtuuri on yksinkertaista rakentaa ja ottaa käyttöön pienissä projekteissa, se voi muuttua jäykäksi ja vaikeasti ylläpidettäväksi, kun sovellus kasvaa. (Richardson, 2018, kappale 1)

Mikropalvelu-pohjainen (engl. *microservices*) arkkitehtuuri tarkoittaa, että sovellus jaetaan useisiin pieniin ja itsenäisiin palveluihin, jotka kommunikoivat keskenään API-rajapintojen kautta. Jokaisella mikropalvelulla voi olla oma teknologiansa, tietokantansa ja kehitystiiminsä. Mikropalvelu-pohjainen arkkitehtuuri tekee sovelluksesta helpommin skaalautuvan, vikasietoisen ja muokattavan. Esimerkiksi jos yksi mikropalvelu tarvitsee päivityksen, sitä voidaan kehittää ja julkaista ilman vaikutusta muihin palveluihin.

Kun halutaan muuttaa olemassa oleva web-sovellus mobiilisovellukseksi, täytyy suunnitella, mitä muutoksia olemassa oleva mikropalveluarkkitehtuuri tarvitsee ja tarvitseeko API-rajapintoihin tehdä muutoksia. Mikropalveluarkkitehtuurissa

voidaan hyödyntää samoja mikropalveluita sekä web-sovelluksessa että mobiilisovelluksessa, mutta API:n toimintaa ohjaava gateway saattaa vaatia päivityksiä. Gateway toimii yhtenä sisäänkäyntipisteenä, jonka kautta kaikki sovelluksen käyttäjien tekemät pyynnöt välittyvät mikropalveluille. Gateway yhdistää, muokkaa ja reitittää pyyntöjä oikeille mikropalveluille, ja sen avulla voidaan hallita tietoturvaa, autentikointia ja liikenteen kuormitusta. Käytännössä gateway siis toimii välittäjänä mobiilisovelluksen ja taustajärjestelmien välillä.

Mobiilisovelluksen API-kehitys alkaa huolellisella suunnittelulla, jonka yhteydessä määritellään tarvittavat endpointit eli rajapintojen yksittäiset toiminnallisuudet. Käytännössä tämä tarkoittaa, että kehittäjät kartoittavat, mitä tietoja mobiilisovellus tarvitsee taustajärjestelmistä ja millaisia toimintoja sovellus suorittaa, kuten esimerkiksi käyttäjän kirjautuminen, tietojen hakeminen ja tietojen tallennus.

Hyvin suunniteltu API on dokumentoitu huolellisesti, jotta kehittäjät voivat helposti ymmärtää, miten sitä käytetään. Dokumentointi nopeuttaa mobiilisovelluksen kehitystä ja varmistaa, että rajapinta täyttää sekä tekniset että käyttäjäkokemukseen liittyvät vaatimukset.

4.4 Sovelluksen optimointi ja suorituskyvyn parantaminen

Mobiilisovelluksen optimoinnin tavoitteena on varmistaa, että sovellus tarjoaa käyttäjille mahdollisimman sujuvan, nopean ja virheettömän kokemuksen samalla, kun se hyödyntää laitteiden resursseja, kuten prosessitehoa, akkua ja muistia, mahdollisimman säästeliäästi.

Mobiilisovelluksen optimoinnissa kannattaa tarkastella sovelluksen kokoa, eli käytännössä sitä paljonko mobiilisovellus vie mobiililaitteelta muistia. Nykyisin keskiverto mobiililaitteessa on 64 Gt tai 128 Gt muistia, mutta vanhemmissa mobiililaitteissa muistia voi olla huomattavasti vähemmänkin. On tavallista, että mobiililaitteelle ladataan useita kymmeniä eri sovelluksia. Älypuhelimen muistia vie myös laitteelle tallennetut kuvat ja videot. Tutkimuksen mukaan kuluttajalla on nykyisin keskimäärin 40 eri sovellusta ladattuna puhelimelleen, joista keskimäärin 9 eri mobiilisovellusta on päivittäisessä käytössä (TechJury, 2024).

Applen App Storeen ladattava sovellus saa olla nykyään maksimissaan 500 Mt iOS 9.0 ja sitä uudemmille iOS-käyttöjärjestelmien mobiililaitteille (Apple Developer, 2024). Googlen Play Storeen ladattavan sovelluksen maksimikoko on puolestaan 200 Mt (Play Console Help, n.d.). Nyrkkisääntönä voisi silti sanoa, että mitä pienempi sovelluksen koko sitä parempi (The Magic of App Size Reduction, 2023).

React Nativella rakennetun mobiilisovelluksen optimointi tapahtuu pitkälti kooditasolla. Taulukkoon 1 on kuvattu yleisimpiä optimointia edistäviä toimia React Native -koodikannassa.

Ominaisuus	Optimoitu	Ei optimoitu
Lista	FlatList komponentti	ListView komponentti
Console.log	Koodissa ei console.logeja	Koodissa (paljon) console.logeja
Tarpeettoman uudelleen renderöinnin estäminen	Käytössä React.memo ja React hookit useMemo, useCallback	Ei käytössä apukeinoja uudelleen renderöinnin estämiseen
Kirjastojen käyttö	Vain tarpeellisia kirjastoja käytössä, kirjastoja käytetty harkiten	Paljon eri kirjastoja käytössä ja/ tai käyttämättömiä, vanhentuneita kirjastoja
Animaatiot, navigointi, välilehdet	Ei turhia animaatioita, yksinkertainen navigointi- ja välilehti rakenne	Paljon monimutkaisia animaatioita, monimutkainen navigointi- ja välilehti rakenne
Hermes	Käytössä	Ei käytössä
MomentJS	Ei käytössä, korvattu jollain muulla kirjastolla	Käytössä, ja koodissa kutsutaan MomentJS kirjaston funktioita usein

TAULUKKO 1. Yleisimpiä keinoja React Native -mobiilisovelluksen optimointiin (React Native – Performance Overview, 2024; Sinha, 2022; Rahman, 2024)

React Native -kehityksessä hyödyllisiä ja yleisesti suositeltuja suorituskyvyn seurantatyökaluja ovat Flipper ja Firebase. Näitä työkaluja kannattaa ottaa käyttöön mobiilisovelluskehityksen varhaisessa vaiheessa, jotta sovelluksen suorituskykyä voidaan seurata, analysoida ja optimoida jatkuvasti kehitysprosessin aikana. Flipper on kehittäjäystävällinen työkalu, joka tarjoaa kattavat debuggausominaisuudet, kuten verkkoliikenteen tarkastelun, konsolilokit ja suorituskyvyn mittaustyökalut (Flipper, 2024). Firebase Performance Monitoring puolestaan auttaa seuraamaan mobiilisovelluksen todellista käyttäjäkokemusta tuotantoympäristössä (Firebase, 2024). Se tarjoaa tietoa esimerkiksi latausajoista, verkon viiveistä ja renderöintiongelmista.

4.5 Tietoturva ja yksityisyyden suoja mobiiliympäristössä

Tietoturva ja yksityisyyden suoja mobiilisovelluksessa ovat keskeisiä tekijöitä, erityisesti nykyisessä digitaalisten palvelujen aikakaudessa, jossa käyttäjien henkilökohtainen data on arvokasta ja herkkää. Mobiilisovelluksen kehityksessä on tärkeää huomioida ja suunnitella tietoturvaa jo hyvissä ajoin ennen mobiilisovelluksen toteuttamista.

GDPR eli Euroopan yleinen tietosuoja-asetus on Euroopan alueella vuodesta 2018 sovellettu asetus, jonka tarkoituksena on vahvistaa Euroopan Unionin alueella asuvien henkilöiden oikeuksia omiin henkilötietoihinsa. GDPR vaatii, että sovelluksen käyttäjät saavat selkeän tiedon siitä, mitä tietoja kerätään, kuinka niitä käytetään ja kuinka niitä suojataan. GDPR edellyttää, että käyttäjillä tulee olla oikeus tarkastella, muokata tai poistaa omia tietojaan. Tietojen käsittelyssä on noudatettava tiettyjä periaatteita, kuten tietojen minimointia ja tietojen säilytyksen rajoittamista vain tarpeelliseksi ajaksi. Mobiilisovelluksen suunnittelun näkökulmasta se tarkoittaa sitä, että tietoturvaominaisuudet on suunniteltava jo alkuvaiheessa, ja varmistettava, että sovelluksen käyttöehdot ja tietosuojakäytännöt ovat käyttäjille helposti saatavilla ja ymmärrettäviä. (GDPR EU, 2024)

Datan suojaaminen mobiilisovelluksessa edellyttää myös joidenkin teknisten toimenpiteiden toteuttamista, kuten datan enkryptausta. Datan enkryptauksella varmistetaan, että käyttäjien henkilökohtaiset tiedot, kuten maksutiedot, viestit ja

sijaintitiedot, pysyvät suojattuina ei pelkästään tietokannassa vaan jopa silloin, kun ne siirtyvät verkossa. Datan salaaminen estää kolmansia osapuolia pääsemästä käsiksi arkaluontoisiin tietoihin, mikä on erityisen tärkeää mobiilisovelluksissa, jotka usein luottavat langattomaan yhteyteen kuten Wi-Fi- tai mobiiliverkkoihin. Langattomat verkot ovat alttiimpia hyökkäyksille kuin kiinteät verkot. Erilaiset enkryptaustekniikat tarjoavat vahvaa suojaa ja varmistavat, että vain valtuutetut osapuolet voivat purkaa ja lukea salatun datan. Mobiilisovelluksen suunnitteluvaiheessa on tärkeää valita sopiva salausmenetelmä (Sharma, 2024).

Monivaiheinen todennus eli MFA tarjoaa lisäsuojakerroksen käyttäjän tunnistautumiselle. Pelkkä salasana ei riitä kirjautumiseen, vaan MFA-käyttäjältä vaaditaan useampi todennuskerros, kuten kertakäyttöinen koodi, biometrinen tunnistus tai push-ilmoitus, ennen kuin hän pääsee mobiilisovellukseen sisään. Mobiilisovelluksen suunnitteluvaiheessa on tärkeää valita, mitkä MFA-menetelmät sovelluksessa otetaan käyttöön ja miten ne integroidaan saumattomasti käyttäjäkokemukseen. MFA:n lisääminen ei saisi hankaloittaa liikaa kirjautumisprosessia, mutta sen tulisi kuitenkin tarjota riittävä turva käyttäjälle.

5 TESTAUS JA LAADUNVARMISTUS

5.1 Testaus mobiilisovellusympäristössä

Mobiilisovelluksen testauksen suunnitelu on kriittinen vaihe, joka vaikuttaa merkittävästi tuoteen laatuun ja käyttäjäkokemukseen. Testauksen suunnittelussa on otettava huomioon tiimin resurssit, kuten käytettävissä oleva aika ja henkilöstön määrä. Suuret ohjelmistoyritykset voivat hyödyntää erillisiä QA-tiimejä, jotka keskittyvät yksinomaan testaukseen ja laadunvarmistukseen, kun taas pienemmissä kehitystiimeissä testaus on usein kehittäjien vastuulla tai se voidaan jossain tapauksissa ulkoistaa. Resurssien rajallisuus korostaa entisestään tarvetta hyvin suunnitelluille testausprosesseille, jotta kehitystiimin aika voidaan käyttää tehokkaasti.

Testausstrategian valinnassa tulee huomioida mobiilisovelluksessa käytetyt teknologiat, sillä ne määrittelevät, mitä työkaluja ja testausmenetelmiä sovelletaan. Esimerkiksi React Native -pohjaisessa sovelluksessa yksikkötestit (engl. *unit tests*), integraatiotestit ja end-to-end-testit (E2E) muodostavat kattavan testauspatteriston (React Native Testing, 2024). Näiden yksityiskohtainen toteuttaminen vaatii kuitenkin huomattavia resursseja, ja siksi QA-tiimin tuki voi olla tarpeen. Myös erilaiset automatisoidut työkalut, kuten ESLint, ovat hyödyllisiä, sillä ne havaitsevat helposti yleisimpiä koodivirheitä. Lintterien käyttö ja käyttöönotto on vaivatonta ja skaalautuu hyvin pieniin tiimeihin, kunhan niiden konfiguraatio on riittävän tiukka.

Mobiilisovelluksen testaaminen fyysisellä laitteella on mobiilisovelluksen laadunvarmistuksessa välttämätöntä, sillä simulaattorit eivät pysty kattamaan kaikkia mobiilisovelluksen toiminnallisuuksia, kuten kameran käyttöä tai mobiililaitteen muiden antureiden toimintaa. Simulaattorit ovat hyödyllisiä kehitysvaiheessa, mutta ne eivät korvaa fyysisiä laitteita, erityisesti käyttöympäristön monimuotoisuuden testaamisessa. (Testscenario, 2024)

Mobiilisovelluksen suorituskykytestauksessa mitataan sovelluksen reagointinopeutta, resurssien kulutusta ja vakaata toimintaa erilaisissa

kuormitustilanteissa. Suorituskykytestauksen suunnittelussa on tärkeää määrittää realistiset käyttötilanteet ja rajata testaukset niihin. Esimerkiksi React Native -sovelluksen suorituskykytestaus voi sisältää animaatioiden sulavuuden, verkkopyyntöjen viiveen ja akun kulutuksen analysointia. Hyvin suunniteltu ja toteutettu suorituskykytestaus varmistaa, että sovellus toimii saumattomasti sekä uusimmilla että vanhemmilla laitteilla.

5.2 Sovelluksen jakelu- ja päivitysprosessit

Mobiilisovelluksen jakelu- ja päivityssuunnitelma on olennainen osa onnistunutta sovelluksen elinkaarta. Mobiilisovelluksen jakelun ensimmäinen askel on rekisteröityä sovelluskauppoihin, kuten Applen App Storeen ja Google Play Storeen, joihin pääsy edellyttää Apple Developer- ja Google Play Console -tilien perustamista. Näiden alustojen kautta sovelluksen kehittäjät voivat hallita sovelluksen julkaisua, päivityksiä, analytiikkaa ja käyttäjien antamia arvosteluja. Apple Developer -tilin käyttö tarjoaa mahdollisuuden päättää, halutaanko sovelluksen olevan käytettävissä myös macOS-laitteilla, mikä voi avata uusia käyttäjäsegmenttejä mutta tuo samalla mukanaan lisävaatimuksia käyttöliittymäsuunnittelulle ja tekniselle testaukselle.

Sovelluksen julkaisu- ja päivityssuunnitelmassa on tärkeää ottaa huomioon, mitä mobiilikäyttöjärjestelmän versioita tuetaan. Mobiilisovelluksen kehittäjän on tehtävä valinta suorituskyvyn, käyttäjäkunnan tarpeiden ja resurssien välillä. Esimerkiksi vanhimpien mobiilikäyttöjärjestelmäversioiden tukeminen voi kasvattaa kehitys- ja ylläpitokustannuksia huomattavasti ja rajoittaa mobiilisovelluksen ominaisuuksia. Sovelluskauppojen oma analytiikka tarjoaa tietoa yleisimmin käytetyistä käyttöjärjestelmäversioista. Tätä analytiikkaa hyödyntämällä voidaan tehdä harkittu päätös, mitä käyttöjärjestelmäversioita tuetaan.

Julkaisuaikataulua suunniteltaessa on huomioitava käyttäjäkunnan aktiivisuus. Esimerkiksi liiketoimintasovellusten päivitykset voidaan ajoittaa ilta-aikoihin tai viikonloppuihin, jolloin käyttäjien työskentelyyn kohdistuu vähemmän mahdollisia häiriöitä. Kuluttajille suunnattujen sovellusten osalta julkaisu voi tapahtua milloin tahansa, mutta on hyvä pitää mielessä, että julkaisuajankohta voi vaikuttaa

käyttäjien kokemukseen ja päivityksen vastaanottoon. Suuren käyttäjäkunnan sovelluksissa julkaisua tulisi välttää huipputuntien aikana. Samalla on myös tärkeää suunnitella julkaisuaikataulu siten, että kehitystiimi ja muu tukihenkilöstö ovat saatavilla heti julkaisun jälkeen. Näin mahdollistetaan nopea reagointi, mikäli julkaisussa ilmenee ongelmia, ja varmistetaan että käyttäjät saavat apua ilman viiveitä.

Over-the-air (OTA) -päivitykset tarjoavat mahdollisuuden toimittaa uusia ominaisuuksia ja virheenkorjauksia ilman, että käyttäjän tarvitsee päivittää sovellusta manuaalisesti sovelluskaupan kautta. OTA-päivitykset eivät itsessään ole täysin riskittömiä, sillä ne ovat jossain tapauksissa aiheuttaneet käyttäjien mobiililaitteisiin esimerkiksi jatkuvan uudelleenkäynnistys kierteen (Sorell, 2024). Ennen OTA-päivitystä tulisi olla tarkkana, että mobiilisovelluksen julkaistava versio on varmasti testattu asianmukaisesti.

Mahdollisten ongelmatilanteiden varalle on syytä laatia selkeä ”roll back” -suunnitelma, joka mahdollistaa edelliseen versioon palaamisen nopeasti, mikäli uusi julkaisu aiheutta merkittäviä häiriöitä. Asianmukainen projektin versionhallinta tukee roll back -suunnitelmaa. Roll back -suunnitelman tulisi sisältää selkeät toimintavaiheet myös siinä tapauksessa, että uusi julkaisu aiheuttaa tietokantamuutoksia. Tällaisten tilanteiden varalta tulee miettiä etukäteen, miten tietokannat voidaan palauttaa aikaisempaan tilaan vahingoittamatta olemassa olevia tietoja.

Hotfix-päivitykset ovat välttämättömiä kriittisten virheiden korjaamiseen nopeasti. Hotfix-päivitykset eivät yleensä sisällä uusia ominaisuuksia, vaan niiden tavoitteena on minimoida käyttäjäkokemuksen haitat ja ylläpitää sovelluksen luotettavuutta. Hotfix-päivityksiin kannattaa valmistautua suunnittelemalla etukäteen selkeä prosessi kiireellisten korjausten toteuttamiseen ja julkaisemiseen.

6 YLLÄPITO JA JATKOKEHITYS

6.1 Versionhallinta ja ylläpidon haasteet

Versionhallinta on tärkeää, vaikka kehitystiimi olisi pieni. Erilaiset versionhallintajärjestelmät, kuten Git, tarjoavat mahdollisuuden seurata muutoksia koodikannassa. Versionhallintatyökalu helpottaa ryhmätyöskentelyä, etenkin jos useampi kehittäjä työstää samaa koodikantaa. Versionhallinta minimoi inhimillisiä virheitä, kuten päällekkäisiä muutoksia tai vanhentuneen koodin käyttämistä. Versionhallintatyökalun käyttöönotto tekee ohjelmistoprojektista vakaamman.

Yksi mobiilisovelluksen ylläpidon prosesseista on virheiden monitorointi, joka tuo mukanaan omat haasteensa. Mobiilisovelluksen käyttäjät toimivat usein erilaisissa ympäristöissä, ja virheet voivat ilmetä ainoastaan tietyissä laite- tai käyttöjärjestelmäyhdistelmissä, mikä tekee virheiden jäljittämisestä ja korjaamisesta toisinaan haastavaa. Tehokkaiden virheidenseurantatyökalujen, kuten esimerkiksi Sentryn tai Firebase Crashlyticsin, avulla voidaan kerätä tietoa sovelluksen kaatumisista ja muista virhetilanteista.

Löydettyjen bugien priorisointi voi olla toisinaan haasteellista, sillä prioriteetit vaihtelevat liiketoimintavaatimusten, käyttäjäkokemuksen ja teknisten rajoitusten mukaan. Kriittisimmät virheet jotka estävät sovelluksen käytön, vaativat välitöntä huomiota, kun taas pienemmät käytettävyyshaasteet voidaan aikatauluttaa myöhempiin päivityksiin. Bugien priorisointi perustuu virheiden vaikutusten arviointiin, käyttäjäpalautteeseen ja kehitystiimin resurssien tehokkaaseen käyttöön.

Mobiilisovelluksen ylläpito tuo mukanaan monia muitakin haasteita, joista yksi on laitteistojen ja käyttöjärjestelmien moninaisuus. Eri valmistajien laitteet ja niiden vaihtelevat suorituskyvyt voivat vaikuttaa sovelluksen toimivuuteen. Androidin ja iOS:n jatkuvat käyttöjärjestelmäpäivitykset voivat rikkoa sovelluksen toiminnallisuuksia tai vaatia huomattaviakin muutoksia koodiin.

Toinen tavallisimmista haasteista on käyttäjien odotusten muuttuminen. Mobiilisovelluksia käyttävät ihmiset odottavat jatkuvaa kehitystä, kuten uusia ominaisuuksia, parannettua käyttökokemusta ja modernia ulkoasua. Trendien tai kilpailijoiden aiheuttamat odotukset voivat johtaa siihen, että kehitystiimi joutuu priorisoimaan nopeaa päivitystahtia, mikä voi vaarantaa koodin laadun tai projektin resurssoinnin pitkäjänteisempiin tavoitteisiin.

Jos käyttäjämäärä kasvaa nopeasti, skaalautuvuudesta voi tulla yksi kriittisimmistä haasteista ylläpidon näkökulmasta. Palvelinpuolen infrastruktuurin on kyettävä käsittelemään nopeasti lisääntyvä kuormitus ilman merkittäviä viiveitä tai käyttökatkoksia. Mobiilisovelluksen käytön kasvu saattaa edellyttää esimerkiksi tietokantojen optimointia, tehokkaampaa tiedonsiirtoa ja pilviresurssien tehokasta hallintaa.

Tekninen velka kasvaa uusien ominaisuuksien myötä. Teknisen velan ottoa on lähes mahdotonta kokonaan välttää. Teknistä velkaa lisäävät monet eri tekijät, kuten nopeat kompromissit ja tiukat aikataulut, vanhentuneet kirjastot, huono koodinlaatu ja puutteellinen testaus. Tekninen velka ei välttämättä ole heti näkyvää, mutta ajan myötä se voi aiheuttaa ongelmia, jotka hidastavat kehitystyötä ja nostavat ylläpitokustannuksia. Tekninen velka voidaan rinnastaa taloudelliseen velkaan: lyhyellä aikavälillä tehtyjen kompromissien seurauksena ”velka” kasvaa korkoa, jos sitä ei hoideta.

Tekninen velka ei ole aina huono asia. Joskus se on täysin tietoinen päätös, jolla saavutetaan nopeammin liiketoimintahyötyjä. On kuitenkin tärkeää pitää tekninen velka hallinnassa, tunnistaa sen määrä ja suunnitella, miten velka maksetaan takaisin. Teknisen velan hallintaan voi kuulua esimerkiksi koodin refaktorointi, jatkuva integraatio ja testausautomaatio, teknologioiden päivittäminen sekä koodin laadun valvonta. Hyvä dokumentaatio auttaa myös vähentämään velan syntymistä. Pitkään huomiotta jätetty tekninen velka ennen pitkää johtaa kehitystyön hidastumiseen, mobiilisovelluksen haurastumiseen ja jopa sen käytöstä poistumiseen, joten teknisen velan hallinta on olennainen osa pitkäjänteistä mobiilisovelluskehitystä.

Versionhallinnan tehokas käyttö ja ylläpidon haasteiden ennakointi ovat ratkaisevassa roolissa mobiilisovelluksen pitkän aikavälin menestykselle. Ylläpidon haasteiden hallinta auttaa paitsi varmistamaan teknisen vakauden, myös vastaamaan käyttäjien kysyntään ja markkinoiden asettamiin vaatimuksiin.

6.2 Käyttäjien sitouttaminen ja palautteen hyödyntäminen

Käyttäjien sitouttaminen mobiilisovellukseen on erittäin olennaista, koska se vaikuttaa suoraan sovelluksen menestykseen ja elinkaareen. Sitoutuneet käyttäjät eivät ainoastaan käytä mobiilisovellusta aktiivisesti, vaan he myös suosittelevat sitä muille, lisäävät asiakasuskollisuutta ja voivat tuoda merkittävää taloudellista hyötyä, esimerkiksi toistuvien ostojen muodossa. Lisäksi käyttäjien sitoutuminen vähentää käyttäjien vaihtuvuutta (engl. *churn*), mikä on ensiarvoista, sillä uusien käyttäjien hankkiminen on yleensä kalliimpaa ja työläämpää kuin olemassa olevien käyttäjien säilyttäminen.

Heinig kirjoittaa blogissaan (2024), että käyttäjien sitouttaminen edellyttää hyvin suunniteltua mobiilisovellusta, joka yhdistää toimivuuden, personoinnin ja käyttäjien tarpeisiin vastaamisen. Ensivaikutelma on ensisijaisen tärkeässä roolissa, sillä selkeä käyttöliittymä ja helposti ymmärrettävä käyttöönopastus auttavat käyttäjiä löytämään nopeasti sovelluksen arvon.

Personointi on käyttäjien sitoutumista lisäävä tekijä. Kun sovellus tarjoaa yksilöllisiä suosituksia tai mukautettuja ominaisuuksia käyttäjän toiminnan perusteella, se luo tunteen että sovellus on suunniteltu juuri käyttäjää varten. Push-ilmoitukset voivat muistuttaa käyttäjiä sovelluksen olemassaolosta ja houkutella heitä takaisin, kunhan ilmoitukset ovat harkittuja ja aidosti relevantteja, sillä liiallinen ilmoitusten määrä voi helposti myös karkoittaa käyttäjät.

Käyttäjien sitoutumista voidaan vahvistaa erilaisilla palkitsemisjärjestelmillä, kuten pisteillä, saavutuksilla tai alennuksilla, jotka motivoivat käyttäjiä pysymään aktiivisina. Sosiaaliset ominaisuudet, kuten mahdollisuus vuorovaikutukseen muiden käyttäjien kanssa tai osallistuminen kilpailuihin, voivat luoda yhteisöllisyyttä, mikä sitoo käyttäjiä entistä vahvemmin sovellukseen.

Käyttäjäpalautteen kerääminen ja hyödyntäminen on keskeinen osa käyttäjien sitouttamista, sillä se osoittaa käyttäjille, että heidän mielipiteensä ovat arvokkaita. Käyttäjäpalautteen hyödyntäminen jatkokehityksessä vaatii avoimuutta ja vuorovaikutusta. Käyttäjille on hyvä viestiä, kuinka heidän antamansa palaute on vaikuttanut sovelluksen kehitykseen, esimerkiksi esittelemällä uusia ominaisuuksia tai parannuksia, jotka on tehty heidän ehdotustensa perusteella. Viestintä käyttäjien suuntaan luo luottamusta ja vahvistaa käyttäjien sitoutumista sovellukseen entisestään.

7 POHDINTA

Opinnäytetyön tavoitteena oli luoda suunnitelma, miten web-sovellus saadaan muutettua laadukkaaksi ja käyttäjäystävälliseksi mobiilisovellukseksi. Työssä esiteltiin mobiilisovelluskehityksen peruselementtejä ja teknologioita. Aiheena mobiilisovelluskehitys oli todella laaja, vaikka työ olikin rajattu mobiilisovelluksen suunnitteluun. Jokaisen kappaleen aihe-alue mahdollistaa jatkokehityksen ja aiheeseen pidemmälle syventymisen. Tämä opinnäytetyö oli nopea pintaraapaisu ja läpileikkaus mobiilisovelluskehitykseen, heijastaen mitä mobiilisovelluskehitys on viimeisen muutaman vuoden sisällä ollut.

Työn aikana ymmärsin, että tuotekehitys on monimutkainen prosessi ja se sisältää useita eri vaiheita ja kerroksia. Mobiilisovelluksen suunnittelu voi näyttää hyvinkin erilaiselta riippuen siitä, paljonko suunnitteluun ja eri prosesseihin on varattu aikaa, millaista tuotetta kehitetään ja kenelle, sekä millaisella porukalla.

Opinnäytetyötä varten luin lukuisia julkaisuja aiheesta. Valitsin työhöni eri tyyppisiä lähteitä, jotta työ pysyisi mahdollisimman monipuolisena ja luotettavana. Yllätyin positiivisesti siitä, että löysin odotettua enemmän opinnäytetöitä ja graduja mobiilisovelluskehitykseen ja mobiilisovelluksen suunnitteluun liittyen. Toki suurin osa mobiilisovelluskehitykseen liittyvästä tiedosta jaetaan erilaisten blogipostausten yhteydessä, sillä tieteellisempien tutkimusten tekeminen on huomattavasti hitaampaa. Mobiilisovelluskehityksestä kertovat julkaisut vanhenevat nopeasti, joten blogipostaukset saatetaan alalla nähdä tehokkaampana julkaisumuotona.

Yritin välttää lähteitä, kuten joidenkin yritysten kirjoittamia tai tilaamia artikkeleita, joissa mainostetaan tuotteita tai palveluita. Tämän vuoksi jätin esimerkiksi käyttäjäpalautteen keräämisestä kertovan kappaleen ilman virallisia lähteitä. Osa työn sisällöstä perustunee myös osittain omiin kokemuksiini ja näkemyksiini. Mielipiteisiin pohjautuvat julkaisut eivät kuitenkaan välttämättä ole huonoja tai epäluotettavia, mutta niitä kannattaa aina lukea kriittisesti ja täydentää tietoja tarvittaessa muista lähteistä.

Opinnäytetyön aikana syntyneen suunnitelman ja dokumentaation perusteella on helpompaa alkaa kehittämään Omaisnäkymä-mobiilisovelluksen ensimmäistä prototyyppiä Nursebuddylle. Seuraava askel on käydä suunnitelma ja löydetyt havainnot läpi muun kehitystiimin kanssa, ja tehdä päätös tarkemmista aikatauluista.

Mobiilisovelluskehittäjän ja -suunnittelijan on tärkeää seurata mobiililaitteiden ja -sovellusten uusimpia teknologiapäivityksiä ja alan trendejä. On vaikeaa sanoa, mihin suuntaan mobiililaitteet ja mobiilisovellukset tulevaisuudessa kehittyvät. Viimeisten parin vuoden aikana nousussa ovat olleet erilaiset tekoälyä hyödyntävät työkalut, lisätty todellisuus (engl. *augmented reality, AR*) sekä puettava teknologia (engl. *wearable technology*).

LÄHTEET

FiCom. 2024. Digilaitteiden käyttö. Verkkosivu. Viitattu 9.11.2024. https://ficom.fi/ict-ala/tietopankki/digilaitteet/digilaitteiden-kaytto/digilaitteiden-kaytto/#mihin-alypuhelimia-kaytetaan

Nursebuddy. 2024. Verkkosivu. Viitattu 9.11.2024 https://nursebuddy.co/fi/etusivu

Kansallinen senioriliitto ry. N.d. Perusasioita nettiajassa – Mikä se selain on? Verkkosivu. Viitattu 9.11.2024. https://www.senioriliitto.fi/jasenille/seniorit-nettiajassa/perusasioita-nettiajasta/perusasioita-nettiajassa-mika-se-selain-on/

Liini Agency. 2023. Mobiilisovellus vs verkkosivut – 10 syytä miksi mobiilisovellus on parempi vaihtoehto. Verkkosivu. Viitattu 9.11.2024. https://liini.agency/blogi/mobiilisovellus-vs-verkkosivut-10-syyt%C3%A4-miksi-mobiilisovellus-on-parempi-vaihtoehto

Goray, Sunila. 2023. History of Mobile Apps – The Past, Present and Future. Verkkosivu. Viitattu 20.11.2024. https://webandcrafts.com/blog/history-of-mobile-apps

Geeks for Geeks. 2024. What is a mobile operating system? Verkkosivu. Viitattu 9.11.2024. https://www.geeksforgeeks.org/what-is-a-mobile-operating-system/

Statista. 2024. Leading mobile operating systems in Finland as of April 2024, by market share. Verkkosivu. Viitattu 9.11.2024. https://www.statista.com/statistics/623924/most-popular-mobile-operating-systems-in-finland/

Global Times. 2024. China's first fully home-grown mobile operating system HarmonyOS NEXT is launched. Global Times 22.10.2024. Viitattu 9.11.2024. https://www.globaltimes.cn/page/202410/1321670.shtml

Galatioto, Chris. 2023. The Great Debate: Native vs. Non-Native Mobile Apps. Verkkosivu. Viitattu 20.11.2024. https://www.heady.io/blog/the-great-debate-native-vs.-non-native-mobile-apps

React Native. 2024. Communication between native and React Native. Verkkosivu. Viitattu 20.11.2024. https://reactnative.dev/docs/communication-ios

AlanB, Medium. 2024. Native vs Non-Native Mobile Apps – What's the Difference? Verkkosivu. Viitattu 20.11.2024. https://medium.com/swlh/native-vs-non-native-mobile-apps-whats-the-difference-b3a641e06f52

Kauriola, Meri. 2021. Saavutettavuus osana mobiilisovellusten suunnittelutyötä. Opinnäytetyö. Viitattu 20.11.2024. https://www.theseus.fi/bitstream/handle/10024/503519/Meri_Kauriola.pdf?sequence=2

Peris.ai – Cybersecurity. 2024. Why Security UX Matters More Than You Think. Verkkosivu. Viitattu 20.11.2024. https://www.linkedin.com/pulse/why-security-ux-matters-more-than-you-think-perisai-cybersecurity-jlnnc

Apple Support. 2024. About iOS 16 Updates. Verkkosivu. Viitattu 20.11.2024. https://support.apple.com/en-us/101566

Matheus Henrique de Souza Fontenele. 2024. Live Activity on Flutter Android – An alternative implementation. Viitattu 20.11.2024. https://medium.com/@matheusdeveloper.henrique/live-activity-on-flutter-android-an-alternative-implementation-1e16bec1fbd8

Geeks for Geeks. 2024. MoSCoW Prioritization Technique in Product Management. Verkkosivu. Viitattu 20.11.2024. https://www.geeksforgeeks.org/moscow-prioritization-in-product-management/

Martins, Julia. 2024. 7 common causes of scope creep, and how to avoid them. Verkkosivu. Viitattu 20.11.2024. https://asana.com/resources/what-is-scope-creep

Figma. N.d. Components, styles, and shared library best practices. Verkkosivu. Viitattu 20.11.2024. https://www.figma.com/best-practices/components-styles-and-shared-libraries/

React Native. N.d. Verkkosivu. Viitattu 20.11.2024. https://reactnative.dev

Flutter. N.d. Verkkosivu. Viitattu 20.11.2024. https://flutter.dev/

Expo. 2024. Introduction. Verkkosivu. Viitattu 25.11.2024. https://docs.expo.dev/get-started/introduction/

Expo. 2024. Development and production modes. Verkkosivu. Viitattu 25.11.2024. https://docs.expo.dev/workflow/development-mode/

Expo. 2024. Send over-the-air updates. Verkkosivu. Viitattu 25.11.2024. https://docs.expo.dev/deploy/send-over-the-air-updates/

Expo. 2024. Prebuild. Verkkosivu. Viitattu 25.11.2024. https://docs.expo.dev/workflow/prebuild/

Hunt, Andrew & Thomas, David. 1999. The Pragmatic Programmer. Yhdysvallat: Addison-Wesley. Viitattu 25.11.2024.

Flux. 2023. In-Depth Overview. Verkkosivu. Viitattu 25.11.2024. https://facebookarchive.github.io/flux/docs/in-depth-overview/

Geeks for Geeks. 2023. What are the differences between Redux and Flux in ReactJS? Verkkosivu. Viitattu 25.11.2024. https://www.geeksforgeeks.org/what-are-the-differences-between-redux-and-flux-in-reactjs/

Lusan Das. 2018. The best way to architect your Redux app. Verkkosivu. Viitattu 25.11.2024. https://www.freecodecamp.org/news/the-best-way-to-architect-your-redux-app-ad9bd16c8e2d/

Richardson, Chris. 2018. Microservices Patterns. Manning Publications. Viitattu 2.12.2024. https://www.manning.com/books/microservices-patterns

TechJury. 2024. 55+ Jaw Dropping App Usage Statistics in 2024 [Infographic]. Verkkosivu. Viitattu 12.11.2024 https://techjury.net/blog/app-usage-statistics/

Apple Developer. 2024. Maximum build file sizes.Verkkosivu. Viitattu 2.12.2024. https://developer.apple.com/help/app-store-connect/reference/maximum-build-file-sizes

Play Console Help. N.d. Create and set up your app. Manage your app and app bundles. Verkkosivu. Viitattu 2.12.2024. https://support.google.com/googleplay/android-developer/answer/9859152?hl=en#zippy=%2Cmaximum-size-limit

Datarockets. 2023. The Magic of App Size Reduction. Verkkosivu. Viitattu 2.12.2024. https://datarockets.medium.com/the-magic-of-app-size-reduction-bc4c1f9bad0a

React Native. 2024. Performance Overview. Verkkosivu. Viitattu 2.12.2024. https://reactnative.dev/docs/performance

Sinha, Sumit. 2022. Stop using MomentJS. Verkkosivu. Viitattu 2.12.2024. https://sumitaec108.medium.com/stop-using-momentjs-f71872f01716

Rahman, Anisur. 2024. React Native – Ultimate Guide on Performance Optimization. Verkkosivu. Viitattu 2.12.2024. https://github.com/anisurrahman072/React-Native-Advanced-Guide/blob/master/Performance-Optimization/Performance-Optimization-coding-guide.md

Flipper. 2024. Introduction. Verkkosivu. Viitattu 2.12.2024 https://fbflipper.com/docs/features/

Firebase. 2024. Firebase Performance Monitoring. Verkkosivu. Viitattu 2.12.2024. https://firebase.google.com/docs/perf-mon

GDPR EU. 2024. What is GDPR, the EU's new data protection law? Verkkosivu. Viitattu 2.12.2024. https://gdpr.eu/what-is-gdpr/

Sharma, Rajnish Kumar. 2024. How to Protect Data in Mobile and Web Apps Using Encryption. Verkkosivu. Viitattu 2.12.2024. https://www.netsolutions.com/hub/mobile-app-development/data-encryption

React Native. 2024. Testing. Verkkosivu. Viitattu 2.12.2024. https://reactnative.dev/docs/testing-overview

Sorell, Sami. 2024. Over-the-Air update and should you have it? Verkkosivu. Viitattu 3.12.2024. https://we-do.vincit.com/over-the-air-update-and-should-you-have-it

Testscenario. 2024. Why is Mobile App Testing on Real Devices Important? Verkkosivu. Viitattu 3.12.2024. https://www.linkedin.com/pulse/why-mobile-app-testing-real-devices-important-testscenario-r1evf

Heinig, Ian. 2024. 13 proven strategies to increase app engagement. Verkkosivu. Viitattu 3.12.2024. https://sendbird.com/blog/increase-app-engagement-with-these-strategies