Jere Vänttinen

Sovellusohjelman kehittäminen autologistiikan toimintaan

Biila Solutions Oy

Energia- ja
informaatiotekniikka
Kandidaatintutkielma
Automaatio ja tietotekniikka

Vaasa 2023

VAASAN YLIOPISTO
Energia- ja informaatiotekniikka
Tekijä: Jere Vänttinen
Tutkielman nimi: Sovellusohjelman kehittäminen autologistiikan toimintaan : Biila Solutions Oy
Tutkinto: Tekniikan kandidaatti
Oppiaine: Automaatio ja tietotekniikka
Työn ohjaaja: Jouni Lampinen
Valmistumisvuosi: 2023 **Sivumäärä:** 46

TIIVISTELMÄ:
Tässä tutkielmassa tarkastellaan nykyajan eri sovelluskehityksen vaihtoehtoja ja niiden toimivuutta eri näkökulmista huomioiden toimeksiantajan tarpeet. Tutkielman tavoitteena on etsiä Biila Solutions Oy -yritykselle heidän kannaltaan paras sovellusratkaisu yrityksen työntekijöinä toimivien autonkuljettajien tarpeisiin. Tutkielma on näin ollen rajattu yrityksen toimintaa koskevaksi. Tutkielma on laajempi kirjallisuuskatsaus, joka sisältää myös empiirisen osan pohtien sovelluksen kehitystä ja sen toimivuutta. Tutkielman tarkoitus on vertailla vaihtoehtoja yrityksen nykyisen web-sovelluksen tilalle.

Työ aloitetaan tutustumalla teoriapohjaan ja keräämällä tarvittava käyttäjäpalaute. Tämän jälkeen pohditaan nykyisen sovelluksen toimintaa ja sen ominaisuuksia. Tästä loogisesti edeten ryhdytään pohtimaan järkevää vaihtoehtoa nykyisen sovelluksen tilalle. Lopuksi luodaan perusteltuja johtopäätöksiä työstä ja sen tuloksista. Tämän työn työmenetelminä käytetään hankittua teoriapohjaa ja käyttäjien antamaa palautetta sekä ideoita. Palaute kerätään Google Forms -kyselyn muodossa. Käyttäjien antama palaute ja ideat huomioiden pohditaan tulevan sovelluksen toimintaa ja toteutusta sekä tarjotaan omia ideoita sovelluksen kehittäjälle. Työn lopputuloksena syntyy selvitys mahdollisesta tulevasta sovelluksesta.

Työn ja sen tuloksien perusteella natiivisovellusta voidaan pitää Biilan kuljettajien käyttöön optimaalisimpana sovellustyyppinä. Natiivisovellus tarjoaa nopeat latausajat sekä laajan skaalan erilaisia mahdollisuuksia uusille toiminnoille sekä olemassa olevien toimintojen parantamiseen. Tärkeänä päätelmänä voidaan myös todeta, että sovellustyypin valinta ei ole yksiselitteinen, vaan sitä pohtiessa täytyy priorisoida erilaisia asioita ja katsoa valintaa monesta näkökulmasta. Valinnan on hyvä kohdistua siihen sovellustyyppiin, joka on lähinnä toimeksiantajan vaatimuksia järkevä ja kustannustehokas toteutus huomioiden. Tämän työn perusteella suosittelen työn toimeksiantajalle natiivisovelluksen kehittämistä.

AVAINSANAT: Sovelluskehitys, Natiivisovellus, Hybridisovellus, Progressiivinen web-sovellus, Käyttökokemus

Sisällys

Kuviot

Taulukot

Lyhenteet

PWA – Progressive web application eli Progressiivinen web-sovellus

API – Application programming interface eli ohjelmointirajapinta

JSON – JavaScript Object Notation, JavaScriptiin liittyvä tiedostomuoto

1 Johdanto

Mobiilisovellusten määrä on valtava nykypäivänä ja niitä käytetään päivittäin arkisten asioiden helpottamiseksi. Mansoor (2023) arvioi, että maailmassa on tällä hetkellä 7 miljoonaa IOS- ja Android-alustalla toimivaa mobiilisovellusta. Applen App Store julkistettiin vuonna 2008 ja sovelluksia App Storessa oli tällöin 500 (Mansoor, 2023). Jo tämä itsessään kuvastaa alan valtavaa kasvua pelkästään kuluneen 15 vuoden aikana. Sovelluskehityksen ala käykin tällä hetkellä erittäin kuumana, minkä takia aihe on tärkeä lähes jokaiselle toimintaansa kasvattavalle yritykselle.

Sovelluskehityksessä sovellustyypin valintaan yleisesti vaikuttavia tekijöitä ovat esimerkiksi resurssit, käyttäjäkokemus, saatavuus, jakelu sekä ylläpito (Riikola, 2022). Näitä tekijöitä pohdittaessa jokainen sovelluskehitystä pohtiva taho joutuu punnitsemaan kyseiset tekijät omalta kannaltaan. Tämä on kuitenkin vain yleistä pohdintaa, joten todelliset päätökset tulee tehdä tapauskohtaisesti. Tämän työn tavoitteena on pohtia sovellustyypin valintaa ja sen kehitystä Biila Solutions Oy -yritykselle heidän tarpeensa huomioiden.

Biila Solutions Oy on tämän hetken johtava autologistiikka-alan teknologiayhtiö. Biila suorittaa kuljettajiensa toimesta asiakkaan tilaaman autonsiirron paikasta A paikkaan B. Biilan (2023) nettisivujen mukaan autonsiirtoja tapahtuukin päivässä yli 200 ympäri Suomea. Oman mottonsa mukaan Biila ”kokoaa yhteen autonsiirron tarvitsijat sekä siirron suorittavat kuljettajat”. Jo pelkästään maaliskuuhun mennessä vuonna 2023 siirtoja on suoritettu liki 18 000 kappaletta. Tämä kuvastaa alan laajuutta tällä hetkellä hyvin, ja Biilan kumppaneihin kuuluukin useita Suomen johtavia autoalan toimijoita. Näin ollen kovassa kasvussa olevan yhtiön tuleekin laajentaa toimintojaan, esimerkiksi nykypäivänä jopa arkisen sovellusteknologian osalta.

Tässä tutkielmassa tutkimusmenetelminä käytetään jo hallittavan ja hankitun teoriapohjan soveltamista sekä käyttäjäpalautetta sovelluksen toimintaan liittyen. Ensin tutustutaan tiivistetysti sovelluskehityksen vaihtoehtoihin teoriaosuudessa, minkä

jälkeen voidaan alkaa pohtimaan Biilalle sopivaa ratkaisua. Käyttäjäpalaute kerätään Biilan alaisuudessa toimivilta kuljettajilta Google Forms -kyselyn muodossa. Käyttäjät voivat myös jättää kyselyyn omia parannusehdotuksia ja ideoitaan sovelluksen toimintaan liittyen. Tämä käyttäjäpalaute ja teoriapohja yhdistämällä luodaan juuri Biilalle sopivin ratkaisu sovelluskehitykseen liittyen.

Lähtökohtaisesti sovellus on tarkoitus kehittää yrityksen autonkuljettajien käyttöön, joten kuljettajien tarpeet otetaan huomioon sovelluksen kehitystä miettiessä. Työ aloitetaan pohtimalla nykyisiä järkeviä sovellusvaihtoehtoja kategorioiden ne alustariippumattomiin ja alustariippuvaisiin sovelluksiin. Tämän jälkeen tarkastellaan jo nykyisen olemassa olevan web-sovelluksen tilaa ja toimintaa. Täten mietitään järkevintä sovellustyyppiä juuri Biilalle. Tutkielman hypoteesi on, että todennäköisesti järkevin ratkaisu on kehittää natiivisovellus. Lähtökohtainen oletus on myös, että natiivisovellus tuo merkittävää etua sovelluksen suorituskykyyn ja toimintoihin nähden. Tutkielman lopuksi pohditaan, miten valittu sovellustyyppi kehitetään ja kerätään kehitystä varten käyttäjäpalautetta ja ideoita. Tutkielma viedään päätökseen johtopäätökset-kappaleessa, jossa luodaan tutkielman pohjalta perusteltuja väitteitä.

.

2 Sovelluskehitys

Sovelluskehityksellä viitataan esimerkiksi tietokoneella tai älylaitteella toimivan sovelluksen kehittämiseen. Sovelluskehityksen päämääränä on kehittää sovelluksia haluttuihin käyttötarkoituksiin ja näin ollen sovellustyypin valinta on olennainen osa prosessia. Tämä luku käsittelee nykypäivän järkevimmät vaihtoehdot sovellustyypille. Luku on jaettu alustariippumattomiin ja alustariippuvaisiin sovellustyyppeihin.

2.1 Alustariippumattomat sovellustyypit

Alustariippumattomien sovellusten perusideana on, että yhdestä koodikannasta voidaan toteuttaa useilla eri alustoilla toimiva sovellus. Tämä on tehokas tapa optimoida sovelluksen kehitykseen ja ylläpitämiseen käytettäviä resursseja. Lähtökohtainen ajatus on, että alustariippumattomat sovellukset häviävät natiivisovelluksille suorituskyvyssä ja tarjoamissaan toiminnoissa, mutta niiden laaja yhteensopivuus sekä saavutettavuus tarjoavat omanlaisiaan etuja sovelluksen tarjoajalle. Riikolan (2022) mukaan alustariippumattomien sovellusten kehityssuunnat ovat vielä kesken, joten parhaan teknologian saavuttaminen voi viedä vielä vuosia. Alustariippumattomien sovellusten etuna on myös niiden päivitettävyys. Ne voidaan päivittää keskitetysti, jolloin työmäärä on huomattavasti pienempi natiivipäivityksiin verrattuna. Tämä luku esittelee nykypäivän yleisimmät alustariippumattomat sovellustyypit: progressiivinen web-sovellus (PWA, Progressive web application) ja hybridisovellus.

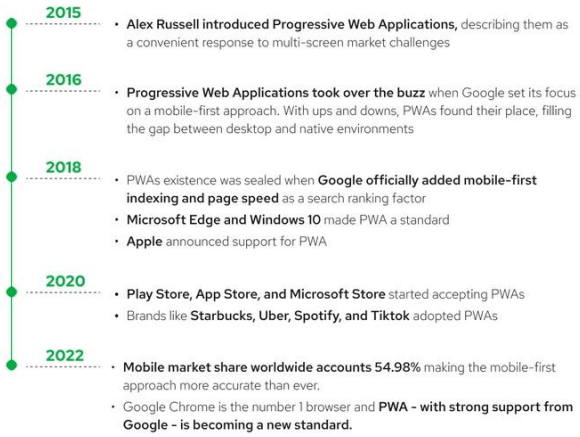
2.1.1 Progressiivinen web-sovellus (PWA)

Progressiivinen web-sovellus on selainpohjalla toimiva sovellus, joka toimii moninaisesti eri selainten ja niiden versioiden ominaisuudet huomioiden. PWA rakentuu modernien webteknologioiden, kuten HTML-, CSS- ja JavaScriptin varaan, hyödyntäen myös ohjelmointirajapintoja eli API-teknologiaa. Yksi tärkeä PWA:n etu on, että PWA tukee laitteen omia toimintoja, jolloin esimerkiksi kameran käyttö sekä hetkellisen sijainnin hakeminen onnistuvat (Riikola, 2022). Progressiivinen web-sovellus on ikään kuin

normaalia web-sovellusta läheisempi sukulainen natiivisovellukselle käyttötuntumansa ja ominaisuuksiensa puolesta.

2.1.1.1 Historiaa

Hietalan (2021) mukaan käsite ”progressiivinen web-sovellus” esiteltiin vuonna 2015 Google Chromen kehittäjä Alex Russellin blogijulkaisussa asettaen lähtöpisteen tuleville PWA:n kehityssuunnille. Russell (2015) totesikin blogikirjoituksessaan näppärän kielikuvan avulla: ”Nämä sovellukset ovat vain verkkosivustoja, jotka söivät juuri oikeita vitamiineja”. Tällä Russell viittaa PWA:n tarjoamiin uusiin mahdollisuuksiin uusia teknologioita hyödyntäen. Alkuperäinen PWA:n idea olikin täyttää tavallisten web-sovellusten ja natiivisovellusten väliin jäävä tila yhdistäen niiden toimintaperiaatteita sekä ominaisuuksia. PWA-teknologia nopeutti sivun toimintaa perinteiseen web-sovellukseen verrattuna tarjoten myös käyttömahdollisuuksia, vaikka laite ei olisi kytketty verkkoon. PWA:n suosio onkin ollut jatkuvassa kasvussa sen aikaisempaa edistyneempien ominaisuuksien ansiosta. Tähän päivään mennessä PWA-teknologiaa tukevat lähes kaikki eri selaimet tehden teknologian hyödyntämisestä kannattavaa monille palveluntarjoajille. Oheisessa kuvassa (kuvio 1) kuvattuna PWA:n kehitysvaiheita.



Kuvio 1. Progressiivisen web-sovelluksen kehitysvaiheet (Vue Storefront, 2022, ENG).

2.1.1.2 Toteutus

Niin kuin jo kappaleen johdannossa todettiinkin, PWA:n kehityksessä käytetään pääasiassa HTML, CSS ja JavaScript -ohjelmointikieliä. Sovelluksen runko latautuu käyttäjälaitteen muistiin ja rungon sisältö ladataan dynaamisesti verkosta sovellusta käytettäessä. PWA hyödyntää myös ohjelmointirajapintoja eli API-teknologiaa. Näin ollen saadaan erilliset järjestelmät ”keskustelemaan keskenään”. Mobiiliyhteensopivina verkkosivuina toteutetut PWA:t toimivat kolmen perusvaatimuksen alla: (Hietala, 2021)

1. Sovellusta pyörittävän palvelimen täytyy tukea HTTPS-protokollaa.
2. Käytössä täytyy olla *Service Worker*, joka vastaa sovelluksen offline-käytöstä.
3. Sovelluksella tulee olla JSON-tiedosto, joka määrittelee sovelluksen perusinformaation (metadata).

Kaikki edellä mainitut kohdat tulee toteuttaa, jotta sovellusta voidaan kutsua progressiiviseksi web-sovellukseksi. HTTPS-protokollaa tarvitaan turvallisen yhteyden luomiseen sovelluksen ja palvelimen välille luottamuksellisen tiedon salaamiseksi. Niin sanottu *Service Worker* mahdollistaa sovelluksen käytön ilman internetyhteyttä. *Service workerin* tarjoama sivu voikin olla itsessään vain sivu, mikä ilmoittaa verkkoyhteyden olevan poikki. *Service Worker* on toteutettu JavaScriptillä ja se myös mahdollistaa push-ilmoitusten lähettämisen osalla käyttöjärjestelmistä. 3. kohdassa mainittu JSON-tiedosto sisältää sovelluksen nimen, kuvakkeen sekä käynnistysosoitteen. JSON-tiedosto kertoo selaimelle sovelluksen ilmenemisestä sekä miten sovellus käynnistetään. JSON-tiedosto sisältää toiminnallisuuden myös yhteen PWA:n oleellisimmista ominaisuuksista: selainpohjalla toimivan sovelluksen pystyy asentamaan esimerkiksi mobiililaitteen kotivalikkoon (Hietala, 2022).

2.1.1.3 Käyttäjän näkökulma

Hietala (2022) toteaa, että käyttäjän kannalta yksi oleellisimmista ominaisuuksista on se, että PWA:t ladataan verkosta sovelluskaupan tai muun alustan sijaan. Tästä huolimatta käyttäjä saa kuitenkin natiivisovelluksen kaltaisen käyttökokemuksen alustasta riippumatta. Käyttäjälle PWA tarjoaa myös perinteistä web-sovellusta sujuvamman ja vakaamman käyttökokemuksen tinkimättä sovelluksen laadusta tai toiminnoista. Käyttäjän kannalta oleellisimpia sovelluksen piirteitä on sen saavutettavuus, sillä sovellukseen pääsee käsiksi mobiilikäyttöjärjestelmästä riippumatta myös tietokoneella. Jo pelkkä sovelluksen latausprosessi karsii osan käyttäjistä natiivisovelluksen tapauksessa, sovellusten päivittämisestä puhumattakaan. PWA päivittyy ilman erillisiä alustakohtaisia päivityksiä, joten päivittämiseen käyttäjän toimia ei vaadita.

2.1.1.4 Kehittäjän näkökulma

Myös kehittäjän näkökulmasta PWA on optimaalinen tarjoten helposti yhteensopivan perusrakenteen, laitteesta ja käyttöjärjestelmästä riippumatta. PWA:n kehittämiseen käytetään aiemmin tässä luvussa mainittuja ohjelmointikieliä ja -työkaluja, mitkä ovat tuttuja suurimmalle osalle sovelluskehittäjistä ja ohjelmoijista. Jo tämä itsessään nopeuttaa kehittäjän työprosessia, koska kehittäjän ei tarvitse opetella uutta sovellusrakennetta tai ohjelmointikieltä. PWA:n ylläpitäminen ja päivittäminen on myös kehittäjälle vaivattomampaa, kuin esimerkiksi natiivisovelluksen ylläpito. Sovellus voidaan päivittää katkaisematta sen toimintaa nykyhetkessä aiemmin tässä luvussa mainitun Service Workerin avulla. Toisaalta sovelluksen poissaolo sovelluskaupasta voi huonontaa sen näkyvyyttä joissain määrin, jolloin kehittäjälle kannattavampaa olisi natiivisovelluksen luominen (Kaikkonen 2021).

2.1.2 Hybridisovellus

Tässä luvun osassa tutustutaan hybridisovellukseen. Hybridisovellus toimii PWA:n tavoin selaimella, mutta erona on se, että sen voi ladata kunkin käyttöjärjestelmän omaan sovelluskauppaan. Kaarakainen (2021) toteaakin hybridisovellusten käyttävän samoja web-teknologioita kuin web-sovellukset, mutta sovelluksen suoritus tapahtuu kullekin alustalle natiivissa ohjelmistosäiliössä. Nimi hybridisovellus tuleekin juuri natiivin sovellusosan ja web-osan yhdistelmästä.

2.1.2.1 Historiaa

Hybridisovellukset saivat alkunsa web-teknologioiden kehittyessä HTML5:n julkaisun myötä. Sidana (2015) painottaakin, että web-kehittäjien työkalupakki kasvoi valtavasti HTML5:n julkistamisen myötä. Se mikä oli ennen mahdotonta, oli nyt mahdollista. JavaScriptin ja jQueryn avulla luodut verkkosivut näyttivät entistä kauniimmilta. Angularin ja Ionicin julkistamisen myötä mobiilisovellusten hybridikehittäminen nosti päätään. Hybridisovellukset ovatkin nykypäivänä toimiva ja edullinen ratkaisu mobiilisovellusten kehittämiseen.

2.1.2.2 Toteutus

Perusrakenteeltaan web-osan toteutuksessa käytetään JavaScript-, HTML5- ja CSS-ohjelmointia samoin kuin PWA-toteutuksessa, mutta hybridisovelluksessa tämä sovelluksen ydin kapseloidaan natiivin sovelluksen sisälle. Valmista sovellusta ajetaan siihen itseensä sulautetun natiivin verkkoselaimen sisällä (Riikola, 2021). Esimerkiksi iOSalustalle toteutettu hybridisovellus käyttää WKWebView-työkalua sovelluksen esittämiseen käyttäjälle, kun taas Anroidilla vastaavaan toimintoon käytetään WebViewtyökalua. Apache Cordovalla tai vastaavalla työkalulla luodaan natiivi kapselointi, jonka avulla sovellus saadaan sovelluskaupalle kelvolliseen rakenteeseen. Cordovan pluginlisätoiminnolla sovelluksen toimintaa voidaan laajentaa laitteen ominaisuuksien tasolle, jolloin sovellus voi käyttää esimerkiksi kameraa tai Applella FaceID-toimintoa (Griffith, 2023).

2.1.2.3 Käyttäjän näkökulma

Käyttäjälle hybridisovellus tarjoaa alustavapaan sovellusratkaisun, josta löytyvät samat ominaisuudet ja sama rakenne alustasta riippumatta. Sovellus on näin ollen hyvin saavutettavissa, mutta toisaalta se täytyy ladata kullekin laitteelle laitteen omasta sovelluskaupasta. Tämä itsessään jättää jo osan käyttäjistä saavuttamatta. Hybridisovellusten huonona puolena ovat suorituskykyyn ja toimivuuteen liittyvät ongelmat, jotka heikentävät sovelluksen käyttökokemusta.

2.1.2.4 Kehittäjän näkökulma

Kehittäjän näkökulmasta hybridisovellukset ovat työläämpi ratkaisu kuin esimerkiksi PWA, koska jokaiselle alustalle täytyy ohjelmoida alustakohtaiset komponentit. Näin ollen kehittämiseen kuluu enemmän resursseja, kuten aikaa ja rahaa. Toisaalta natiivisovelluksiin verrattuna hybridisovellukset ovat parempi ratkaisu resurssien optimoimisen kannalta, sillä hybridisovelluksissa osa koodikannasta on kaikilla alustoilla yhteistä ja näin ollen koko sovellusta ei tarvitse ohjelmoida jokaiselle alustalle erikseen. Hybridisovellus perustuu myös web-teknologioihin, joten sovelluksen tietoturvaan voi liittyä riskejä, sillä yleensä web-teknologiat ovat haavoittuvaisempia, kuin natiivit teknologiat. Hybridisovellus on kehittäjän kannalta kuitenkin hyvä ratkaisu, jos halutaan saavuttaa enemmän käyttäjiä pienemmillä resursseille suhteessa natiiveihin sovelluksiin (Riikola, 2021).

2.2 Alustariippuvainen natiivisovellus

Alustakohtaisella natiivisovelluksella tarkoitetaan tietylle alustalle kehitettyä sovellusta alustan tarjoamia kehitystyökaluja käyttäen. Natiivisovellus on yleensä käyttäjäystävällinen vaihtoehto, joskin kehittäjälle se on monesti työläämpi sovellustyyppi. Nykypäivänä yleisimmät kehitysalustat ovat joko Android- tai iOS-

pohjaisia. Molemmille käyttöjärjestelmillä on oma kehitysympäristönsä, jossa sovellukselle ja alustalle natiivi ohjelmointi toteutetaan alustan tarjoamia ohjelmointikirjastoja hyödyntäen (Kaikkonen, 2021). Alla oleva kuva (kuvio 2) esittää natiivin sovelluskehityksen hyötyjä tiivistetysti.

Kuvio 2. Natiivisovelluksen tärkeimmät hyödyt (Medium, 2020, ENG).

2.2.1 Historiaa

Jo vuonna 1983 Applen silloinen toimitusjohtaja ja perustaja Steve Jobs kaavaili App Storen ensimmäistä versiota. Myös Nokian vuonna 1997 julkaisemaa Snake-puhelinpeliä voidaan pitää natiivin sovelluksen esiversiona. Kesti kuitenkin vielä yli kymmenen vuotta Nokian Snake-pelin julkistamisesta, ennen kuin Applen App Store julkistettiin vuonna 2008. Silloinen App Store oli luotu toimimaan nimenomaan vuonna 2007 julkaistun ensimmäisen iPhonen natiivina sovelluskauppana. Applen App Storea seurasivat uusien älypuhelinmallien julkistamisen myötä esimerkiksi Google Play, Amazon App Store ja Blackberry App World -sovellusalustat kasvattaen sovellustarjontaa entisestään. Vuonna 2009 julkistettu Angry Birds -mobiilipeli kasvatti mobiilipelien sekä sovellusten suosiota valtavasti ja avasi uusia ovia sovelluskehityksen maailmassa.

Applen tilastojen mukaan jo pelkkä iPhonen julkistaminen loi Yhdysvaltoihin 300 000 uutta työpaikkaa (Inventionland, 2023). Inventionlandin (2023) blogissa todetaankin, että mobiilisovellukset ovat tulleet tänne jäädäkseen. Myös Hietala (2021) painottaa natiivisovellusten valtavaa suosiota julkistamisen yhteydessä, sillä jo pelkästään ensimmäisenä viikonloppuna App Storen julkistamisen jälkeen sen natiivisovelluksia ladattiin yli 10 miljoonaa kertaa.

2.2.2 Toteutus

Kaikkosen (2021) mukaan natiivisovelluksen keskeisimpiä ominaisuuksia on se, että sovellus tulee kehittää jokaiselle alustalle sen natiivilla kehitysympäristöllä sekä -työkaluilla. Kaksi yleisintä nykypäivänä käytettävää mobiilialustaa ovat iOS ja Android. Kummallekin on olemassa oma kehitysympäristö; Androidilla Android Studio ja Applella Xcode. Androidilla käytetään ohjelmointikielinä Javaa ja Kotlinia, kun taas Applella käytetään Objective-C- ja Swift-kieliä. Näiden lisäksi nykyään käytetään Xamarin Native ja React Native ohjelmointikehyksiä ja -kirjastoja. Alla olevassa kuvassa (kuvio 3) on esitetty alustakohtaiset kehitysympäristöt sekä yleisimmin käytetyt ohjelmointikielet.

Kuvio 3. Natiivisovelluksen alustakohtaiset kehitysympäristöt ja ohjelmointikielet (9Expert, 2023).

Hietala (2021) toteaa, että Swift on jatkuvan kehityksen kohde, johon julkaistaan usein uusia päivityksiä. Rajoittavaksi tekijäksi Applen Xcodessa muodostuu se, että koodia voi allekirjoittaa ja kääntää vain Applen MacOS-käyttöjärjestelmällä, eli kehittäjän täytyy omistaa Mac-tietokone kääntääkseen koodin. Koodin kirjoittaminen luonnollisesti onnistuu muissakin ympäristöissä. Lisäksi Xcode sisältää iOS-sovelluskehitykseen hyödyllisiä virheenkorjaukseen ja suorituskyvyn analysointiin tarkoitettuja työkaluja.

Xcodella voi mobiilisovellusten lisäksi kehittää myös Java-appletteja, Mac-sovelluksia sekä komentoriviohjelmia. Xcodessa on emuloitu virtuaalilaite, jolla voi testata kirjoittamaansa koodia käytännön sovelluksena. Sovelluksen julkaisemin App Storeen ei kuitenkaan ole niin yksinkertainen prosessi kuin esimerkiksi Androidilla. Sovelluksen julkaiseminen vaatii Apple Developer -jäsenyyden, joka kustantaa 99 Yhdysvaltain dollaria vuodessa eli nykykurssilla (2023) noin 90 euroa vuodessa. Apple pyrkii tällä ja muilla seulontatoimenpiteillään pitämään sovelluskauppansa laadukkaana ja yhteensopivana Applen laitteille. Apple Developer -jäsen pääsee liittymisen jälkeen käyttämään Apple Store Connect -nettisivua, josta kehittäjä hallintasovelluksena kehittäjän ja App Storen välillä. Jotta sovelluksen voi julkaista App Storeen, sovellus ei saa sisältää bugeja ja sen täytyy olla stabiili eli se ei kaadu. Sovelluksen tulee olla myös turvallinen, ettei se vahingoita käyttäjien laitteita. Lisäksi laiton ja haitallinen materiaali täytyy sulkea kokonaan pois, jotta sovelluksen saa Applen seulan läpi. Sovellus täytyy testata edellä mainittujen bugien varalta sekä kehitysvaiheessa tarvittu turha koodi tulee poistaa ja backend-ohjelmoinnin toimivuus testata. Kutene Androidilla, myös Applella tarvitaan kuvia sovelluksen toiminnasta ja käyttöliittymästä, mutta lisäksi Apple vaatii vielä demokäyttäjän, joka testaa sovelluksen toiminnan. Sovellus palautetaan herkästi takaisin kehittäjälle korjattavaksi, mikäli se ei vastaa Applen vaatimuksia. Apple vaatii myös dokumentaation kehittäjän bisnesmallista, mikäli sovelluksen hankkimiseen tai sen toimintaan liittyy maksutoimenpiteitä (Kaikkonen, 2021).

Android studio julkaistiin vuonna 2013 ja se korvasikin jo vuonna 2014 alun perin Android-sovelluskehityksessä käytetyn Eclipse-ohjelman kehitystyökaluna. Android studio sisältää itsessään mobiililaitteiden emulaattorin, joka mahdollistaa sovellusten testaamisen virtuaalisesti kehitysympäristön sisällä. Näin ollen kehittäjä voi helposti testata jo ohjelmoimaansa sovellusta sen kehitysvaiheessa. Sovellusten julkaiseminen Google Play -sovelluskauppaan vaatii Googlen kehittäjätunnuksen sekä kolme muuta vaatimusta: sovelluksen otsikko, lyhyt ja pitkä kuvaus sovelluksesta. Sovelluksella on hyvä myös valita kategoria, sovelluskuvake sekä muutama kuva sovelluksen toiminnasta. Lopuksi on vielä hyvä jättää yhteystiedot käyttäjien palautetta sekä virheilmoituksia varten. Sovellus itsessään ladataan sovelluskauppaan luomalla Android Studion -projektista sovelluspakettitiedosto eli APK-tiedosto, joka mahdollistaa sovelluksen asentamisen Android-käyttöjärjestelmään. Lopuksi tehdään ”rollout”, joka julkaisee sovelluksen saataville haluttuun kohdemaahan (Kaikkonen, 2021).

2.2.3 Käyttäjän näkökulma

Natiivisovellukset ovat hyvin käyttäjäystävällisiä niiden sujuvan toiminnan ja käyttöjärjestelmälle luodun rakenteen ansiosta. Natiivisovellus kykenee ottamaan kaiken hyödyn irti käyttäjän laitteesta, joten lähes kaikkia laitteen fyysisiä toimintoja pystytään hyödyntämään ohjelmallisesti. Natiivisovellus voi siis esimerkiksi hyödyntää laitteen kameraa tai sijaintia helpottaakseen sovelluksen käyttöä. Käyttäjän näkökulmasta myös sovelluksen kyky toimia ilman verkkoyhteyttä parantaa käyttökokemusta. Natiivisovelluksen yhtenä huonona puolena on sen yhteensopivuus. Käyttäjä tarvitsee tietyn käyttöjärjestelmän omaavan laitteen, mikäli haluaa käyttää sovellusta. Kehittäjä saattaa myös julkaista sovelluksen vain jollekin tietylle alustalle, jolloin osa käyttäjistä jää sovelluksen saavuttamattomiin. Tämä itsessään karkottaa osan käyttäjistä, joten saavuttavuus heikkenee. Toisaalta esimerkiksi Applen brändäämä ”Download from the App Store” tai Googlen vastaava ”Available on the Google play” kasvattavat saatavuutta, koska ihmiset tietävät tällöin, mistä ja miten sovellus on saatavilla. Tällöin muun muassa mainonnalla voidaan tavoittaa entistä enemmän brändin tuntevia käyttäjiä. Hyvin

toteutettuna natiivisovellus tarjoaa erittäin miellyttävän ja luotettavan käyttäjäkokemuksen unohtamatta brändättyä sovellustuotetta.

2.2.4 Kehittäjän näkökulma

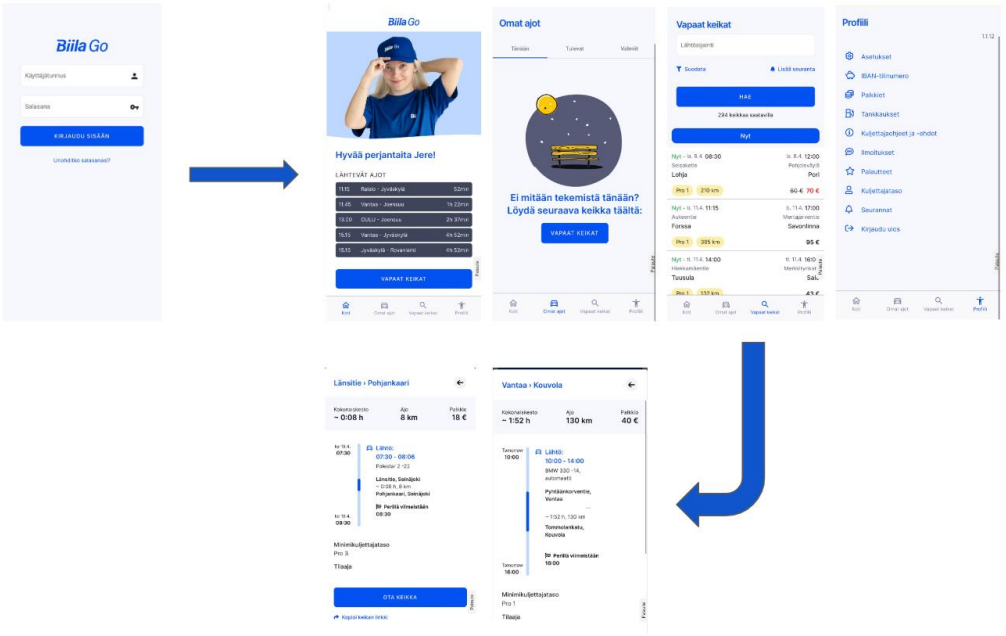
Kehittäjän näkökulmasta natiivisovelluksissa on omat haasteensa, mutta ne tarjoavat toisaalta useita etuja sovelluksen kehityksessä ja käyttäjäkokemuksen optimoimisessa. Natiivisovelluksella voidaan tarjota parempi käyttäjäkokemus sen edistyneemmillä ominaisuuksilla, kuten laitteen GPS-toimintoa ja Push-ilmoituksia hyödyntämällä. Natiivisovellukset tarjoavat myös tuen monelle eri ohjelmointirajapinnalle, joten ne voidaan sulauttaa helpommin osaksi isoa järjestelmää tai luoda kommunikaatiota eri järjestelmien välille. Kehittäjän käytössä ovat myös alustan natiivit kehitystyökalut, kuten SDK:t ja natiivit ohjelmointirajapinnat, joten sovelluksen käyttökokemusta voidaan parantaa maksimaalisesti. Natiivisovelluksella voidaan parantaa sovelluksen tietoturvaa, koska kehittäjä pystyy ohjelmoinnissa hyötymään alustan tietoturvaominaisuuksista sekä laitteen omista turvallisuusominaisuuksista, kuten Applella Touch-ID tai Face-ID. Natiivisovellusten kehittäminen vie kuitenkin paljon aikaa ja se on yleensä kallista, koska natiivit sovellukset täytyy luoda kullekin alustalle erikseen. Näin ollen ohjelmointiin käytettävä työmäärä saattaa moninkertaistua verrattuna eri selainpohjaisiin ratkaisuihin. Sovelluksen päivitykset ja ylläpito täytyy tehdä kullekin alustalle erikseen, joka myös lisää sovelluksen ylläpitoon liittyvää työmäärää. Natiivisovelluksen edut tulevatkin enimmäkseen esille käyttäjän päässä, eikä natiivisovellus välttämättä ole kehittäjälle aina paras ratkaisu, mutta sillä voidaan huomattavasti parantaa käyttäjäkokemusta sen tarjoamien toiminnallisten mahdollisuuksien avulla (Kaikkonen 2021).

3 Nykysovellus

Biilalla on tällä hetkellä käytössä selainpohjainen progressiivinen web-sovellus. Sovellus on näin ollen saatavilla kaikille alustoille selainpohjaista asennusta hyödyntäen. Sovellus on tarkoitettu Biilan kuljettajien toimintaan. Sovelluksen avulla kuljettajat pystyvät varaamaan vapaita siirtoajoja ja suorittamaan niitä. Tässä luvussa tutustutaan tarkemmin sovelluksen käyttöliittymän rakenteeseen ja toimintaan.

3.1 Ulkoasu ja käyttöliittymän rakenne

Biilan kuljettajasovellus omaa modernin nykypäivänä paljon käytetyn käyttöliittymän, jonka päänäkymää käyttäjä ohjaa alapalkissa olevaa valikkoa käyttäen. Ensin sovellus ohjaa käyttäjää kirjautumaan sisälle, jonka jälkeen käyttäjä pääsee sovelluksen päänäkymään. Sovelluksen päänäkymä koostuu neljästä eri sivusta, joiden välillä käyttäjä voi siirtyä alapalkkia hyödyntäen. Päävalikon palkista käyttäjä voi siirtyä joko Koti-, Omat ajot-, Vapaat keikat-, tai Profiili-näkymään. Sovelluksen värimaailma on Biilan brändiin perustuva sinivalkoinen. Alla oleva kuvio (kuvio 4) havainnollistaa sovelluksen näkymää ja käyttöliittymän rakennetta.

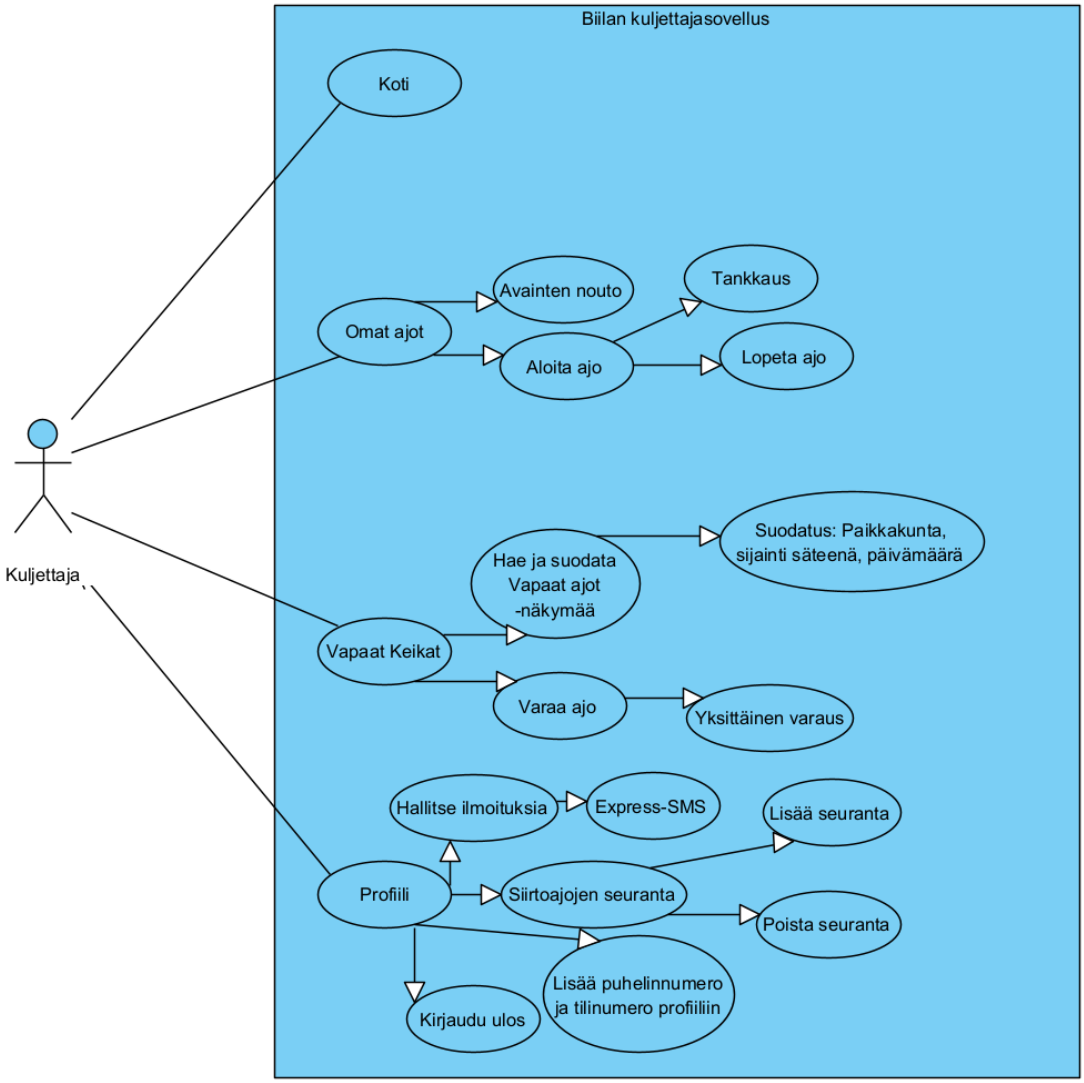


Kuvio 4. Biilan kuljettajasovelluksen käyttöliittymä (Biila, 2023)

3.2 Toiminta

Nykyinen sovellus toimii pääasiassa kuljettajien alustana siirtoajojen varaamiselle ja ajojen suorittamiselle. Sovellus esittää kaikki Biilalle lähetetyt siirtoajotoimeksiannot Vapaat keikat -sivulla, josta kuljettajat voivat varata haluamansa siirtoajon omatoimisesti. Siirtoajon varattuaan ajo siirtyy kuljettajan Omat ajot -sivulle, josta kuljettaja näkee tarkempia tietoja ajosta ja pystyy suorittamaan ajon. Ajon suorittaminen aloitetaan siirtymällä suoritettavaan ajoon painamalla sen kuvaketta Omat ajot -näkymässä, josta sitten painetaan Aloita ajo -painiketta. Tämän jälkeen järjestelmä kysyy kuljettajalta auton kilometrilukeman, mahdolliset lisä- ja vauriotiedot autoon liittyen sekä pyytää kuljettajaa lataamaan kuvat autosta ennen ajoa. Hyväksyttyään Biilan siirtoehdot, kuljettaja voi aloittaa ajon sovelluksessa, jolloin ajon aika lähtee rullaamaan. Kuljettaja voi kohdistaa sovelluksen kautta ajon aikana muutaman eri toimenpiteen. Kuljettaja voi pyytää tankkausta Biilalta, jolloin kuljettaja syöttää järjestelmään tankkausaseman tiedot sekä auton kilometrilukeman. Tämän perusteella Biila avaa tankkausmittarin kuljettajan pyynnön mukaan ja kuljettaja tankkaa tarvittavan määrän polttoainetta sekä lisää kuitin tankkauksesta järjestelmään.

Kuljettaja pystyy määränpäähän saavuttuaan myös ajonäkymästä painamaan lopeta ajo -painiketta, joka päättää ajon, jolloin järjestelmä pyytää kuljettajaa syöttämään auton kilometrilukeman, mahdolliset lisätiedot ajoon liittyen sekä liittämän uudet kuvat autosta ajon jälkeen. Nämä vaiheet suoritettuaan kuljettaja pystyy painamaan valmis-painiketta, joka päättää ajon ja kirjaa sen suoritetuksi. Näiden kahden toimenpiteen lisäksi kuljettaja pystyy sovelluksessa lisäämään haluamansa seurannan tiettyyn paikkaan eli kuljettaja saa ilmoituksen, joka kerta, kun hänen määrittämäänsä sijaintiin ilmestyy uusi siirtoajo. Toistaiseksi nämä ilmoitukset lähetetään sähköpostitse, mutta tulevaisuudessa Push-ilmoitukset voisivat mahdollistua esimerkiksi natiivisovelluksen kehittämisen myötä. Alla esitetty sovelluksen käyttötapauskaavio (kuvio 5).

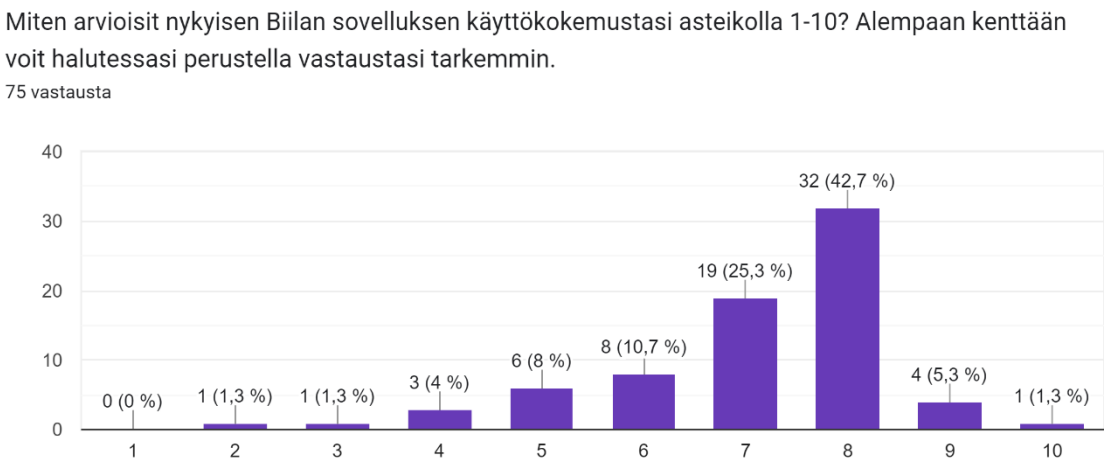


Kuvio 5. Biilan kuljettajasovelluksen käyttötapauskaavio.

3.3 Käyttökokemus

Sovelluksen käyttökokemus on pääpiirteissään miellyttävä ja sen käyttöliittymän rakenne on selkeä. Sovelluksen toiminnot ovat sujuvia ja ne on järkevästi sijoiteltu käyttäjää ajatellen. Sovelluksen ulkoasu on moderni sekä se mukailee hyvin Biilan luomaa brändi-ilmettä. Sovellus sisältää muutamia bugeja, jotka olisi hyvä korjata käyttäjäkokemuksen

optimoimiseksi. Bugeista lisää käyttäjäpalauteosiossa. Muutamia informaatiokenttiä olisi hyvä myös muuttaa, jotta tarjolla olevasta tilasta saataisiin maksimaalinen hyöty irti. Informaatiokentät saattavat monesti sisältää virheellistä tietoa. Informaatiokentistä lisää käyttäjäpalautteessa. Sovelluksen toimintoja voisi sujuvoittaa, sillä monesti sovellus toimii jokseenkin aika hitaasti. Alla olevassa kaaviossa (kuvio 6) esitetty käyttäjäpalautetta sovelluksen tämänhetkisestä toiminnasta. Arvosanoja kerättiin asteikolla 1–10, josta suosituin arvosana oli 8.



Kuvio 6. Käyttäjien kokemus Biilan nykyisestä sovelluksesta asteikolla 1–10. (Käyttäjäpalautekysely)

4 Sovellusvaihtoehtojen sopivuus Biilalle

Tämän luvun tarkoitus on selvittää, mikä kolmesta sovellustyypistä olisi paras vaihtoehto Biilalle ja jonka kehittämistä lähdetään pohtimaan myöhemmin tässä työssä. Kullakin sovellustyypillä on omat vahvuutensa ja heikkoutensa, joten valinta ei ole lähes ikinä yksiselitteinen.

4.1 Natiivi-, web- vai hybrid-sovellusten erot ja yhtäläisyydet

Aiemmin tässä työssä alustariippumattomiin ja alustariippuvaiseen sovelluksiin luokitelluilla sovellustyypeillä on olennaisesti myös merkittäviä eroja sekä yhtäläisyyksiä. Alla olevassa taulukossa (taulukko 1) on esitetty keskeisimmät sovellukseen liittyvät tekijät sovellustyypeittäin.

Käyttökokemukseltaan paras sovellusvaihtoehto on natiivisovellus. Se tarjoaa alustalle ominaisen ulkoasun ja tuntuman hyvää suorituskykyä unohtamatta. PWA- ja hybridisovellus eivät tähän pysty. Ylläpidoltaan helpoin ja edullisin vaihtoehto on PWA-sovellus yhteisen koodikantansa ansiosta, kun taas kallein on natiivisovellus alustaspesifillä koodikannallaan. Natiivisovellus on myös samasta syystä toteutukseltaan kallein vaihtoehto. Sovellusvaihtoehdoista ainoa, jota ei tarvitse asentaa laitteen sovelluskaupasta on PWA-sovellus. Muilla vaihtoehdoilla jakelu tapahtuu laitteen sovelluskaupasta. Natiivisovellus taas tarjoaa parhaan tuen laitteen ja sen myötä sovelluksen toiminnoille. Natiivisovellus tuo mukanaan myös optimaalisen suorituskyvyn, johon muut sovellustyypit eivät pysty.

	PWA	Hybridi	Natiivi
Käyttökokemus	Yhtenäinen toimintatapa kaikilla laitteilla. Käyttökokemus voi vaihdella alustan mukaan.	Kahta muuta hitaampi ja yksinkertaisempi sovellus. Tuntuma vaihtelee sovelluksen mukaan.	Alustan ominainen ulkoasu ja tuntuma. Hyvä suorituskyky sekä optimoidut toiminnot.
Ylläpito	Ylläpito edullista ja helppoa. Päivitykset suoraan palvelimelle.	Ylläpito edullista yhteisen koodikannan ansiosta. Kuitenkin alustalle ominaiset komponentit.	Kalliimpi ja työläämpi ylläpitää useiden alustoille natiivien koodikantojen vuoksi.
Hinta	Yksi koodikanta, joten edullinen toteuttaa.	Web-teknologioihin perustuva yksi koodikanta, joten edullinen natiiviin verrattuna.	Joka alustalla oma koodikanta, joten toteuttaminen on kallista.
Jakelu	Jako omalla domainilla Ei asennusta laitteelle, vaan toimii suoraan selaimessa.	Asennettava laitteen sovelluskaupasta.	Asennettava laitteen sovelluskaupasta.
Toiminnot	Tukee laitteen yksinkertaisia toimintoja.	Tukee laitteen yksinkertaisia toimintoja.	Tukee laitteen ja käyttöjärjestelmän kaikkia toimintoja.

Taulukko 1. Sovellustyyppien erot ja yhtäläisyydet.

4.2 Hinta ja ylläpito

Sovelluksen kehittämiseen tarvittavat resurssit ja ylläpito muodostavat sovelluksen elinkaaren aikana suurimman osan sovellukseen liittyvistä kustannuksista (Riikola, 2022).

Sovelluksen kehitysvaiheessa muodostuu erilaisia kuluja ohjelmakoodin tuottamisesta julkaisualustan julkaisulisenssien hankkimiseen. Ylläpito vaatii pääosin henkilöstöpuolen resursseja ohjelmakoodin tuottamisen ja päivitysten julkaisun muodossa. Biilan tapauksessa sovelluksen kehittäminen sekä ylläpito onnistuvat pääosin yrityksen omilla resursseilla, joten työläänkin sovelluksen kehittäminen voi olla kannattavaa, mikäli näin voidaan saavuttaa halutut kriteerit sovelluksen toiminnassa.

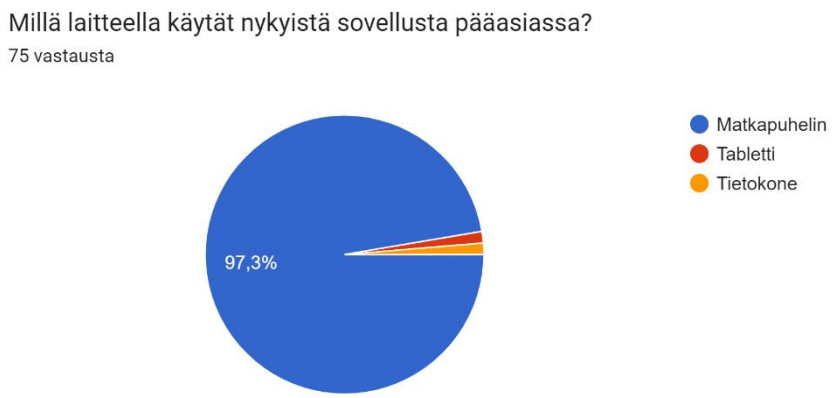
Lähtökohtaisesti Natiivisovellus on PWA- ja hybridisovellusta kalliimpi ratkaisu kehittämisen ja ylläpidon vaatimiin resursseihin nähden. Jo pelkästään natiivisovelluksen yhden päivityksen yhteydessä joka alustalle on koodattava oma toteutus. Näin ollen hinnan puolesta PWA- ja hybridisovellus ovat optimaalisempia ratkaisuja, mutta natiivisovellus tuo taas omat etunsa muilla osa-alueilla.

4.3 Tavoittavuus ja toiminnot

Sovellusta kehitettäessä on tärkeää miettiä, miten se tavoittaa käyttäjänsä ja millaisia toimintoja sen käyttäjät tulevat tarvitsemaan. Laitteesta riippumattomassa tavoitettavuudessa PWA on näistä kolmesta vaihtoehdosta hyvin vahvoilla, sillä sovelluksen voi käynnistää millä tahansa laitteella, jossa on internet-selain sekä internet-yhteys. Natiivi- sekä hybridisovellus taas tarjoavat hieman erilaisempaa bränditaktiikkaa, sillä niiden jakelussa voidaan käyttää jo kaikkien tuntemia Applen ”Download on the App Store” ja ”Available on the Google Play” -nappeja, mistä käyttäjä pääsee suoraan lataamaan sovelluksen.

Toiminnoissaan ja suorituskyvyssään natiivisovellus on taas ehdoton ykkönen. Sen alustakohtaisen koodikannan ja alustalle optimoitujen toimintojen ansiosta käyttäjä voi saada erinomaisen käyttökokemuksen. PWA- ja hybridisovellus tarjoavat taas heikompaa suorituskykyä yksinkertaisemmilla toiminnoilla. Lyhyet latausajat sekä laaja toiminnallisuus ovat hyviä valttikortteja käyttäjien tavoittelemisessa.

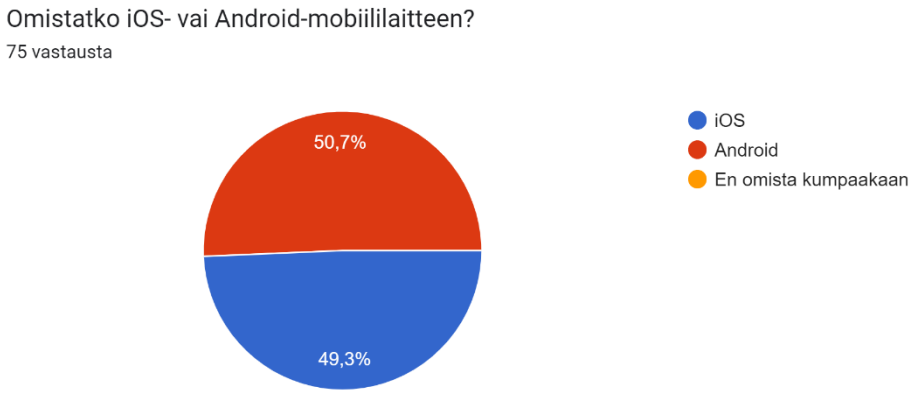
Biilan tapauksessa kuljettajat tekevät työtään pääasiassa tien päällä, joten sovelluksen hyvä suorituskyky ja laajat toiminnot tulevat tarpeeseen. Tien päällä myös puhelin on lähtökohtaisesti kätevin työväline, koska se on pieni ja omaa siihen nähden hyvän suorituskyvyn. Käyttäjäpalautekyselyn yhteydessä laadittu alla oleva kaavio (kuvio 7) esittää Biilan käyttäjien suosimat laitteet sovelluksen käytössä. Kaaviosta voidaan huomata, että lähes jokainen (97,3 %) käyttää nykyistä sovellusta matkapuhelimella ja vain pieni murto-osa käyttää tablettia tai tietokonetta. Käyttäjät myös raportoivat puhelimen käytön syyksi sen, että se on aina mukana ja sitä kautta on helpointa pysyä ajan tasalla vapaisiin siirtoajoihin liittyen. Osa käyttäjistä ei myöskään omista muuta laitetta puhelimen lisäksi.



Kuvio 7. Biilan kuljettjien käyttämät laitteet. (Käyttäjäpalautekysely)

Käytettävän laitteen lisäksi vastanneilta kerättiin tietoa siitä, millä mobiilikäyttöjärjestelmällä he käyttävät puhelintaan. Alla oleva kaavio (kuvio 8) esittää prosentuaaliset osuudet Biilan kuljettajien käyttämistä mobiilikäyttöjärjestelmistä. Kaaviosta voidaan huomata, että lähes puolet (49,3 %) omistavat Applen iOS-käyttöjärjestelmällä toimivan puhelimen, kun taas puolet (50,7 %) omistavat Googlen Android-käyttöjärjestelmällä toimivan puhelimen. Kyselyssä oli myös tarjolla vaihtoehto ”En omista kumpaankaan” tai ”Muu”. Yksikään vastaaja ei valinnut näitä

vaihtoehtoja, joten voidaan lähtökohtaisesti olettaa, että Biilan kuljettajat omistavat joko iOS- tai Android-laitteen.



Kuvio 8. Biilan kuljettajien käyttämät mobiilikäyttöjärjestelmät. (Käyttäjäpalautekysely)

4.4 Käyttäjäpalautekysely

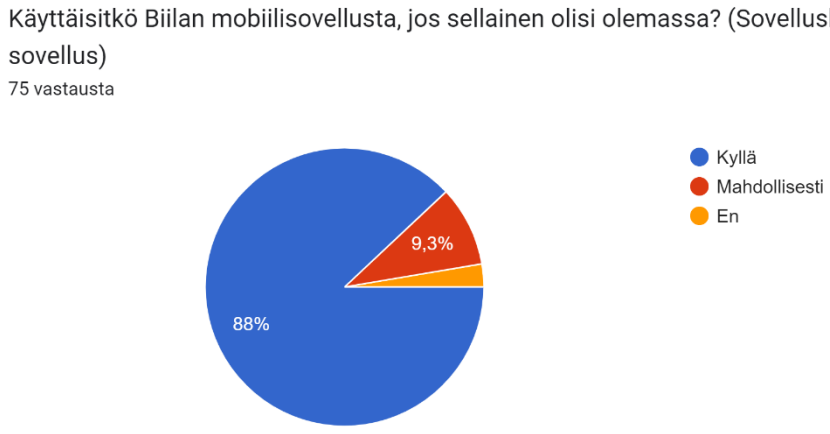
Tämän työn yhteydessä teetettiin Biilan nykyiseen sovellukseen liittyvä käyttäjäpalautekysely Biilan kuljettajille. Kuljettajilta kysyttiin Google Forms -kyselylomakkeella muutama pakollinen kysymys, joiden lisäksi kuljettaja sai vastata halutessaan muutamiin vapaaehtoisiin kysymyksiin. Kyselyyn vastasi kaiken kaikkiaan 75 kuljettajaa. Alla listattuna käyttäjäpalautekyselyssä kysytyt kysymykset järjestyksessä. Osaa kysymysten tuottamista vastauksista on hyödynnetty tässä työssä jo aiemmin.

1. **Miten arvioisit nykyisen Biilan sovelluksen käyttökokemustasi asteikolla 1-10?**
2. **Tähän kenttään voit halutessasi jättää perustelut edellä antamallesi arviolle.**
3. **Onko sovelluksen käyttöliittymä mielestäsi toimiva?**
4. **Millä laitteella käytät nykyistä sovellusta pääasiassa?**
5. **Miksi käytät kyseistä laitetta?**
6. **Omistatko iOS- vai Android-mobiililaitteen?**
7. **Milloin käytät nykyistä sovellusta mobiililaitteella?**
8. **Käyttäisitkö Biilan mobiilisovellusta, jos sellainen olisi olemassa? (Sovelluskaupasta ladattava sovellus)**

9. **Jos vastasit edelliseen ei, niin miksi et?**
10. **Tähän vapaaehtoiseen kenttään voit antaa ideoita ja palautetta sovellusta varten. (esim. uusia toimintoja, parannuksia vanhaan jne.)**
11. **Mitä tahansa sovellukseen liittyvää asiaa:**

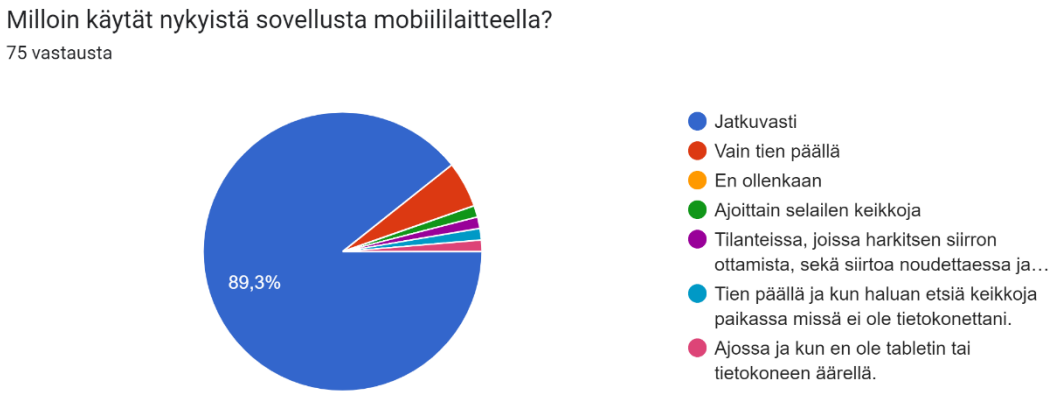
4.5 Yhteenveto

Vedoten käyttäjäpalautteeseen sekä Biilan nykyiseen tilanteeseen suosittelen Biilaa kehittämään natiivisovelluksen kuljettajien käyttöön. Natiivisovellus tuo merkittävää etua sovelluksen toimivuudessa sekä suorituskyvyssä, mikä nousi ongelmaksi käyttäjäpalautekyselyn perusteella. Natiivisovellus tarjoaa myös luotettavuutta, jota tien päällä oleva kuljettaja tarvitsee toiminnan onnistumiseksi. Natiivisovelluksen riski yllättävään kaatumiseen tai sammumiseen on pienempi natiivin koodipohjan ja sujuvan alustakohtaisen toiminnan myötä. Käyttäjäpalautekyselyssä kerättiin tietoa siitä, kuinka moni kuljettaja käyttäisi Biilan natiivisovellusta, jos sellainen kehitettäisiin. Alla oleva kaavio (kuvio 9) esittää kuljettajien kiinnostuksen natiivisovellusta kohtaan. 88 % kuljettajista vastasi suoraan, että käyttäisi sovellusta, jos sellainen kehitettäisiin. 9,3 % kuljettajista vastasi mahdollisesti käyttävänsä sovellusta. Loput 2,7 % vastasi, ettei käyttäisi sovellusta. Näin ollen 97,3 % kuljettajista on myönteisiä uuden natiivisovelluksen käyttämiselle. Tätä voidaankin jo pitää hyvänä perusteena sovelluksen kehittämiselle.



Kuvio 9. Kuljettajien kiinnostus natiivisovellusta kohtaan. (Käyttäjäpalautekysely)

Kyselyn perusteella 90,6 % kuljettajista käyttää sovellusta jatkuvasti, joten sovelluksen toiminnan olisi hyvä olla luotettavaa sekä sujuvaa. Loput kuljettajista käyttävät sovellusta vain ollessaan ajossa. Alla oleva kaavio (kuvio 10) esittää kuljettajien sovelluksen käytön mobiililaitteella.



Kuvio 10. Kuljettajien sovelluksen käyttö. (Käyttäjäpalautekysely)

5 Natiivisovelluksen suunnittelu

Tässä luvussa luodaan suunnitelma natiivisovelluksen toteutukselle. Itse sovellus toteutetaan tämän luvun suunnitelman mukaan teoriamuodossa luvussa 6. Ensin määritellään, mitä tehdään. Sen jälkeen suunnitellaan sovelluksen toimintaa ja rakennetta käyttäjäpalautteen ja omien ideoiden perusteella. Lopuksi pohditaan suunnitelmaa sovelluksen toteutukselle.

5.1 Työn määrittely

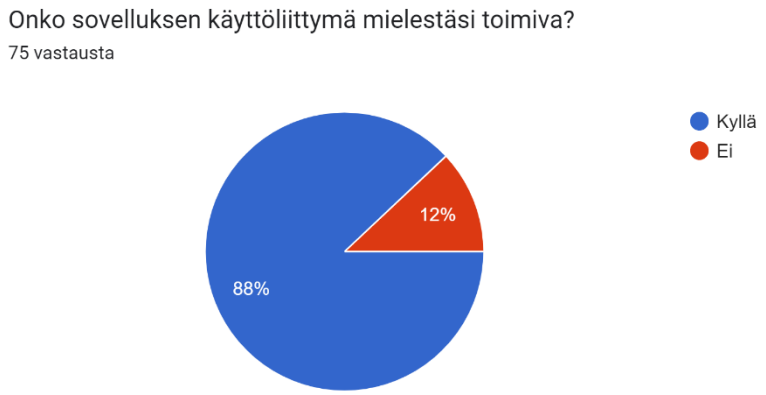
Työn tarkoitus on luoda Biilalle teoriapohja natiivisovelluksen toteuttamiseen Biilan kuljettajille. Uuden natiivisovelluksen kehittäminen nykyisen progressiivisen web-sovelluksen (PWA) tilalle on tarpeen, koska natiivisovelluksella voidaan saavuttaa merkittävästi parempi sovelluksen suorituskyky ja paljon nykyistä monipuolisemmat toiminnot. Natiivisovelluksen haasteena on se, että sen kehittäminen ja ylläpito vaativat enemmän resursseja kuin progressiivisen web-sovelluksen kehitys ja ylläpito. Sovellus on tarkoitus kehittää sekä iOS- että Android-käyttöjärjestelmille. Odotan uuden natiivisovelluksen parantavan kuljettajien käyttökokemusta merkittävästi helpottamalla ja nopeuttamalla heidän työtään.

5.2 Suunnittelu ja käyttäjäpalaute

Tässä alaluvussa käsitellään käyttäjäpalautekyselyssä esille tulleita käyttäjien ideoita sovellukseen liittyen. Suunnitellaan sen perusteella sovelluksen mahdollista käyttöliittymää ja sen sisältämiä toimintoja. Tuon myös tässä osiossa esille omia ideoitani sovellukseen liittyen.

Sovelluksen käyttöliittymä on hyvin olennainen osa käyttäjän käyttökokemusta. Näin ollen sovelluksen käyttöliittymän tulee omata järkevä rakenne. Kyselyn perusteella suurin osa käyttäjistä on sitä mieltä, että sovelluksen käyttöliittymä on toimiva nykyisessä muodossaan. Alla olevassa kaaviossa (kuvio 11) on esitetty käyttäjien

mielipiteitä käyttöliittymän toimivuudesta, kun vastausvaihtoehdot olivat joko ”Kyllä” tai ”Ei”. 88 % käyttäjistä oli sitä mieltä, että käyttöliittymä on nykyisellään toimiva. Loput 12 % totesivat, että käyttöliittymä ei ole toimiva. Nykyinen käyttöliittymä koostuu neljästä välilehdestä, joiden välillä käyttäjä voi liikkua käyttämällä sovelluksen alaosassa olevaa navigointipalkkia. Tästä tarkempi kuvaus edellä olevassa ”Nykyinen sovellus”-luvussa (luku 3). Tämän kaavion perusteella voidaan myös todeta, että vastaavaa käyttöliittymää voidaan käyttää myös tulevassa natiivisovelluksessa.



Kuvio 11. Nykyisen käyttöliittymän toimivuus. (Käyttäjäpalautekysely)

Käyttäjiltä kerättiin myös avointa palautetta. Avoimissa kentissä käyttäjät saivat antaa palautetta sovelluksen käyttöliittymästä sekä tuoda esille omia ideoita sovelluksen kehittämiseen. Nämä avoimet kentät keräsivät monia hyviä mielipiteitä ja ideoita. Alla listattuna pääkohtia sovelluksen hyvistä ja huonoista puolista käyttäjien palautteen perusteella.

Nykyisen sovelluksen hyvät puolet:

- ✓ Helppo ja toimiva käyttöliittymän rakenne
- ✓ Toimii käytännöllisesti sekä helppo ottaa käyttöön
- ✓ Toimii ajoittain hyvin ja ”sillä tulee toimeen”

Nykyisen sovelluksen huonot puolet:

- Ajoittainen huono suorituskyky sekä bugit ja toimintahäiriöt
- Ongelmia ja virheilmoituksia ladattaessa kuvia ajon tietoihin
- Monen käyttäjän mielestä natiivisovellus olisi parempi vaihtoehto
- Ajojen ketjutus vaikeaa nykyisillä toiminnoilla sekä sähköposti-ilmoitukset epäkäytännöllisiä
- Vapaat ajot -listassa vähän ominaisuuksia, kuten suodattimia siirtoajoille

Uusia käyttökelpoisia ideoita syntyi palautekyselyssä huomattava määrä. Alla listattuna ainakin niitä ideoita omien ideoideni lisäksi, joita aion käyttää sovelluksen toteutusta miettiessä.

Käyttäjien ideoita ja toiveita:

- Siirtoajojen ketjutukseen uusia tapoja
    - Esimerkiksi ajojen monivaraus
    - Haku tietyn pituisille siirroille
    - Haku Pro-tason mukaan
    - Näkyviin vain siirtoajot, joiden määränpäästä on toinen siirtoajo tarjolla
- Push-ilmoitukset uusille siirtoajoille
- Ajon reitti näkyviin kartalle jo ennen ajon varaamista, kartalla näkyy ympyrän sisällä ajon lähtöpaikan ja määränpään sijainti esimerkiksi 2 km säteellä
- Siirtoajojen seurannat väliaikaisesti tauolle
    - Tietystä kaupungista lähteviin ja sinne saapuviin ajoihin yksi seuranta (päälle/pois)
    - Ajossa olevan siirtoajon määränpäästä lähtevät autot (päälle/pois)
    - Yhdestä kaupungista tietyllä säteellä olevat siirtoajot (päälle/pois)
- Avainten noudon peruminen sekä useampien siirtoajojen avainten haun avaaminen

- Vapaat ajot -näkymään automaattinen päivitys eli uudet ajot tulevat listalle reaaliajassa
- Tilaajien avainten luovutustiedot aina ajon tietoihin, esim. avainsäiliöiden sijainnit sekä tilaajan ja asiakkaan tiedot ja puhelinnumerot eritellysti aina ajon tietoihin
- Sähköinen ajolupapohja
- Odotuskorvauksiin sovelluksen sisäinen lomake ja sen tila näkyviin käyttäjälle
- Lisää tietoja ajettavasta autosta esim. auton käyttövoima (bensa, diesel, kaasu tms.)
- Nykyisen sovelluksen jatkokeikat näkyvät väärin tai eivät näy ollenkaan
- Session terminointi väljemmäksi
- Uuden siirtoajon kohdalle jokin efekti esim. väläytys
- Kuljettajan status: etsii siirtoajoa (sijainti), passiivinen

Uusi natiivisovellus voisi siis toimia vastaavalla käyttöliittymällä, jolla nykyinenkin sovellus toimii. Käyttöliittymä todettiin palautteen perusteella pääosin hyvin toimivaksi. Yllä olevat käyttäjien ideat ja toiveet huomioidaan luvun 6 sovelluksen toteutusta mietittäessä.

5.3 Toteutussuunnitelma

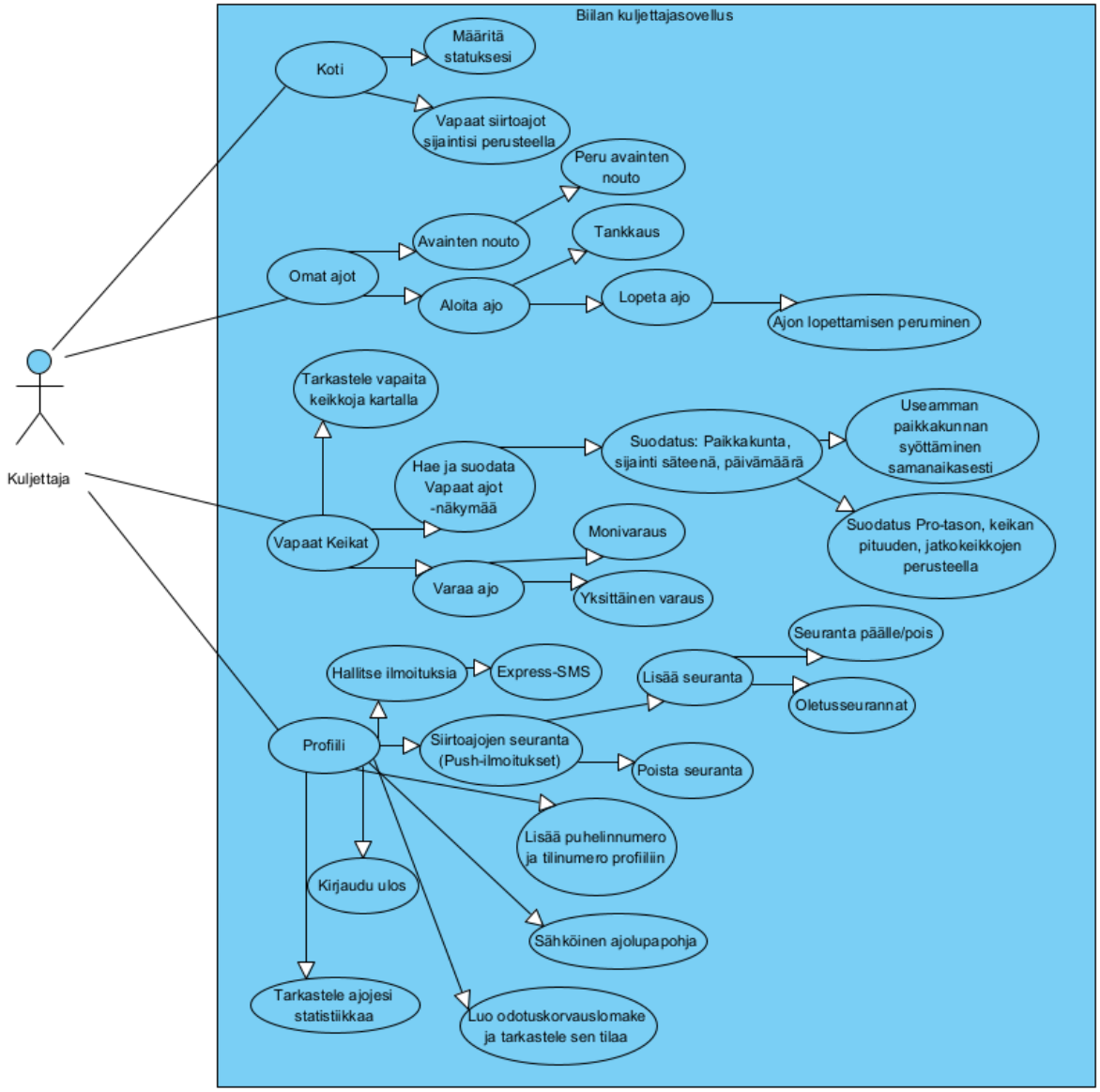
Työ toteutetaan tämän kandidaatintutkielman luvussa 6. Ensin alaluvussa 6.1 natiivisovellukselle esitetään vaatimusmäärittely ja käyttötapauskaavio eli yksinkertaistettuna se, mitä toimintoja sovellus pitää sisällään ja missä suhteessa ne ovat toisiinsa. Vaatimusmäärittely toteutetaan Visual Paradigm -ohjelmalla. Tämän jälkeen alaluvussa 6.2 kuvataan sovelluksen rakenne ja siihen mahdollisesti lisättävät ominaisuudet. Lopuksi 6.3 alaluvussa käydään natiivisovelluksen toteuttaminen vaihe vaiheelta yksityiskohtaisesti läpi. Näin luodaan ohjeet sovelluksen toteuttamiseen käytännössä. Työn lopputuloksena syntyy selvitys omasta mielestäni järkevän natiivisovelluksen kehittämisestä Biilalle.

6 Natiivisovelluksen toteutus

Tässä luvussa määritellään toteutus Biilan uudelle natiivisovellukselle. Sovellukselle luodaan vaatimusmäärittely, josta edetään sovelluksen rakenteen ja ominaisuuksien tarkasteluun. Lopuksi kuvataan yksityiskohtainen toteutus teoriassa natiivisovellukselle.

6.1 Vaatimusmäärittely

Uusi natiivisovellus voi sisältää edellä esitettyjen (luku 3) nykyisten toimintojen lisäksi käyttäjien ideoiden perusteella luotuja uusia toimintoja. Sovellus sisältää nykyisellään jo sille olennaisimmat toiminnot: käyttäjä voi varata siirtoajon sekä suorittaa siirtoajon ja siihen liittyvät toimenpiteet. Näitä toimintoja optimoimalla voidaan saavuttaa entistä parempi käyttökokemus. Havainnollistan uuden sovelluksen toimintoja käyttötapauskaavion avulla. Alla esitettynä uudelle sovellukselle mahdollinen käyttötapauskaavio (kuvio 12). Käyttötapauskaaviota voidaan verrata nykyisen sovelluksen käyttötapauskaavioon (kuvio 5, s. 21). Näin on helpompi erottaa sovellukseen uudet suositeltavat ominaisuudet. Kaaviossa sovelluksen käyttäjänä toimii Biilan kuljettaja, joka kohdistaa sovellukseen toimenpiteitä. Kaaviossa on esitetty sovelluksen pääsivujen (Koti, Omat ajot, Vapaat keikat, Profiili) takana olevat toiminnot ja niiden rakenteellinen suhde toisiinsa. Kaavion sisältämät uudet toiminnallisuudet on kuvattu tarkemmin luvussa 6.2.

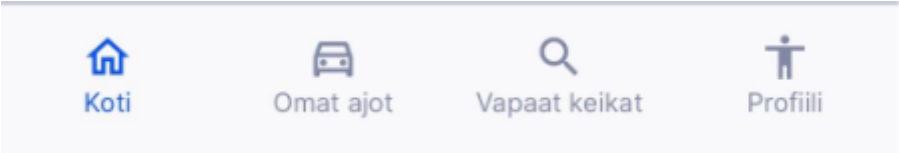


Kuvio 12. Natiivisovelluksen käyttötapauskaavio.

6.2 Sovelluksen rakenne ja ominaisuudet

Nähdäkseni nykyisen sovelluksen käyttöliittymän rakenne on järkevä ja sitä voidaan hyödyntää myös uudessa natiivisovelluksessa. Natiivisovellus voi siis sisältää pääsivut: Koti, Omat ajot, Vapaat ajot ja Profiili. Alla oleva kuva (kuvio 13) esittää nykyisen sovelluksen käyttöliittymän rakenteen. Etenkin Omat ajot, Vapaat ajot sekä Profiili -

sivujen toiminnallisuus on riittävä niiden olemassaolon kannalta. Koti-välilehti ei tällä hetkellä sisällä ollenkaan toiminnallisuutta. Tässä alaluvussa käyn edellisen alaluvun 6.1 kuvion 12 perusteella läpi uuteen natiivisovellukseen hyödylliseksi näkemiäni toimintoja. Käyn pääsivujen perusteella uudet toiminnot läpi järjestyksessä Koti→Omat ajot→Vapaat Keikat→Profiili.



Kuvio 13. Natiivisovelluksen käyttöliittymän mahdollinen rakenne. (Biila)

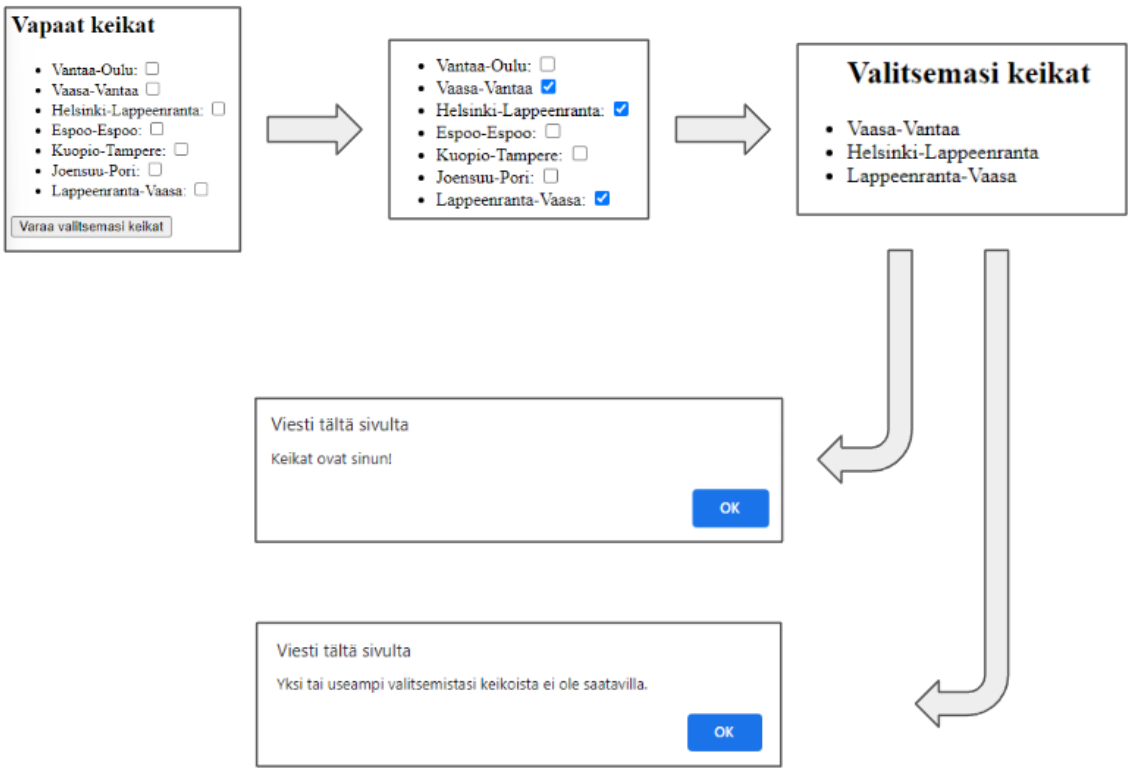
Tässä kappaleessa kuvaan Koti-välilehden mahdollista tulevaa toiminnallisuutta. Koti-välilehti ei edellä mainitsemani mukaan sisällä mitään toiminnallisuutta. Nykyisellään siitä voi seurata lähteviä ajoja. Sovelluksella on hyvä olla etusivu, joka tuo esille sovelluksen omistajan, joka toteutuukin nykyisessä sovelluksessa. Lähtevät ajot -näkymä on nähdäkseni turha käyttäjän kannalta. Tämän voisi korvata tulevassa natiivisovelluksessa näkymällä, joka esittää käyttäjän sijainnista tai sen lähistöltä lähtevät siirtoajot. Näin käyttäjä heti näkisi tarjolla olevat siirtoajot, joka näin ollen myös sujuvoittaisi käyttökokemusta. Koti-välilehdellä voisi olla myös painike, jolla kuljettaja voisi määrittää oman statuksensa. Status voisi olla ”Etsii siirtoajoa”, ”Ajossa” tai ”Passiivinen”. Näin kuljettaja voisi helposti ilmaista Biilan henkilökunnalle, mikä on hänen tilansa tällä hetkellä. Biilan henkilökunta voisi heidän listanäkymästään nähdä kaikki kuljettajat, jotka ovat vailla siirtoajoa. Näin henkilökunta voisi tarjota kuljettajalle siirtoajoa kuljettajan sitä etsiessä. Lisäksi Biilan henkilökunta voisi nähdä statuksen avulla kuljettajan nykyisen sijainnin, mikäli kuljettaja on antanut tähän lupansa. Sijainti olisi myös hyvä olla piilotettavissa kuljettajan toimesta. Tämän myötä Biilan henkilökunta voisi tarjota sopivaa ajoa sopivassa sijainnissa olevalle kuljettajalle ja näin siirtoajo löytäisi nopeammin kuljettajansa.

Tässä kappaleessa keskityn Omat ajot -välilehteen. Omat ajot välilehden tulee sisältää toiminnallisuudet omien siirtoajojen hallintaan, auton avainten outoon, siirtoajon aloittamiseen sekä ajon lopettamiseen. Siirtoajoa suoritettaessa pystytään merkittävästi hyötymään natiivisovelluksen mahdollistamasta laitteiston ominaisuuksien käytöstä, kuten kamerasta ja sijainnin hakemisesta. Avainten noutoon liittyen uuteen sovellukseen voisi tehdä avainten noudon peruuttamisen mahdolliseksi esimerkiksi käyttäjän sijainnin avulla. Käyttäjä voi avata avainten noudon, mikäli hän on tietyllä säteellä avainten noutopisteestä. Tämän lisäksi noudon peruutus onnistuu käyttäjän ollessa tämän säteen sisällä. Näin voitaisiin myös mahdollistaa useiden avainten noutojen avaaminen samanaikaisesti. Ajon tietoihin olisi hyvä lisätä myös enemmän tietoa ajettavasta autosta, esimerkiksi mikä on sen käyttövoima.

Ajon aloittaminen ja tankkaustoiminto ovat nykyisellään palautteen perusteella rakenteeltaan pääosin toimivia. Merkittävin osa palautteesta keskittyi ajon aloittamisen yhteydessä ladattaviin kuviin, joiden lataamisessa ilmenee palautteen perusteella lähes jatkuvasti ongelmia. Tähän liittyvät toimintahäiriöt olisi hyvä korjata uuteen natiivisovellukseen ja natiivisovellus itsessään jo varmasti helpottaa kuvien latausongelmia. Lisäksi ajon lopettamisen peruminen olisi hyvä mahdollistaa esimerkiksi vahinkopainallusten sattuessa. Tähän voisi myös hyödyntää laitteen GPS-sijaintia.

Vapaat keikat -välilehdessä on myös potentiaalia monille lisätoiminnoille. Suuri osa palautteesta keskittyi ajojen suodattamiseen listanäkymässä sekä ajojen ketjutukseen. Tässä kappaleessa esittelen mahdollisia lisäominaisuuksia ajojen suodattamiseen sekä hieman pidemmälle viedyn version nykyisestä ajojen varaussysteemistä. Nykyisten suodatustoimintojen lisäksi uusi sovellus voisi sisältää suodatuksen kuljettajan Pro-tason mukaan, ajon pituuden mukaan sekä näkymän ajoille, joiden määränpäästä on jatkokeikkaa. Erillinen suodatus olisi siis hyvä olla ajon kilometrimäärän perusteella. Myös usean paikkakunnan hakeminen samanaikaisesti olisi hyvä mahdollistaa. Vapaat keikat -näkymän olisi hyvä päivittyä automaattisesti ilman käyttäjän toimia.

Siirtoajoille voisi mahdollistaa ns. monivarauksen eli kuljettaja pystyy varaamaan monta siirtoajoa kerralla tietyn näkymän avulla. Nykyisellään kuljettajan täytyy varata yksi siirtoajo kerrallaan, jolloin mahdolliset jatkokeikat voivat jäädä saamatta tai siirtoajoa ei yksinkertaisesti uskalla ottaa, koska ei ole varmaa, saako sille jatkokeikkaa. Ideana on siis, että käyttäjä voi nykyiseltä listalta valita useamman siirtoajon ja varata ne yhtä aikaa. Käyttäjä valitsee haluamansa ajot keräämällä ne nykyiseltä listalta, jonka jälkeen sovellus näyttää hänelle hänen valitsemansa ajot listana. Tämän jälkeen käyttäjä voi painaa ”varaa valitsemasi ajot”-painiketta, joka varaa kaikki käyttäjän valitsemat ajot käyttäjälle. Mikäli jokin ajoista ei ole enää saatavilla käyttäjän painaessa varaa ajot-painiketta, käyttäjä saa virheilmoituksen ”Yksi tai useampi valitsemistasi ajoista ei ole enää saatavilla”. Varauksen onnistuessa sovellus antaa viestin ”Keikat ovat nyt sinun!”. Lisäksi monivarausnäkymä voisi sisältää hieman informaatiota ajoista, jotta käyttäjä tietää, mitä hän on varaamassa. Tein yksinkertaistetun alla olevan HTML-toteutuksen (kuvio 14) havainnollistaakseni tätä ideaa.



Kuvio 14. Monivarauksen havainnollistaminen.

Profiili-välilehti sisältää muutamia toimintoja ja niiden lisäksi informatiivista tietoa. Uudessa sovelluksessa käyttäjän profiilista voisi löytyä statistiikkaa käyttäjän siirtoajoista. Esimerkiksi nykyhetkeen mennessä kertyneet kaikkien siirtoajojen kilometrit ja siirtoajojen keskipituus kilometreinä ja ajallisesti. Käyttäjien palautteiden eniten huomiota saanut asia oli uusien siirtoajojen seurantailmoitukset. Nykyisessä sovelluksessa ilmoitukset tulevat sähköpostitse, mutta natiivisovellus mahdollistaa Push-ilmoitusten käytön, joka oli erittäin haluttu asia palautteissa. Uuden seurannan lisäämisen ja poistamisen lisäksi seuranta olisi hyvä saada pois päältä väliaikaisesti poistamatta sitä. Näin käyttäjä voisi aktivoida seurannan sitä tarvitessaan. Myös suora seurannan aktivointi ajossa olevan keikan määränpäästä sekä tietystä kaupungista ja siitä käyttäjän määrittämällä säteellä tulevat uudet keikat olisi hyvä saada ilmoituksina. Käyttäjä voisi myös määrittää profiilistaan ”kotikaupungin”, josta hän saisi halutessaan suoraan ilmoitukset puhelimeensa lähtevistä ja saapuvista ajoista. Profiiliin voisi myös sisällyttää sähköisen ajolupapohjan, jonka siirtoajon tilaaja voisi omasta järjestelmästään kuitata hyväksytyksi. Tämä sujuvoittaisi käyttäjien työtä palautteiden perusteella. Odotuskorvauslomake olisi hyvä sisällyttää sovellukseen ja sen tila saada näkyviin kuljettajalle. Näin käyttäjä voi tarkistaa useamman tätä nykyä eri järjestelmässä olevan asian yhdestä sovelluksesta helposti. Seuraava luku 6.3 käsittelee natiivisovelluksen toteutusta käytännössä.

6.3 Natiivisovelluksen toteutusvaiheet

Natiivisovelluksen toteuttaminen vaatii huomattavan määrän työtunteja sekä alustakohtaista ohjelmointiosaamista. Kehittäjien tulee toteuttaa tässä tapauksessa iOS- ja Android-alustalle omat erilliset toteutukset alustojen omilla kirjastoilla sekä komponenteilla. Tässä alaluvussa kuvaan työvaiheet natiivisovelluksen toteuttamiseen yksityiskohtaisesti iOS- ja Android-alustalle.

6.3.1 iOS

Applen iOS-käyttöjärjestelmälle luotava sovellus kehitetään käyttämällä Applen ohjelmakirjastoja sekä työkaluja. Natiivin iOS-sovelluksen kehittämistä varten kehittäjällä tulee olla Applen Mac-tietokone, sillä Applen kehitysympäristö Xcode on saatavilla ainoastaan MacOS-käyttöjärjestelmälle. Xcode tarvitaan nimenomaan natiivin iOS-koodin kääntämiseen, mutta koodin voi luonnollisesti kirjoittaa myös muussa ympäristössä. Lisäksi kehittäjä tarvitsee Apple Developer -tilin, joka maksaa 99 dollaria vuodessa. Jäsenyys kuitenkin tarvitaan vasta sovelluksen julkaisuvaiheessa, joten sovelluksen kehittämisen voi aloittaa jo heti, kun Xcode on ladattu ja asennettu Mac-tietokoneelle. Uusin versio Xcode:sta on Xcode 14, joka on saatavilla Applen sivuilta.

Sovelluksen ohjelmakoodin voi kirjoittaa joko Swiftillä, Objective-C:llä tai niiden yhdistelmällä. Apple itse suosittelee sovellusten kehittämiseen Swift-kieltä, koska sen avulla kehitetyt sovellukset ovat Applen mukaan turvallisempia ja nopeampia. Sovelluksen kehittäminen aloitetaan luomalla sille oma Xcode-projekti. Projektin sisäisiin tiedostoihin kirjoitetaan sen ohjelmakoodi esimerkiksi .Swift-tiedostoihin. Ohjelmakoodia voi testata käytännössä Xcoden sisäisellä iOS-simulaattorilla. iOS-käyttöjärjestelmän sovelluksen käyttöliittymän voi toteuttaa Xcoden sisäisellä SwiftUI:lla tai storyboards-näkymillä.

Natiivin iOS-sovelluksen kehittämiseen Biila siis tarvitsee Swift- tai Objective-C -kehittäjän, Apple Developer -jäsenyyden sekä Mac-tietokoneen ja Xcoden. Kun Biila omaa Apple Developer -jäsenyyden ja sovelluksen ohjelmakoodi on testattu sekä käännetty, voidaan sovellus lähettää Applen tarkastukseen. Applen seulan läpäisemiseksi sovellus ei saa sisältää bugeja tai se ei saa kaatua, joten testauksen tulee olla huolellista. Applen tarkastuksen ja mahdollisen beta-testauksen jälkeen sovellus julkaistaan App Storeen käyttäjien saataville.

6.3.2 Android

Toisin kuin Applen Xcode, Androidin kehitysalusta Android studio on saatavilla MacOS-käyttöjärjestelmän lisäksi Windowsille, Linuxille ja ChromeOS-järjestelmälle. Kehittäjä voi siis valita, millä käyttöjärjestelmällä hän haluaa sovelluksen kehittää. Android studiota käytetään sovelluksen koodin kääntämiseen, mutta myös sovelluksen koodin kirjoittaminen on suositeltavaa Android studion sisällä. Android studion sovellusemulaattorilla voidaan testata sovellusta käytännössä kesken kehityksen. Sovelluksen julkaisemiseen tarvitaan Google Play Developer -tili, jonka kautta sovellus voidaan ladata Google Play -kauppaan.

Android-sovellus ohjelmoidaan käyttämällä joko Javaa tai Kotlinia. Näistä Java on yleisemmin käytetty sekä Java on useammalle kehittäjälle tuttu kieli. Sovelluksen kehittäminen Android studiossa aloitetaan luomalla sille uusi projekti. Projektin luomisvaiheessa voidaan määrittää sen ominaisuudet, kuten tuki eri Android-versioille. Sovelluksen ohjelmakoodi kirjoitetaan tämän projektin sisälle. Sovelluksen käyttöliittymä voidaan luoda käyttämällä XML-pohjaisia tiedostoja. Sovelluksen ohjelmakoodin ja testauksen valmistuttua sovelluksen koodi käännetään Android studiolla APK-tiedostoksi (Android Package Kit), joka voidaan ladata Google Play -kauppaan. Jokaisella Android-sovelluksella tulee olla myös AnroidManifest.xml-tiedosto juurihakemistossaan, joka sisältää olennaiset tiedot Android-rakennustyökalulle. Kehityksessä voidaan käyttää hyödyksi Android studion tarjoamia kirjastoja sekä ohjelmointirajapintoja. Sovellus julkaistaan lataamalla APK-tiedosto Google Play Developer -kehittäjäkonsoliin. Julkaisun yhteydessä tulee lisätä sovellukseen tehdyt päivitykset ja muutokset tekstinä. Näin sovellus tulee käyttäjien saataville Google Play -kauppaan. Natiivia Android-sovellusta varten Biila tarvitsee siis Java- tai Kotlin -kehittäjän, tietokoneen sekä Android studion ja Google Play Developer -tilin.

7 Johtopäätökset

Tämän kandidaatintyön tavoitteena oli tuottaa selvitys Biila Solutions Oy:lle järkevän sovellustyypin valintaan liittyen. Työssä vertailtiin mahdollisia vaihtoehtoisia sovellustyyppejä nykyisen progressiivisen web-sovelluksen tilalle. Vertailtuja sovellustyyppejä olivat natiivisovellus, hybridisovellus sekä progressiivinen web-sovellus. Työssä tutustuttiin myös Biilan nykyiseen sovellukseen ja sen toimintaan. Lisäksi toteutettiin käyttäjäpalautekysely käyttäjien palautteen keräämiseksi nykyisestä sovelluksesta ja selvitettiin kiinnostusta mahdollisesti tulevaan natiivisovellukseen liittyen. Työssä myös pohdittiin lopuksi, minkälainen tuleva sovellus voisi olla ja mitä toimintoja se sisältää.

Työn tuloksena voidaan todeta, että natiivisovellus on Biilan kuljettajien käyttöön optimaalinen sovellusratkaisu. Natiivisovellus tarjoaa loistavaa suorituskykyä ja luotettavuutta, joka on käyttäjäpalautteiden perusteella tarpeen autonkuljettajan työssä. Tätä hyvää suorituskykyä ja luotettavuutta nykyinen sovellus ei palautteiden perusteella pysty tarjoamaan. Natiivin sovelluskehityksen avulla tulevasta sovelluksesta voidaan tehdä entistä käyttäjäystävällisempi laajempien toiminnallisuuksien ansiosta. Natiivisovelluksen olennaisena etuna onkin paremman käyttökokemuksen saavuttaminen, sillä natiivisovellus tarjoaa kullekin alustalle optimoidut sovellusrakenteet ja toiminnot. Työn perusteella voidaan myös todeta, että natiivina kehitetyn sovelluksen saavutettavuus tulee olemaan riittävä, sillä kaikki palautekyselyyn vastanneista käyttäjistä omistivat joko iOS- tai Android-laitteen, joille natiivisovellus on tarkoitus kehittää. Natiivisovelluksen haittapuolena ovat sen korkeat kehitys- ja ylläpitokulut. Nähdäkseni tämä kyseinen haittapuoli kuitenkin pystytään kompensoimaan sovelluksen toiminnan parantumisella.

Työn perusteella voidaan vetää muutamia olennaisia johtopäätöksiä. Ensinnäkin sovelluskehityksen sovellustyypin ratkaisua mietittäessä valinta ei ole lähes ikinä yksiselitteinen. Sovellustyypin valintaan vaikuttaa monta eri tekijää, kuten käytettävissä olevat resurssit, sovelluksen käyttötarkoitus, sovelluksen kohdekäyttäjät, tavoiteltu

käyttökokemus, toiminnallisuus, saavutettavuus ja niin edelleen. Tämä lista jatkuu ikuisuuksiin, joten voidaan todeta, että sovellustyypin valinta on hyvä tehdä aina tapauskohtaisesti. Voidaan myös todeta, että natiivisovellus on paras ratkaisu silloin, kun käytettävissä on riittävästi resursseja ja halutaan saavuttaa paras mahdollinen käyttökokemus. Todetaan myös, että progressiivinen web-sovellus toimii parhaiten tilanteissa, joissa halutaan saavuttaa kohtuullinen toiminta sekä hyvä saavutettavuus matalilla kustannuksilla. Lisäksi tehdään päätelmä, että hybridisovellus on hyvä ratkaisu, mikäli tähtäimessä on natiivisovelluksen ja web-sovelluksen välimuotoon sijoittuva toteutus. Johtopäätöksenä voidaan myös todeta, että mikäli tavoitteena on kehittää käyttäjäystävällinen ja käyttökokemusta optimoiva sovellusratkaisu, on käyttäjien palautteiden kuunteleminen äärimmäisen tärkeää. Näin voidaan saavuttaa paras mahdollinen käyttökokemus unohtamatta sovelluksen elinehtoa: käyttäjää.

Riikola (2021) toteaa, että sovellustyypin valinnassa ei ole olemassa yhtä ainoaa toimivaa ratkaisua. Myös Hietala (2021) on Riikolan kanssa yhtä mieltä siitä, että valinta sovelluskehityksessä on tehtävä tapauskohtaisesti. Tämä Riikolan ja Hietalan tutkimusten päätelmä on linjassa tämän tutkimuksen päätelmän kanssa, joten päätelmää voidaan pitää hyvin paikkansa pitävänä. Teknologianala ja sen myötä sovellustekniikka kehittyy jatkuvasti, joten lisätutkimus tästä aiheesta on aina ajankohtaista sekä tarpeellista alan kestävän ja optimaalisen kehityksen kannalta. Lisätutkimukselle voisi olla aihetta esimerkiksi uusien sovellustekniikoiden kehittämisen parissa, jotta saadaan tuotettua entistä saavutettavampia ja käyttäjäystävällisempiä sovellusratkaisuja.

Lähteet

Hietala J. (2021). Natiivisovellusten ja progressiivisten web-sovellusten erot ja yhtäläisyydet sovellustuotannon ja käyttäjän näkökulmista. Kandidaatintyö. Jyväskylän yliopisto. Noudettu 9.2.2023 osoitteesta https://jyx.jyu.fi/bitstream/handle/123456789/75673/URN%3aNBN%3afi%3ajyu-202105172946.pdf?sequence=1&isAllowed=y

Mansoor I. (2023). App Download Data. Artikkeli. BusinessofApps. Noudettu 10.2.2023 osoitteesta https://www.businessofapps.com/data/app-statistics/

Riikola O.P. (2022). Sovellustyypin valinta mobiilisovelluksia suunniteltaessa ja kehitettäessä. Kandidaatin työ. Jyväskylän yliopisto. Noudettu 20.2.2023 osoitteesta https://jyx.jyu.fi/bitstream/handle/123456789/81246/URN%3aNBN%3afi%3ajyu-202205242876.pdf?sequence=1&isAllowed=y

Bilberg D. (2018) comparing performance between react native and natively developed smartphone applications in swift. Kandidaatintyö. University of Skövde Noudettu 25.2.2023 osoitteesta http://www.diva-portal.org/smash/get/diva2:1215717/FULLTEXT01.pdf

McLachlan. P.J, Tom Buckley. (2020) Easy to build, monetize, and discover: List your web app on Google Play. Artikkeli. Chromium Blog. Noudettu 25.2 2023. https://blog.chromium.org/2020/12/pwa-play-billing-support.html

Hämäläinen T. (2022). Mobiilisovelluskehitys React Nativella. Insinööri opinnäytetyö. Metropolia Ammattikorkeakoulu. Noudettu 25.2.2023 osoitteesta https://www.theseus.fi/bitstream/handle/10024/746456/hamalainen_toni.pdf?sequence=2

Jobe W. (2013). Native vs. Mobile Web Apps. Artikkeli. Tukholman yliopisto. Noudettu 25.2.2023 osoitteesta https://www.researchgate.net/profile/WilliamJobe/publication/268153001_Native_Apps_Vs_Mobile_Web_Apps/links/546346b30cf2cb7e9da765c3/Native-Apps-Vs-Mobile-Web-Apps.pdf

Kaarakainen A. (2021). Alustariippumattomat sovelluskehykset Android-sovelluskehityksessä. Kandidaatintutkielma. Tampereen yliopisto. Noudettu 25.2.2023 osoitteesta https://trepo.tuni.fi/handle/10024/131594

Kaikkonen H. (2021). Natiivi-, web- ja hybridisovelluskehityksen vertailu sekä mobiilisovelluksen kehittäminen. Insinööri opinnäytetyö. KAMK. Noudettu 25.2.2023 osoitteesta https://www.theseus.fi/bitstream/handle/10024/500832/Harri%20Kaikkonen%20Opinn%C3%A4ytety%C3%B6_Final.pdf?sequence=2

Russell A. (2015). Progressive Web Apps: Escaping Tabs Without Losing Our Soul. Blogikirjoitus. Noudettu 5.4.2023 osoitteesta https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/

Sidana M. (2015). The Rise of Hybrid Apps. Blogikirjoitus. Medium. Noudettu 5.4.2023 osoitteesta https://medium.com/@Mandysidana/the-rise-of-hybrid-apps-f101a825991b

Griffith C. What is Hybrid Mobile App Development ?. Artikkeli. Ionic. Noudettu 5.4.2023 osoitteesta https://ionic.io/resources/articles/what-is-hybrid-app-development

Charland A. ja Leroux B. Mobile application development : web vs. native. Artikkeli. ACM Portal. Noudettu 6.4.2023 osoitteesta https://dl.acm.org/doi/fullHtml/10.1145/1941487.1941504

Inventionland. The History of Mobile Apps. Blogikirjoitus. Noudettu 6.4.2023 osoitteesta https://inventionland.com/blog/the-history-of-mobile-apps/

Kuula M. (2019). iOS-mobiilisovelluksen kehittäminen Swift-ohjelmointikielellä. AMK opinnäytetyö. Noudettu 10.5.2023 osoitteesta https://www.theseus.fi/bitstream/handle/10024/266982/Mikko_Kuula_Opinn%E4ytety%F6.pdf;jsessionid=93E27D72BB7B9711E67E737A10B87461?sequence=3

Peltoniemi M. (2021). Android-sovelluksen datapohjainen kehitys. Insinöörityö. Noudettu 11.5.2023 osoitteesta https://www.theseus.fi/bitstream/handle/10024/356148/Peltoniemi_Miska.pdf?sequence=2&isAllowed=y