Olli-Pekka Riikola

Sovellustyypin valinta mobiilisovelluksia suunniteltaessa ja kehitettäessä.

Tietotekniikan kandidaatintutkielma

23. toukokuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Olli-Pekka Riikola

Yhteystiedot: olli-pekka.v.riikola@student.jyu.fi

Ohjaaja: Timo Tiihonen

Työn nimi: Sovellustyypin valinta mobiilisovelluksia suunniteltaessa ja kehitettäessä.

Title in English: Choice of application type in design and development of mobile applications

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 24+0

Tiivistelmä: Sovelluksia voidaan nykyään kehittää monella eri teknologialla ja lähestymistavalla, ja sen myötä ilmenee tarve valita sopivin paradigma sekä parhaat teknologiat ao. projektin tarpeisiin. Sovellustyypin valintaa lähestytään käyttäjäkokemuksen, resurssien, jakelun ja ylläpidon näkökulmasta. Sovellustyypin ominaisuuksien osalta suurimman vaikuttimen luo sovelluksen kehittämisessä käytettävät teknologiat eli joko natiivit teknologiat tai monialustaiset web-teknologiat. Toistaiseksi varmatoimisin, mutta mahdollisesti kallein ratkaisu on natiivisovellus. Vastaavasti edullisin ja kevein ratkaisu näyttää olevan progressiivinen web-sovellus.

Avainsanat: mobiilisovellus, natiivi sovellus, PWA

Abstract: Applications can be developed with various technologies or with different ways of approach. And because of that there appears to be need to choose the most suitable paradigm and the best technologies to match an one's project needs. Choice of application type is approached from the perspective of UX, resources, distribution and maintenance. From the perspective of application type itself the most important factor is created by the technologies used during the application development process. It matters whether native technologies or web technologies are used in the project. For now the most reliable solution is a native application, but it might be the most expensive also. Respectively the most favourable and

the lightest solution seems to be a progressive web-application.

Keywords: mobile application, native application, PWA

Termiluettelo

Agile Ohjelmistokehityksen paradigma, joka pyrkii ketterään ja asiakas-/ihmiskeskeiseen ohjelmistojen kehittämistapaan.

Alustariippumaton sovellus (Engl. cross-platform application) on sovellus, joka on kehitetty yhdellä ja samalla koodikannalla (engl. codebase) ja on tarkoitettu toimimaan eri alustoilla.

Hybrid-sovellus Hybrid-sovellus hyödyntää web-teknologioita, jotka kapseloidaan natiivin sovelluksen sisään.

Natiivi sovellus Sovellus, joka on kehitetty juuri tietylle alustalle tai laitteelle sopivaksi ko. alustaan yhteensopivia teknologioita käyttäen.

PWA (Progressive Web Application) on web-sovellusteknologia, joka on tarkoittettu parantamaan sovellusten saatavuutta, luotettavuutta ja käyttöä. PWA:n tunnistaa sille asetetusta kriteeristöstä, ks. luku 2.2.1.

Responsiivinen verkkosivu Verkkosivu, jonka ulkoasu mukautuu näyttökoon mukaan sekä näyttää hyvältä ja toimii käyttäjäystävällisesti erikokoisilla näytöillä on ulkoasultaan responsiivinen.

Sovellustyyppi Kuvaa konkreettista lähestymistapaa, mitä sovellusta kehitettäessä on käytetty. (Esim. natiivi, web, hybrid)

Taulukot

Sisältö

1 Johdanto

Maailmassa käytetään vuosittain valtavia summia rahaa mobiilisovelluksiin ja käyttäjät lataavat sovelluksia käyttöönsä todella suurissa määrin. Sovelluskaupat kuten Google Play ja Apple Store eivät kuitenkaan itse ilmoita tarkkoja tilastotietoja siitä kuinka paljon sovelluksia markkinoilla tosiasiallisesti on, mutta mm. Google Playn markkinointiviestinnän mukaan käyttäjille on tarjolla miljoonia sovelluksia ja niitä väitetään olevan yksinomaan Google Playssa ladattavissa jopa yli 3 miljoonaa. Nykyisellä mobiilisovelluskentällä on myös paljon erilaisia mahdollisuuksia toteuttaa sovellus ja tästä johtuen sovelluksen kehittämistavan valinta voi olla haastavaa. Tyypillisesti mobiilisovelluksia on kehitetty käyttäen natiivia lähestymistapaa, mutta nyttemmin natiivien sovellusten rinnalla on alkanut näkyä myös erilaisia web-teknologioilla toteutettuja sovelluksia. Vaikka tarkkaa määrää erityyppisten sovellusten markkinaosuuksista ei voida sanoa, on selvää, että jokaisen sovellusprojektin alkuvaiheessa täytyy tietyin perustein päättää mikä sovellustyyppi sopii parhaiten käynnistyvään projektiin. Tässä tutkielmassa pyritään tunnistamaan sovellustyypin valintaan vaikuttavia tekijöitä.

Kiinnostavana yksityiskohtana tutkielmassa on web-teknologioihin perustuva progressiivinen web-sovellus (PWA), joka on Googlen vuonna 2015 esittelemä kuvaus modernista web-sovelluksesta. (MDN 2022). Mm. sosiaalisen median alusta Twitter lanseerasi vuonna 2017 progressiivisen web-sovelluksen nimeltä Twitter Lite, joka tuotti 65% kasvun käyttäjien lataamien sivujen määrässä yhden session aikana (engl. pages per session), 75% kasvun lähetettyjen tviittien osalta sekä 20% laskun välittömään poistumisprosenttiin (engl. bounce rate). (ks Love 2018, s. 6). Tuoreiden verkkoselainpäivitysten ansiosta tämäntyyppisten sovellusten kehittäminen on mahdollista ja siksi on syytä tarkastella lähemmin, mikä PWA-sovellusten tilanne on ja miltä mobiilisovelluskenttä nykyisen kirjallisuuden valossa näyttää. Toisaalta, kuten sanottua, on hyödyllistä tietää mitkä syyt puoltavat edelleen natiivien mobiilisovellusten kehittämistä tai mitkä tekijät puoltavat hybrid-sovellusten kehittämistä.

Katsaus lähtee liikkeelle luvusta 2 käsittelemällä sovellustyyppejä sekä niiden ominaispiirteitä. Luku 3 alalukuineen käsittelee sovelluskehityksessä vaikuttavia tekjiöitä ja luvussa 4 pyritään löytämään sopivimmat mahdolliset yhdistelmät sovelluskehityksessä vaikuttavien tekijöiden ja eri sovellustyyppien joukosta.

2 Sovellustyypit ja niiden keskeiset piirteet

Sovellusta kehitettäessä on välttämätöntä valita jokin lähestymistapa, millä sovellusta lähdetään rakentamaan. Lähestymistapa määrittää mm. millä ohjelmointikielellä sovellus voidaan kehittää ja mitä kompromisseja täytyy tehdä toiminnallisuuksien osalta. (ks. Tun 2014, luku 1.). Jäljempänä tekstissä sovelluskehityksen lähestymistavasta käytetään konkreettisempaa nimitystä sovellustyyppi. Tässä luvussa perehdytään tarkemmin eri sovellustyyppien piirteisiin, jotta voidaan myöhemmin hahmottaa kunkin sovellustyypin vahvuusalue. Seuraavaksi tarkastellaan päätyyppejä, joita ovat natiivit ja alustariippumattomat sovellukset.

2.1 Natiivit sovellukset

Tyypillisesti sovelluskentällä on nähty natiiveja sovelluksia, jotka kykenevät hyödyntämään täysimittaisesti alustansa ominaisuuksia käyttölupien puitteissa. Monipuolinen laitteiden kirjo ja erilaiset alustat tuottavat kuitenkin haasteen natiivisovellusten kehittämiselle. Tyypillisesti sovellus koodataan ainakin Androidille ja IOS:lle, mutta toisinaan myös Microsoftin Windows Phonelle ja näin ollen natiivin sovelluksen kehittämisessä täytyy käyttää useaa eri ohjelmointikieltä. (ks. Tun 2014, luku 3.). Vaikka natiivin sovelluksen kehittäminen vaatiikin laajaa osaamista omaavan kehittäjätiimin, niin sillä saavutetaan käyttäjille laadukkas käyttäjäkokemus ja laajat toiminnallisuudet sekä mahdollisuuksen mukaan tämä kaikki vielä useammalle alustalle. (ks. Tun 2014, taulukko 2. Ja I. 2010, luku 3.4).

Natiivi sovellus kykenee tyypillisesti hyödyntämään kaikkia sen järjestelmän ominaisuuksia, johon se on asennettu. Sen käyttöliittymäpuoli vetoaa käyttäjiin todennäköisesti melko hyvin, koska natiiveilla alustoilla on omat käyttöliittymäkirjastonsa ja sen myötä sovellukseen saadaan tuttu ko. alustalle ominainen ulkoasu ja tuntuma. (ks. I. 2010, Luku 3.1). Tämän lisäksi natiivit sovellukset ovat myös tehokkaita, joskin hieman vaikeita toteuttaa. (ks. El-Kassas ym. 2017, taulukko 4.). Jakelu hoidetaan lähes yksinomaan sovelluskaupoissa eli saavutettavuudessa natiivisovellus turvautuu erillisiin sovellusten jakeluportaaleihin, kuten Google Play tai Apple Store.

2.2 Alustariippumattomat sovellukset

Monialustasovelluksissa ajatuksena on, että yhteisestä koodikannasta saadaan tuotettua eri alustoille valmiit sovellukset ja näin vähennettyä kehittämiseen ja ylläpitoon panostettavia resursseja. El-Kassas ym. (2017, Taulukko 6.) mukaan monialustasovellusten kehityssuunnat ovat kuitenkin edelleen keskeneräisiä ja voi olla liian aikaista sanoa, mikä teknologia tuottaisi parhaan mahdollisen hyödyn ja säästön sovelluksen kehitysaikaisiin resursseihin. Lisäksi alustariippumaton sovelluskehitysmenetelmä pitää sisällään neljä eri lähestymistapaa, jotka ovat nykyäänkin käytössä: web, hybrid, ajonaikainen tulkkaus (engl. runtime interpret) ja ristiinkääntäminen (engl. cross-compile). (ks. Rahul Raj ja Tolety 2012, Luku 2.).

Käydään lyhyesti läpi pääpiirteet ajonaikaisesti tulkatuista ristiinkäännetyistä sovelluksista ja niihin liittyvistä teknologioista. Ajonaikaisesti tulkattu sovellus sisältää tulkin, joka tulkkaa sovellusta laitteelle abstraktiotason kautta. Abstraktiotasolta on yhteys laitteen natiiviin rajapintaan ja näin sovellus pääsee käsiksi laitteen toiminnallisuuksiin. (ks. Rahul Raj ja Tolety 2012, luku 4.). Ristiinkäännetyt (engl. cross-compiled) sovellukset puolestaan käännetään omasta koodikannastaan aidoiksi natiivikoodeiksi jokaisen tuettavan alustan mukaan ja toimintavarmuus jää tässä tapauksessa paljolti kääntäjäohjelman harteille. (ks. Rahul Raj ja Tolety 2012, luku 5. Sekä Xanthopoulos ja Xinogalos 2013, luku 3.4). Jäljelmäpänä on mielekkäämpää jättää ajonaikaisesti tulkatut ja ristiinkäännetyt sovellukset vähemmälle huomiolle ja keskittyä tarkastelemaan lähtökohtaisesti web- ja hybridsovelluksia, koska tyypillisesti monialustasovellukset ovat nimenomaan web-teknologioihin perustuvia sovelluksia. (ks. El-Kassas ym. 2017, taulukko 6).

2.2.1 Progressiivinen web-sovellus

Progressiivinen web-sovellus on nimensä mukaisesti web-sovellus, jonka on tarkoitus toimia joustavasti eri selainversioiden ominaisuudet huomioiden. Se eroaa tavallisesta web-sovelluksesta siinä, että tyypillisesti web-sovellus sisältää hyvin niukasti laitteen toiminnallisuuksia ja PWA puolestaan tuo niitä saataville. Vaikka tavallisiinkin web-sovelluksiin on pesiytynyt vahvasti eräänlainen "mobile-first" ajattelu, joka ohjaa tekemään web-sovelluksista responsiivisia sekä käyttäjäystävällisiä juuri mobiililaitteille, niitä ei siitä huolimatta voida

yleensä edes luokitella varsinaisiksi mobiilisovelluksiksi. Tavallinen web-sovellus on kuitenkin edullinen sekä helposti saavutettavissa ja sen kehittäminen on nopeaa sekä edullista, monialustasovelluksille tyypilliseen tapaan, yhteisen koodikannan ansiosta. Siksi sitä käytetään ja se voi soveltua pienempien projektien ensimmäiseksi toteutustavaksi toisaalta edullisuutensa vuoksi ja toisaalta siksi, että sovellus saataisiin lanseerattua mahdollisimman nopeasti. Lisäksi vaikuttaa siltä, että mobiilisovellusten käyttöikä vaikuttaa olevan melko lyhyt eli n. 1-5 vuotta - ensimmäinen sovellus ei siis tule olemaan ikuinen, sillä jossain vaiheessa sovellus korvataan uudella vastaamaan paremmin kasvanutta tarvetta, ja siinä vaiheessa voidaan miettiä myös sovellustyypin valintaa uudelleen. (ks. Theunissen ja Heesch 2016, Luku 3.3).

Tavallista web-sovellusta kyvykkäämmän PWA:n termin kehittäjänä pidetty Russell (2015) esittää, että on olemassa tietyt progressiiviselle web-sovellukselle kuuluvat ominaisuudet. Tämän voidaan ajatella olevan eräänlainen kriteeristö, johon kuuluu löydettävyys hakukoneilla, asennettavuus, linkitettävyys, offline-toiminnallisuus, asteittaiset toiminnallisuudet selaimen tuen mukaan, kyky näyttää ilmoituksia, responsiivisuus ja turvallisuus. (MDN 2022). Näistä offline toiminnallisuus on kenties merkittävin edistys, mitä PWA tuo web-sovellusten maailmaan. Vielä 2010-luvun alussa oltiin sitä mieltä, että natiivit sovellukset ovat väistämättä parempia offline-toimintojen osalta. White (2013) toteaa artikkelissaan web-sovellusten lakkaavan toimimasta internet-yhteyden puuttuessa, mutta nimenomaan tähän on tullut muutos uusimpien selainpäivitysten ansiosta. Niiden myötä tulleet service-worker avustimet antavat mahdollisuuden tehdä progressiivisia web-sovelluksia, jotka avautuvat välimuistista ja toimivat staattisen sisällön varassa huolimatta puuttuvasta internetyhteydestä. Juuri service-worker avustimien ansiosta PWA pystyy myös selaimen kautta päästä käsiksi laitteen ominaisuuksiin, mikä omalta osaltaan on osoitus PWA:n pätevyydestä. Se voi rajapintojen kautta käyttää esimerkiksi kameraa, äänilähtöä, bluetoothia ja useaa muuta laitteen ominaisuutta, mikä tekee siitä kyvykkään ja monipuolisen web-sovelluksen. (*The Current State of Progressive Web Apps* 2020). PWA-tekniikka on siis tehokas tapa toteuttaa käyttäjäystävällinen mobiilisovellus yhtä koodikantaa ja valmiita web-ohjelmointikehyksiä hyödyntäen.

2.2.2 Hybrid-sovellus

Hybrid-sovelluksilla pyritään myös hakemaan edullisempaa tulokulmaa mobiilisisovellusten kehittämiseen. Ideana on toteuttaa sovelluksen ydin tunnetuilla web-teknologioilla kuten JavaScript, HTML5 ja CSS3 ja kapseloida tämä natiivin sovelluksen sisälle. Lopulta valmista sovellusta ajetaan sovellukseen itseensä sulautetun natiivin verkkoselaimen sisällä. (Xanthopoulos ja Xinogalos 2013, ks.[luku 3.2). Tässä yhteydessä on syytä huomata, että Hybrid-sovellus ei ole täysi web-sovellus toisin kuin esim. PWA, vaan on ominaisuuksiensa puolesta lähempänä natiivia sovellusta.

Koska hybrid-sovellusta ajetaan sulautetun selaimen sisällä (esim. Apache Cordova), sillä ei ole heti suoranaisia mahdollisuuksia hyödyntää kaikkia laitteen ominaisuuksia kuten kameraa, bluetoothia jne. Ne saadaan kuitenkin käyttöön sulautettuun selaimeen asennettavien lisäosien avulla ja näin saadaan hybrid-sovellus kilpailukykyiseksi vaihtoehdoksi natiivin sovelluksen rinnalle. (Ionic 2019). Toisaalta hybrid-sovellus on natiivia sovellusta huomattavasti hitaampi johtuen siitä, että hybrid-kehys joutuu käyttämään enemmän laitteen resursseja ja sovelluksen latausaika voi venyä 40% pidemmäksi kuin vastaavan natiivin sovelluksen. (Biørn-Hansen ym. 2020)

2.3 Yhteenveto sovellustyypeistä

Tässä kappaleessa käydään lyhyesti läpi edellä esiteltyjen sovellustyyppien eroavaisuudet. Yhteenvedon avulla on helpompi hahmottaa onko joku em. sovellustyypeistä jollain alueella toista vahvempi. Aiempien lukukappaleiden perusteella näyttää siltä, että sovellustyypit eroavat toisistaan ainakin teknologioiltaan, kustannustehokkuudeltaan ja kyvykkyydeltään. Kerrataan nämä kohta kohdalta seuraavasti:

- Teknologiat
  - Natiivisovellus koodataan jokaiselle alustalle erikseen käyttäen alustan natiivia ohjelmointikieltä. Esim. Androidille Java tai Kotlin tai IOS:lle Objective-C. (Tun 2014)
  - Hybrid-sovellus ja PWA kehitetään web-teknologioita käyttäen sillä erolla, että

hybrid-sovellus kapseloidaan lopulta natiiviin konttiin ja PWA toimii sellaisenaan suoraan palvelimelta tai selaimen välimuistista.

- Kustannustehokkuus
  - Natiivin sovelluksen kehittämiseen voi mennä enemmän aikaa ja vaivaa, koska jokainen alusta tarvitsee oman natiivikoodin ja lisäksi ylläpitotyö täytyy tehdä jokaiselle tuetulle alustalle.
  - Hybrid ja PWA toteutukset on koodattu yhteen koodikantaan monialustasovelluksille tyypilliseen tapaan ja tämä mahdollisesti säästää resursseja niin kehityksen aikana kuin ylläpidossa.
- Kyvykkyys
  - Natiivi- ja hybrid-sovellukset ovat kyvykkäitä ja voivat käyttää kaikkia alustansa toiminnallisuuksia. Toteutustapa eroaa siinä, että natiivisovellus pääsee suoraan käsiksi toimintoihin, mutta hybrid-sovellus joutuu hyödyntämään plug-in kirjastoja tai natiivia rajapintaa esimerkiksi laitteen kameran käyttämiseen.
  - PWA on myös kyvykäs web-sovellusten mittapuulla ja pystyy tavallisesta web-sovelluksesta poiketen hyödyntämään laitteensa ominaisuuksia rajapintojen kautta. Toistaiseksi sen ei kuitenkaan ole mahdollista käyttää aivan kaikkia alustansa toiminnallisuuksia, missä se selvästi eroaa natiivi- tai hybrid-sovelluksesta.

3 Sovelluskehityksessä vaikuttavat tekijät

Sovelluksen suunnittelussa ja kehittämisessä on suuri määrä asioita, mitkä tulisi ottaa huomioon onnistuneen lopputuloksen saavuttamiseksi. Todennäköisesti ei ole olemassa yhtä oikeaa tapaa tehdä mobiilisovelluksia, koska erilaiset tekijät voivat vaikuttaa eri tavoilla projektista riippuen. Alla on kuitenkin esitetty lista tekijöistä, joiden voidaan sanoa vaikuttavan lähes jokaisessa projektissa.

- Tarve ja kohdealue
- Resurssit
- Käyttäjäkokemus
- Julkaisu
- Jakelu tai tarjoaminen
- Tuki ja asiakaspalvelu
- Ylläpito

Tehdään lyhyt katsaus kuhunkin listan kohtaan ja tarkastellaan sen tärkeyttä sovellusprojektin osana. Luonnollisesti, sovelluskehitys lähtee tarpeesta ja siksi tarve on hyvä tunnistaa markkinoilta. Projektiin lähdettäessä on hyvä tunnistaa myös kehitykseen panostettavien resurssien tarve, joka tässä kontekstissa tarkoittaisi käytännössä henkilöstöresursseja, osaamista sekä taloudellisia resursseja. Lisäksi on tarpeellista myös kiinnittää erityistä huomiota kehitteillä olevan sovelluksen käyttäjäkokemukseen. Suomalaiset Kangas ja Kinnunen (2005) totesivat jo ennen 2010-luvun kosketusnäyttöbuumia, että käyttäjäkeskeinen suunnittelu osaltaan varmistaa lopullisen tuotteen toimivuuden. Kun sovelluksesta on saatu toimiva versio, se voidaan julkaista jossain jakelukanavassa. (ks. Holzer ja Ondrus 2011, luku 3.2). Toisaalta web-sovellusta ei tarvitse erikseen jaella, vaan se on saavutettavissa suoraan omasta domainistaan. Collier (2011, s. 2) korostaa asiakaspalvelun merkistystä ja toteaa, että pidemmällä tähtäimellä yritys ei voi menestyä ilman asiakaspalvelua ja sen myötä saatavia lojaaleja asiakkaita. Jatkuva asiakastuki on siis tärkeää, mutta niin on myös jatkuva ylläpito ja Jie-Cherng Chen (2009) esittää sen olevan suurin yksittäinen kuluerä ohjelmistoprojekteissa. Siihen verrattuna mikä tahansa muu listan kohta on pelkkä jäävuoren huippu.

Seuraavaksi tarkastellaan em. elementtejä tarkemmin. Tähän tutkielmaan valitaan vain osa aiemmin listatuista tekijöistä ja täten lähestytään sovellustyypin valintaa rajatusta näkökulmasta. Tämä tehdään johtuen siitä, että esim. kohdealueen puuttuessa ei ole mielekästä tarkastella aihetta tarpeen ja kohdealueen näkökulmasta. Samoin julkaisu jää käsittelemättä, mutta toisaalta jakelu ja julkaisu ovat jonkin verran kytköksissä toisiinsa ja näin se tulee jollain tasolla huomioitua osana kokonaisuutta. Lisäksi pois rajataan vielä tuki ja asiakaspalvelu, jotka ylläpidon tavoin ovat luonteeltaan jatkuva kuluerä, mutta toisaalta eivät sovelluksen toiminnan kannalta ole täysin välttämättömiä asioita. Rajauksen ulkopuolelle jääneet tekijät voi olla syytä ottaa mukaan tarkasteluun siinä vaiheessa, kun on joku todellinen tarve ja tiedetään mihin sovellusta tullaan käyttämään. Seuraavissa alaluvuissa siis käsitellään hiukan laajemmin käyttäjäkokemusta, resursseja, jakelua ja ylläpitoa.

3.1 Käyttäjäkokemus

Nykyään on selvää, että etenkin mobiilisovellusten tapauksessa käyttäjä on ensimmäisen luokan kansalainen. (ks. Love 2018, s.13). Kaikki sovelluskehityksen paradigmat pyrkivät mahdollisimman hyvään käyttäjäkokemukseen ja sitä kautta mahdollisimman tyytyväisiin käyttäjiin ja korkeaan käyttöasteeseen. Sovellustyyppi kuitenkin vaikuttaa vääjäämättä käyttäjäkokemukseen konkreettisesti käyttöliittymän osalta ja mahdollisesti toiminnallisin kompromissein. Tässä tutkielmassa hyödynnetään periaatteita, joita Richardson, Campbell-Yeo ja Smit (2021) esittelee voitavan käyttää mobiilisovelluksen käyttäjäkokemuksen tarkasteluun. Periaatteita on kaikkiaan seitsemän (7) ja ne on listattu alla.

- Käytettävyys
- Hyödyllisyys
- Houkuttelevuus
- Löytyvyys
- Esteettömyys
- Vakuuttavuus
- Käyttöarvo

Tässä listatuista periaatteista ainakin kaksi (2) voidaan katsoa sellaisiksi, mitkä voisivat ohja-

ta sovellustyypin valintaa: käytettävyys ja löytyvyys. Tämän tutkielman puitteissa käytettävyyden (engl. usability) osalta tarkastellaan onko sovellus tehokas ja onko sen käyttö helposti opittavissa. Löytyvyyden (engl. findability) osalta huomio kiinnittyy sovelluksen sisäiseen rakenteeseen, lähinnä navigaatioon ja niin ikään visuaaliseen hierarkiaan eli se on vahvassa kytköksessä sovelluksen käyttöliittymäpuoleen. (Richardson, Campbell-Yeo ja Smit 2021, ks. luku 2.2)

3.2 Resurssit

Resurssien käyttöä voidaan tarkastella projektin hallinnan näkökulmasta. Projektin hallintaan sisältyy aina ihmisiä, resursseja, järjestelmiä ja tekniikoita, joiden avulla projektin on tarkoitus valmistua ajallaan ja sille varatussa budjetissa. Kuitenkin onnistuminen on pitkälti kiinni järkevästä resurssien käytöstä ja heikko hallinta tai huono resurssien käyttö voi johtaa henkilöstön vaihtumiseen, aikataulujen tai budjetin ylittymiseen tai ylimääräiseen vaivannäköön. (ks. Liu ja Horowitz 1989, luku 2.). Tässä tutkielmassa resurssien puitteissa yksi kiinnostava tekjiä on nimenomaan budjetti ja henkilöstöresurssit ja etenkin sovelluskehittäjien osaamistaso.

Resurssien käyttöä voi ohjata myös muut intressit, esim. tekniset trendit. Tyypillisesti projekteissa pyritään minimoimaan kustannuksia, ja yksi askel siihen suuntaan on avoimen lähdekoodin teknologioiden käyttö. Mobiilisovelluskentällä ollaan jo jonkin aikaa menty enemmän ja enemmän avoimen lähdekoodin suuntaan. (ks. Holzer ja Ondrus 2011, Luku 3.).

3.3 Jakelu

Tyypillisesti sovelluksia jaellaan sovelluskaupoissa, joista suurimmat ovat Google Play ja Apple Store. Niin natiivit kuin hybrid-sovellukset ja ylipäätään kaikki muut kuin web-sovellukset tarvitsevat jonkin alustan, mistä ne voidaan ladata ja asentaa laitteelle. Sovelluskaupoissa jakelemeinen voi kuitenkin osoittautua hieman kankeaksi, sillä esim. tallennustilan hallinta tai sovelluspäivitykset eivät välttämättä toteudu parhaalla mahdollisella tavalla. Ladattavat sovellukset vievät hiljalleen tilaa laitteelta ja sovelluspäivitykset voivat jäädä asentamatta - loppujen lopuksi ne ovat käyttäjän vastuulla, vaikka kehittäjä olisikin siirtänyt päivitetyn ver-

sion sovelluskauppaan ladattavaksi. (ks. Yang, Jia ja Wang 2015, luku 1.) Web-sovellukset puolestaan eivät vaadi asennusta, vaan niitä voidaan suoraan käyttää vierailemalla niiden sijaitsemassa verkko-osoitteessa. Poikkeuksen tekee PWA, jota on mahdollista käyttää suoraan verkosta tai julkaista myös sovelluskaupassa, ainakin Google Playssa. (Google 2022)

Jakelumenetelmällä on oma vaikutuksensa sovelluksen saavutettavuuteen. Googlen ja Applen sovelluskaupoilla on vahva jalansija mobiilisovellusmarkkinassa ja lähes aina ne löytyvät älypuhelimista valmiiksi asennettuna. Käyttäjät ovat siis tottuneet lataaman sovelluksia sovelluskaupasta ja yleisesti ajatellaan, että virallisten sovelluskauppojen välityksellä sovellusten asentaminen on turvallista. Perinteiset sovellusten jakelukanavat ovat kuitenkin hieman murroksessa, sillä monesti asiakas tutustuu yritykseen tai tuotteeseen ensin verkossa. Kun asiakkaalle tarjotaan mahdollisimman hyvä kokemus verkossa, on hyvät mahdolliset pitää asiakkaat lojaaleina ja tyytyväisinä. (ks. Bedi, Kaur ja Lal 2017, 474).

Tämän seurauksena on potentiaalina tuoda sovellus suoraan web-sivulle, jolloin käyttäjä pääsee siihen heti käsiksi sivulle löydettyään. Toisin sanoen web-sovellusta voi jaella verkossa riippumatta mistään erillisestä sovelluskaupasta ja sovellus on löydettävissä verkosta hakukoneella. Hakukoneoptimoinnin ja muun digimarkkinoinnin avulla sovelluksen tuottaja voi näin parantaa yhtä aikaa sekä verkkosivujensa että sovelluksensa näkyvyyttä verkossa ja se voi olla kannattavaa, koska online-ostajat käyttävät hakukonetta verkon selaamiseen etsiessään tietoa haluamastaan tuotteesta tai palvelusta. (ks. Sen 2005, s. 12) Jos siis tuotteeseen tai palveluun liittyy sovellus, ostaja pääsee siihen heti käsiksi - mikäli se jaellaan suoraan verkossa.

3.4 Ylläpito

Ylläpito muodostaa tyypillisesti suurimman osan kuluista koko sovelluksen elinkaaren aikana ja aiemmin esitetyissä tutkimuksissa on tultu siihen tulokseen, että ylläpito muodostaisi yli 60% projektin kuluista. (Jie-Cherng Chen 2009, ks. luku 1.). Toisaalta myös ohjelmointiprojektin läpivientitapa todennäköisesti vaikuttaa ylläpitotyön määrään kuluihin. Aiheesta on tehty tutkimusta ja löydetty uutta suuntaa esimerkiksi agileen perustuvien läpivientitapojen ylläpitokustannusten arviointiin. (Gradišnik, Beranič ja Karakatič 2020, ks. luku 5.).

Sovelluksen ylläpitoon kuuluu sovelluksen päivittäminen niin sisällön kuin turvallisuuden osalta, mutta myös käyttäjäkokemuksen parantaminen joko analysoinnin tai käyttäjäpalautteiden perusteella. (IEEE 2022).

Sovellusten ylläpidolla on tärkeä merkitys myös niiden tietoturvaan ja sen parantamiseen. Siinä missä jo julkaistuun sovellukseen tehdään muita yleisiä parannuksia ja uudistuksia, niin samoin tulisi myös paikkauksia ja päivityksiä sovelluksen tietoturvaan. Tähän myös vaaditaan henkilöstöltä riittävää kykyä, jotta päivitykset onnistuvat riittävällä tasolla. (ks. IEEE 2022, luvut 6.1.1 ja 6.1.4)

4 Sopivan sovellustyypin valinta

Sovellustyypeistä kerätyn tiedon ja tunnistettujen tekijöiden perusteella pitäisi pystyä muodostamaan käsitys, mikä sovellustyyppi palvelee mitäkin tekijää parhaiten. Tässä luvussa jokainen sovellustyyppi käydään vielä kertaalleen läpi ja tunnistetaan ko. sovellustyypin vahvuusalueet valittujen tekijöiden näkökulmasta. Loppujen lopuksi voi olla mahdotonta löytää täydellistä sovellustyyppiä mihinkään projektiin, mutta sen sijaan hyviä kompromisseja löytyy varmasti. Jokaisella sovellustyypillä on omat vahvuutensa ja heikkoutensa ja näin ollen valinta on siis väistämättä kompromissi. Alla esitettyyn taulukkoon on koottu tiivistetysti tiedot sovellustyyppien vahvuusalueista ja taulukkoa täydentää sitä seuraavien alalukujen sisältämät tekstikappaleet.

	Natiivi sovellus	Hybrid-sovellus	PWA
Käyttökokemus	Alustalle ominainen tuntuma ja ulkoasu. Mahdollisuus hyvin optimoituihin toimintoihin.	Tuntuma voi vaihdella sovelluksesta riippuen. Tyypillisesti muita hitaampi.	Tuntuma voi vaihdella alustasta ja sovelluksesta riippuen. Yhtenäinen toimintatapa kaikilla laitteilla. Toimii nopeasti välimuistista.
Resurssit	Eri alustoille tulee koodata oma sovellus. Kehitysympäristöt saattavat olla maksullisia.	Tuotetaan yhdestä koodikannasta. Perustuu webteknologioihin.	Tuotetaan yhdestä koodikannasta. Koodataan webteknologioilla.
Jakelu	Jaellaan sovelluskaupassa. Asennettava.	Jaellaan sovelluskaupassa. Asennettava.	Jaellaan omalla domainilla. Ei vaadi asennusta. Toimii suoraan selaimessa.
Ylläpito	Työläs ylläpitää, jos sovellus on tuotettu useammalle alustalle.	Edullinen ylläpitää yhden koodikannan ansiosta.	Edullinen ylläpitää. Päivitykset suoraan palvelimelle.

Taulukko 1. Sovellustyypit ja tekijät

4.1 Natiivisovellus - käyttäjäystävällinen ja kyvykäs

Natiivin sovelluksen vahvuudet ovat hyvän käyttäjäkokemuksen ja runsaan toiminnallisuuden tarjonnassa. Luvussa 2.1 todettiin, että natiivien käyttöliitymäkirjastojen ansiosta natiivisovellus tarjoaa aina tutun ja alustaansa sopivan käyttöliitymän mikä on helposti opittavissa ja parantaa näin käyttäjäkokemusta. Käyttäjäkokemusta parantaa myös sovelluksen nopea toiminta ja monipuoliset toiminnallisuudet.

Ylläpidon osalta natiivisovellus on työläämpi muihin verrattuna, sillä jokaiselle alustalle on koodattava oma toteutus ja tästä syystä päivitykset ja muut ylläpitotyöt joudutaan tehdä kaikille alustoille erikseen. Natiivisovelluksen kehittäminen vaatii myös laajempaa osaamista, sillä eri alustoilla käytetään eri ohjelmointikieliä. Myös jakelu on hieman työlästä natiivisovellusten osalta, sillä tyypillisesti natiivin sovelluksen jakelu hoidetaan suosituissa sovelluskaupoissa kuten Google Play Storessa tai Apple App Storessa sovelluskaupan toimiessa ikään kuin välittäjänä käyttäjän ja kehittäjän välillä. Jakelukanavien käyttäminen on lähes vättämätön osa natiivin sovelluksen jakelua, kuten Holzer ja Ondrus (2011, ks. luku 3.2) artikkelissaan toteavat.

4.2 PWA - saavutettava ja tehokas

PWA:n ehdoton valtti on saavutettavuus eli jakelun tehokkuus. Se on helposti jaettavissa linkkinä esim. sähköpostin joukossa tai hakutuloksena hakukoneessa. Klikkauksen jälkeen sovellus avautuu verkkosivuna ja se on heti käytettävissä tai asennettavissa laitteelle, mikä parantaa omalta osaltaan myös käyttäjäkokemusta. (ks. Love 2018, s.14-15). Edelleen Love (2018) toteaa nykyaikaisten verkkoselainten tarjoavan hyvän perustan sovellukselle, mutta lopulta laadukkaan käyttäjäkokemuksen luominen on itsensä koodarin käsissä.

Luvun 2.2.1 lopussa todetaan, että PWA toimii selaimen välimuistin kautta. Tämä nopeuttaa sovelluksen toimintaa parantaa sovelluksen käyttäjäkokemusta ja kelpoisuutta. Näiden lisäksi PWA on alustariippumattoma sovelluksena myös kustannustehokas ja yhden koodikannan ansiosta ylläpito vaatii yleensä pienemmän työmäärän kuin natiivin sovelluksen ylläpito. Tämän vertailu tietysti riippuu siitä, kuinka monelle alustalle natiivisovellus on julkaistu. Resursseja säästynee ylläpidon lisäksi myös kehittäjissä, sillä PWA voidaan kehittää

täysin tunnettujen web-teknologioiden varaan ja näin projektiin voidaan käyttää yksinomaan web-ohjelmointia osaavia koodareita.

Toisaalta, kuten Love (2018) esittää, laadukas käyttäjäkokemus jää pitkälti kehittäjän vastuulle ja PWA ei voi natiivin sovelluksen tavoin turvautua tuttuihin käyttöliittymäkirjastoihin, vaan toteutus on tehtävä hyviä periaatteita noudattaen ohjelmoijan kykyjen mukaan. Lisäksi PWA ei vielä toistaiseksi kykene käyttämään kaikkia alustansa ominaisuuksia selaimen kautta, mikä voi osaltaan huonontaa käyttäjäkokemusta. Näin ollen voidaan todeta, että PWA:n vahvuusalueet löytyvät saavutettavuudesta, kustannustehokkuudesta ja ylläpidosta.

4.3 Hybrid - edullisesti ylläpidettävä

Kuten todettiin luvussa 2.2, alustariippumattomat sovellukset pyrkivät konventionaalisesti resurssien säästöön niin kehityksen aikana kuin ylläpitovaiheessa. Hybrid-sovellus on vahvuusalueellaan juuri näiden kahden tekijän näkökulmasta katsottuna. Luvun 2.2.2 pohjalta nähdään, että hybrid-sovellukseen riittää pääosin web-teknologioiden hallinta ja sovellus saadaan tuotettua eri alustoille tehokkaasti yhdestä koodikannasta.

Kaksi seuraavaa tekijää eivät niinkään kuulu hybrid-sovelluksen vahvuuksiin. Sen käyttäjäkokemus voi nimittäin kärsiä ainakin natiiviin sovellukseen verrattaessa, sillä hybrid-sovellus saattaa kehyksestä johtuen latautua muita hitaammin eikä kaikkia toiminnallisuuksia välttämättä saada sujuvasti käyttöön. Lisäksi käyttöliittymärajapinta ei välttämättä tarjoa kaikkia viimeisimpiä mahdollisuuksia ja tämä voi vaikuttaa heikentävästi käyttäjäkokemukseen. (Tun 2014). Myös saavutettavuus on perinteistä tasoa eli hybrid-sovelluksia jaellaan sovelluskaupoissa, aivan kuten natiivisovelluksiakin.

5 Pohdinta

Ainoa, mikä varmasti voidaan sanoa on se, että sovellustyypin valinnassa ei ole olemassa yhtä ainoaa toimivaa ratkaisua. Edellisissä luvuissa esiteltyyn tietoon nojaten on sanottava, että sovellustyypin valinta vaati aina tapauskohtaista tarkastelua valittujen näkökulmien puitteissa. Voidaan kuitenkin muodostaa tiettyjä nyrkkisääntöjä päätöksenteon tueksi ja nopeuttaa sovelluskehitysprojektin käynnistymistä. Esimerkiksi painottamalla saavutettavaa ja edullista kombinaatiota voidaan melko vähällä tutkiskelulla päätyä PWA ratkaisuun. Vastaavasti haettaessa kyvykästä ja käytettävyydeltään hyvätasoista lopputulosta voidaan päätyä natiivin sovellustyypin valintaa - tai toisaalta tietyin toiminnallisin rajoituksin PWA:n valintaan.

Loppujen lopuksi mobiilisovelluksilla on melko lyhyt historia. Natiivisovellukset tomivat hyvin ja tarjoavat edelleen taatusti hyvän käyttäjäkokemuksen, mutta kehittyvät web-teknologiat ovat kiinnostava ja vakuuttava vaihtoehto natiiveille sovelluksille jo nykypäivänä. Jos tulevaisuudessa päästään selaimen kautta turvallisesti ja tehokkaasti kiinni kaikkiin alustan ominaisuuksiin, siinä vaiheessa saattaa mobiilisovelluskentällä alkaa kokonaan uusi aikakausi.

Lähteet

Bedi, Sarbjit Singh, Sukhwinder Kaur ja Amit Kumar Lal. 2017. “Understanding Web Experience and Perceived Web Enjoyment as Antecedents of Online Purchase Intention”. *Global Business Review* 18 (2): 465–477. doi:10.1177/0972150916668614.

Biørn-Hansen, Andreas, Rieger Christoph, Grønli Tor-Morten, Tim A. Majchrzak ja Gheorghita Ghinea. 2020. “An empirical investigation of performance overhead in cross-platform mobile development frameworks.” 25:2997–3040. doi:10.1007/s10664-020-09827-6.

Collier, Marsha. 2011. *The Ultimate Online Customer Service Guide : How to Connect with Your Customers to Sell More!* John Wiley / Sons, Incorporated.

Google. 2022. *List your Progressive Web App in Google Play.* Viitattu 2. huhtikuuta. https://chromeos.dev/en/publish/pwa-in-play.

Gradišnik, Mitja, Tina Beranič ja Sašo Karakatič. 2020. “Impact of Historical Software Metric Changes in Predicting Future Maintainability Trends in Open-Source Software Development” [kielellä English]. *Applied Sciences* 10 (13): 4624. doi:10.3390/app10134624.

Holzer, Adrian, ja Jan Ondrus. 2011. “Mobile application market: A developer’s perspective”. *Telematics and Informatics* 28 (1): 22–31. ISSN: 0736-5853. doi:10.1016/j.tele.2010.05.006.

I., Wasserman Anthony. 2010. “Software Engineering Issues for Mobile Application Development”. Teoksessa *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research,* 397–400. FoSER ’10. Santa Fe, New Mexico, USA: Association for Computing Machinery. ISBN: 9781450304276. doi:10.1145/1882362.1882443.

IEEE. 2022. “ISO/IEC/IEEE International Standard - Software engineering - Software life cycle processes - Maintenance”. *ISO/IEC/IEEE 14764:2022(E):* 1–46. doi:10.1109/IEEESTD.2022.9690131.

Ionic. 2019. *Ionic Article: What is Hybrid App Development?* Viitattu 2. huhtikuuta 2022. https://ionic.io/resources/articles/what-is-hybrid-app-development.

Jie-Cherng Chen, Sun-Jen Huang. 2009. "An empirical analysis of the impact of software development problem factors on software maintainability". *Journal of Systems and Software* Volume 82, Issue 6 (6). doi:10.1016/j.jss.2008.12.036.

Kangas, Eeva, ja Timo Kinnunen. 2005. "Applying User-Centered Design to Mobile Application Development." *Communications of the ACM* 48 (7): 55–59. ISSN: 00010782. doi:10.1145/1070838.1070866.

El-Kassas, Wafaa S., Bassem A. Abdullah, Ahmed H. Yousef ja Ayman M. Wahba. 2017. "Taxonomy of Cross-Platform Mobile Applications Development Approaches". *Ain Shams Engineering Journal* 8 (2): 163–190. ISSN: 2090-4479. doi:10.1016/j.asej.2015.08.004.

Liu, Lung-chun, ja E. Horowitz. 1989. "A Formal Model for Software Project Management". *IEEE Transactions on Software Engineering* 15 (10): 1280–1293. doi:10.1109/TSE.1989.559781.

Love, Chris. 2018. *Progressive Web Application Development by Example : Develop Fast, Reliable, and Engaging User Experiences for the Web.* ProQuest Ebook Central.

MDN. 2022. *Introduction to progressive web apps.* Viitattu 7. maaliskuuta 2022. https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction.

Rahul Raj, C.P, ja Seshu Babu Tolety. 2012. "A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach". Teoksessa *2012 Annual IEEE India Conference (INDICON),* 625–629. doi:10.1109/INDCON.2012.6420693.

Richardson, Brianna, Marsha Campbell-Yeo ja Michael Smit. 2021. "Mobile Application User Experience Checklist: A Tool to Assess Attention to Core UX Principles". *International Journal of Human–Computer Interaction* 37 (13): 1283–1290. doi:10.1080/10447318.2021.1876361.

Russell, Alex. 2015. *Progressive Web Apps: Escaping Tabs Without Losing Our Soul - Infrequently Noted,* kesäkuu. Viitattu 6. huhtikuuta 2022. https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/.

Sen, Ravi. 2005. "Optimal Search Engine Marketing Strategy". *International Journal of Electronic Commerce* 10 (1): 9–25. doi:10.1080/10864415.2005.11043964.

The Current State of Progressive Web Apps. 2020. Viitattu 2. huhtikuuta 2022. https://www2.stardust-testing.com/en/the-current-state-of-progressive-web-apps.

Theunissen, Theo, ja Uwe van Heesch. 2016. "The Disappearance of Technical Specifications in Web and Mobile Applications". Teoksessa *Software Architecture,* toimittanut Bedir Tekinerdogan, Uwe Zdun ja Ali Babar, 265–273. Cham: Springer International Publishing. ISBN: 978-3-319-48992-6. doi:10.1007/978-3-319-48992-6_20.

Tun, Phyo Min. 2014. "Choosing a mobile application development approach". *ASEAN Journal of Management and Innovation* 1 (1): 69–74. doi:10.14456/ajmi.2014.4.

White, James. 2013. *Going native (or not): Five questions to ask mobile application developers - PMC.* Viitattu 30. maaliskuuta 2022. doi:10.4066/AMJ.2013.1576. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3575060/.

Xanthopoulos, Spyros, ja Stelios Xinogalos. 2013. "A Comparative Analysis of Cross-Platform Development Approaches for Mobile Applications". Teoksessa *Proceedings of the 6th Balkan Conference in Informatics,* 213–220. BCI '13. Thessaloniki, Greece: Association for Computing Machinery. ISBN: 9781450318518. doi:10.1145/2490257.2490292.

Yang, Wang, Lei Jia ja Guojun Wang. 2015. "Application Streaming: A Mobile Application Distribution Method". Teoksessa *Algorithms and Architectures for Parallel Processing,* toimittanut Guojun Wang, Albert Zomaya, Gregorio Martinez ja Kenli Li, 303–316. Cham: Springer International Publishing. ISBN: 978-3-319-27119-4.