TAMPEREEN
AMMATTIKORKEAKOULU

Monialustainen mobiilikehitys Xamarin-alustalla

Pyry Nikunen

Opinnäytetyö
Marraskuu 2016
Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

NIKUNEN, PYRY:
Monialustainen mobiilikehitys Xamarin-alustalla

Opinnäytetyö 40 sivua, joista liitteitä 4 sivua
Marraskuu 2016

Opinnäytetyön tarkoituksena oli toteuttaa työn tilaajalle Evolvit Oy:lle tuntikirjaussovellus Android ja iOS-alustoille helpottamaan työntekijöiden arkea. Tavoitteena oli tutkia Xamarin-alustan soveltuvuutta monialustaiseen mobiilikehitykseen ja laajentaa yrityksen osaamista mobiilikehityksen parissa.

Opinnäytetyössä tutkittiin ja vertailtiin erilaisia mobiiliohjelmointikehyksiä sekä selvitettiin niiden soveltuvuutta yrityksen käyttöön. Työssä kuvattiin Xamarin-alustan valintaan johtanut prosessi ja perustelut. Valinnan lisäksi työssä tutkittiin tarkemmin Xamarin-alustan toimintaa Android ja iOS-alustoilla sekä kuvattiin Xamarin-kehityksen mahdollisuuksia verrattuna natiiviin mobiilikehitykseen.

Ohjelmointikehyksen valinta on hankalaa ja vaihtoehtoja on hyvin paljon. Kehyksien perusteellinen vertaileminen ja rajoitteiden kartoitus tekivät valinnasta helpompaa sekä pienensivät valinnasta aiheutuvaa riskiä. Vertailu myös loi selkeämmän kuvan kehyksien käyttötarkoituksista ja mahdollisti oikean kehyksen valinnan. Lisäksi työn aikana selvisi, ettei kehyksen käyttö välttämättä ainakaan ensimmäisellä kerralla nopeuta tai helpota sovelluksen kehitystä. Kehyksen käyttö kuitenkin lisää ylläpidettävyyttä ja helpottaa jatkokehitystä.

Asiasanat: xamarin, mobiili, alustariippumaton, monialustainen

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Software Development

NIKUNEN, PYRY:
Cross-Platform Mobile Development with Xamarin

Bachelor's thesis 40 pages, appendices 4 pages
November 2016

The purpose of this thesis was to create a time tracking application for iOS and Android for the commissioner of this thesis. The goal of this thesis was to research the possibilities and suitability of Xamarin platform for cross-platform mobile development and to broaden out the know-how of mobile development within the company.

Different cross-platform frameworks compared and researched in this thesis to study their suitability for the company. The process and the reasons which led to the choosing of the Xamarin platform are described. Moreover, a discussion is provided on the working principles of Xamarin Android and Xamarin iOS, and Xamarin development is compared to native mobile development.

Picking up the best framework for the job is a hard task as there are so many frameworks available. That is why comparing the framework and mapping out the issues and limitations made the choosing process easier and lowered the risk of a bad decision. The comparison also clarified the purposes of the different frameworks and made it possible to choose the right one for the job. Also during the development of the time tracking application, it became clear that a framework might not make the development easier or faster, at least upon first use. Using a framework will in any case make it easier to maintain the application and makes the development easier in the future.

Key words: xamarin, mobile, platform-independent, cross-platform

SISÄLLYS

1 JOHDANTO 6
2 MONIALUSTAINEN MOBIILIKEHITYS 8
2.1 Kehyksien esittely 8
2.1.1 Apache Cordova 8
2.1.2 React Native 10
2.1.3 Qt 11
2.1.4 Xamarin 11
2.2 Kehyksen valinta 12
3 XAMARIN-KEHYS 15
3.1 Xamarin Android 15
3.2 Xamarin iOS 17
3.3 Jaettava ohjelmakoodi 18
3.4 Arkkitehtuuri 20
3.4.1 iOS 20
3.4.2 Android 21
3.4.3 MVVM-malli 22
3.5 Linker 22
3.6 Yhteenveto 23
4 TUNTIKIRJAUSSOVELLUS 27
4.1 Kehitysympäristö 27
4.2 Arkkitehtuuri 28
4.3 Ohjelmakoodin jakaminen useille alustoille 28
4.4 Alustakohtaisten komponenttien kehitys 30
4.5 Lopputulos 32
5 YHTEENVETO JA LOPPUTULOKSET 33
LÄHTEET 35
LIITTEET 37

ERITYISSANASTO

.NET	.NET on Microsoftin kehittämä kehitysalusta, jolla voidaan toteuttaa sovelluksia useilla eri ohjelmointikielillä.
AOT	Ahead-of-Time on .NET kääntäjä, joka kääntää ohjelmakoodin kokonaisuudessaan konekielelle.
DLL	Dynamic-link library on Microsoftin toteutus jaetuista kirjastoista
ECMA	Ecma International on kansainvälinen ICT-alan standardointiorganisaatio
JIT	Just-in-Time on .NET kääntäjä, joka kääntää välikielelle käännettyä ohjelmakoodia konekieleksi ajon aikana.
MSDN	Microsoft Software Developer Network on Microsoftin kehittäjäsivusto, joka sisältää .NET dokumentaation.

1 JOHDANTO

Opinnäytetyön tilaajana toimii IT-palveluyritys Evolvit Oy, joka keskittyy erityisesti finanssitoimialaan ja Business Intelligence -ratkaisuihin. Mobiililaitteiden käytön yleistyminen on lisännyt mobiilikehityshankkeiden määrää myös finanssitoimialalla ja tämän seurauksena Evolvitilla on tarve laajentaa ja kehittää osaamista myös mobiiliohjelmoinnin parissa. Työn aikana syntyy tuntikirjaussovellukset iOS ja Android-alustoille, jotka toimivat yrityksen olemassa olevan tuntikirjausjärjestelmän kanssa. Sovellusten tarkoitus on helpottaa työntekijöiden elämää ja karsia pois ylimääräisiä prosesseja. Sovellusten tarkoituksena on luoda pohjaa Evolvitin mobiilikehitykselle ja toimia referenssisovelluksina mobiilihankkeiden myyntiä varten ja näin ollen mahdollistaa yrityksen kilpailu mobiilimarkkinoilla. Mobiilihankkeilla on mahdollista saada hyvin paljon näkyvyyttä ja näkyvyydellä haetaan lisääntyvää myyntiä myös muilla sovelluskehitysmarkkinoilla.

Työssä pohditaan monialustaisen mobiilikehityksen mahdollisuuksia, rajoitteita ja eroja natiivien sovellusten kehitykseen verrattuna. Natiivien sovellusten kehityksellä tarkoitetaan tässä yhteydessä Java-ohjelmointikielellä toteutettavaa Android-kehitystä sekä Objective-C tai Swift-ohjelmointikielellä toteutettavaa iOS-kehitystä. Natiiveja sovelluksia voidaan toteuttaa myös C ja C++ -ohjelmointikielillä, mutta tämä työ ei ota näihin mahdollisuuksiin kantaa, eikä sen enempää esittele näiden kielien mahdollisuuksia. Työssä esitellään erilaisia toteutuksia monialustaisesta mobiilikehityksestä ja verrataan näiden toteutuksien suorituskykyä ja toimintaa sekä toisiinsa että natiivien sovellusten kehittämiseen ja näin pyritään tuomaan esille eri kehysten hyvät ja huonot puolet sekä soveltuvuus Evolvitin käyttöön.

Nyt voidaankin kysyä, miksi monialustatoteutus eikä natiivia. Tähän lienee paikallaan todeta, ettei Evolvitilta löydy juuri ollenkaan aiempaa osaamista mobiilikehityksestä. Monialustaisessa kehityksessä riittää yhden ohjelmointikielen hallitseminen, joten opettelu on nopeampaa ja helpompaa. Lisäksi monialustatoteutus on helpompi ylläpitää ja pitää yhtenäisenä kuin erillisiä sovelluksia. Helppoudella on kuitenkin kääntöpuolensa ja näitä ongelmia pyritään selvittämään tässä työssä ja mahdollisuuksien mukaan tuoda ongelmiin myös soveltuvat ratkaisut.

Työssä käydään läpi kehyksen valintaan johtanut prosessi ja perusteet, jotka johtivat Xamarin-alustan valintaan. Lisäksi työ kuvaa ja selvittää monialustaisen mobiilikehityksen mahdollisuuksia Xamarin-alustalla. Työssä keskitytään erityisesti Xamarin iOS ja Xamarin Android -kehyksiin, jotka mahdollistavat Android ja iOS-kehityksen käyttäen C#-kieltä ja jaettua ohjelmakoodia. Syntyvien sovellusten on tarkoitus näyttää esimerkkiä Xamarin-alustan mahdollisuuksista sekä haasteista ja tavoitteena on syventää tietotaitoa mobiilikehityksen parissa.

2 MONIALUSTAINEN MOBIILIKEHITYS

Monialustainen toteutus (engl. cross-platform) viittaa sovelluksen kykyyn toimia useammalla kuin yhdellä alustalla identtisillä tai lähes identtisillä ominaisuuksilla (The Linux Information Project, 2005). Tämän työn yhteydessä monialustaisuudella tarkoitetaan ohjelman kykyä toimia Android ja iOS-käyttöjärjestelmillä. Monialustaisella mobiilikehityksellä tässä tapauksessa viitataan jonkin kehyksen tarjoamaan mahdollisuuteen käyttää osittain tai täysin samaa ohjelmakoodia edellä mainituilla käyttöjärjestelmillä.

Mobiilikehitykseen on tarjolla useita erilaisia ohjelmointikehyksiä, joilla voidaan toteuttaa monialustaisia sovelluksia. Näillä kehyksillä sovelluksia voidaan toteuttaa esimerkiksi Java, JavaScript, C# ja C++ -ohjelmointikielillä. Koska kehyksiä on tarjolla useita, muodostuu kehyksen valinnasta merkittävä tekijä sovelluksen kehityksen kannalta. Kehyksien toteutustapa nimittäin määrittää hyvin paljon mm. kehyksen suorituskykyä sekä natiivia tyyliä ja tuntumaa (engl. Look and Feel).

2.1 Kehyksien esittely

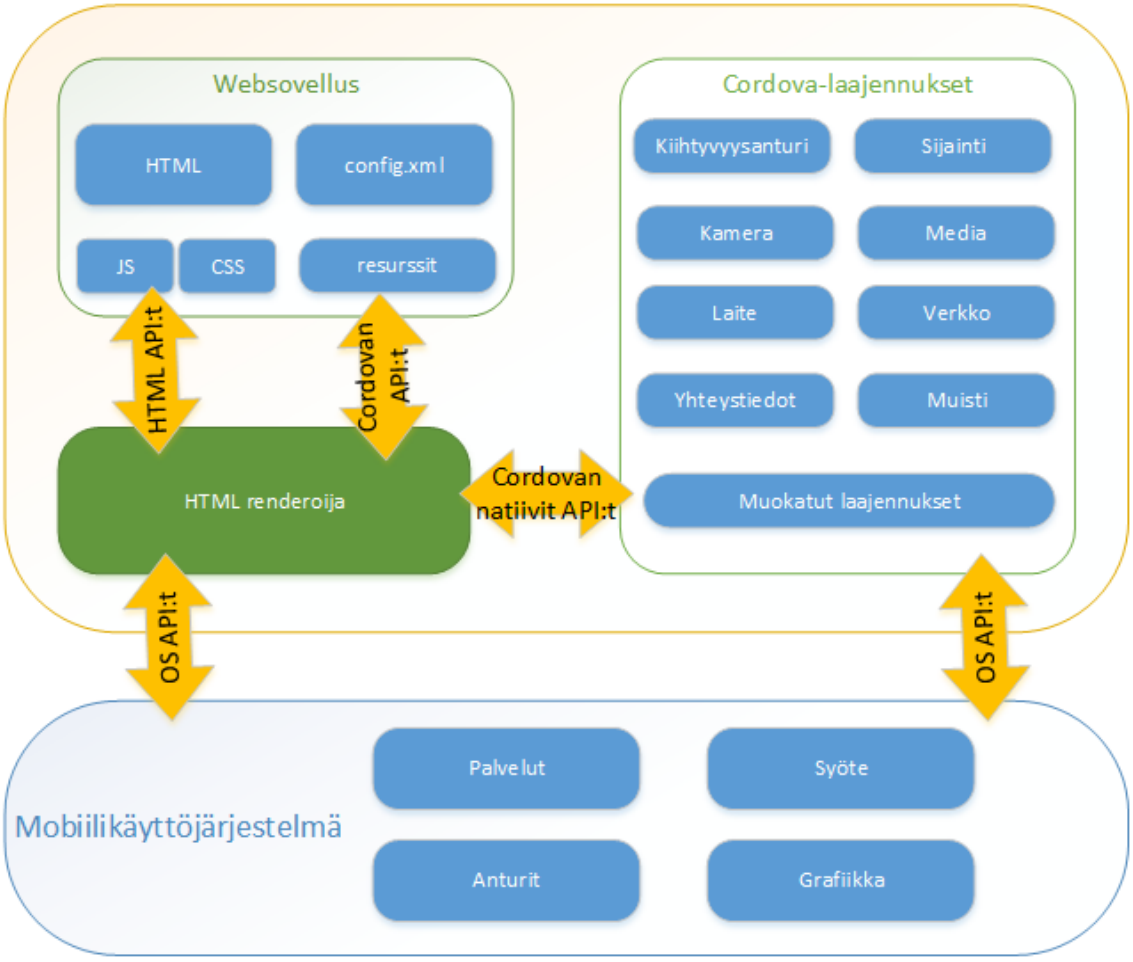
Monialustaiseen mobiilikehitykseen on nykypäivänä käytössä suuri määrä erilaisia kehyksiä, joilla on mahdollista toteuttaa Android ja iOS-sovelluksia. Koska kehyksiä on erittäin paljon, niin tässä työssä ei ole realistista kuvata kaikkia kehyksiä. Työhön onkin valittu kehyksiä, joilla on erilaiset lähestymistavat monialustaisuuden toteutukseen.

Kehystä valittaessa etsittiin paljon tietoa erilaisista vaihtoehdoista ja vertailtiin kehysten hintoja ja käyttötarkoituksia. Kehyksiä päätettiin vertailla kehyksen suorituskyvyn, natiivin tyylin ja tuntuman, ohjelmointikielien sekä kulujen perusteella. Käsiteltäviksi kehyksiksi päätyivät Apache Cordova, React Native, Qt ja Xamarin.

2.1.1 Apache Cordova

Apache Cordova on avoimen lähdekoodin mobiiliohjelmointikehys, joka mahdollistaa mobiilisovellusten kehityksen käyttäen HTML5, CSS3 ja JavaScript -kieliä. Sovellukset

suoritetaan alustakohtaisessa paketissa (kuvio 1), joka mahdollistaa pääsyn laitteen ohjelmointirajapintoihin, mahdollistaen muun muassa laitteen sensoritietojen ja kameran käytön. (The Apache Software Foundation, 2015.)



Kuvio 1. Apache Cordova sovelluksen toiminta (The Apache Software Foundation, 2015)

Cordovan päälle on myös rakennettu useita erilaisia kehyksiä helpottamaan kehitystä. Kehyksistä tunnetuin on Adoben PhoneGap, joka on Cordovan tuotteistettu versio, jonka pohjalta avoimen lähdekoodin versiokin on julkaistu. Muita Cordovan päälle rakennettuja kehityksiä ovat mm. Ionic, joka keskittyy käyttöliittymäkehityksen helpottamiseen ja Telerik Platform, joka tarjoaa yritystason alustan sovellusten kehittämiseen.

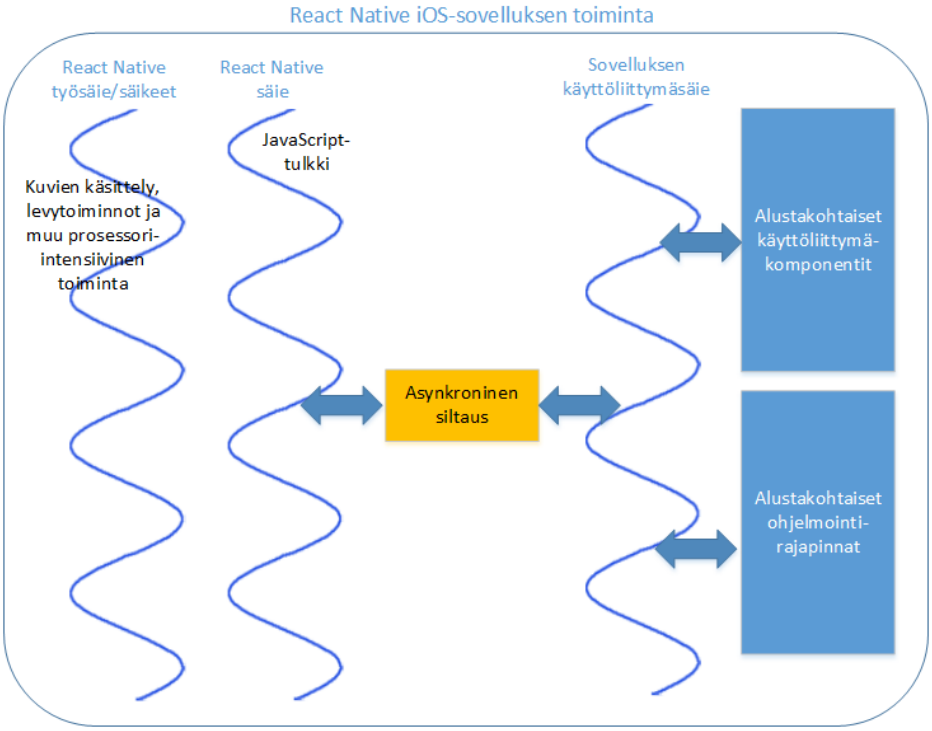
Cordova tarjoaa helpon ja tehokkaan tavan kehittää sovelluksia useille alustoille käyttäen lähes samaa ohjelmakoodia jokaisella alustalla. Tämä kuitenkin tulee sillä hinnalla, että sovelluksista on hankala saada natiivin vastineen näköisiä ja tuntuisia. Myöskin suoritus-

kyvyssä Cordova-pohjaiset sovellukset häviävät kilpaileville monialustaratkaisuille. Cordova onkin hyvä alusta yksinkertaisten esimerkiksi tietojen syöttöön tarkoitettujen sovellusten kehitykseen.

2.1.2 React Native

React Native on JavaScript-kehys natiivien mobiilisovellusten kehittämiseen iOS ja Android -käyttöjärjestelmille. Se pohjautuu Facebookin React JavaScript-kirjastoon, joka on luotu käyttöliittymien toteutukseen (Eisenman, 2015). React Native käyttää natiiveja käyttöliittymäkomponentteja eikä perustu mobiilikäyttöjärjestelmien WebView-komponentin käyttöön. Tämä Cordovasta poikkeava toteutus mahdollistaa natiivin tyylin ja tuntuman sekä korkeamman suorituskyvyn. Natiivien käyttöliittymäkomponenttien käyttö kuitenkin tarkoittaa sitä, että tarvitaan myös natiivin kehityksen tuntemista, sillä komponenttien räätälöinti tapahtuu alustakohtaisesti joko Java tai Objective-C -ohjelmointikielellä.

React Nativen toiminta perustuu JavaScript-ohjelmakoodin ja alustakohtaiseen ohjelmakoodiin siltaukseen (kuvio 2), jolloin pystytään käyttämään alustakohtaisia komponentteja ja ohjelmointirajapintoja. JavaScript-koodia ajetaan omassa säikeessä, jolloin raskaammatkaan operaatiot eivät hidasta käyttöliittymää ja käyttökokemus pysyy mukavana ja sujuvana.



Kuvio 2. React Native-sovelluksen toiminta (Facebook, 2016)

React Nativen ongelmana kuitenkin on kehyksen uutuus ja kypsymättömyys. Kokeillessa kehyksen käyttöä, törmättiin useisiin kehitystä hankaloittaviin tekijöihin sovellusten kääntämisten ja ajamisten kanssa. Lisäksi Facebook on ulkoistanut Windows ja Linux-käyttöjärjestelmille tehtävän kehityksen täysin yhteisön varaan, joten kaikkia työkaluja ei vielä ole saatavilla näille käyttöjärjestelmille.

2.1.3 Qt

Qt on alun perin jo 1990-luvulla kehitetty ohjelmointikehys työpöytä- ja mobiilisovelluksille sekä sulautettuihin järjestelmiin. Tuettuja alustoja ovat muun muassa Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry ja Sailfish OS. Qt laajentaa C++-ohjelmointikieltä omilla ominaisuuksillaan, jotka Qt:n Meta-Olio kääntäjä (engl. Meta-Object Compiler, MOC) kääntää standardiksi C++-koodiksi. (The Qt Company 2016.) Näin ollen Qt:lla totutetut mobiilisovellukset toimivat täysin natiiveina sovelluksina.

Vaikka Qt-sovellukset kääntyvätkin natiiviksi koodiksi, on Qt-sovellusten toiminta kuitenkin alustariippuvaista. Qt:lla toteutetut Android-sovellukset käyttävät natiiveja Java-ohjelmointirajapintoja, joiden käyttö mahdollistaa sovellusten toiminnan Android-alustalla (Blomfelt 2013). iOS-alustalle sovellukset taas käännetään Clang-kääntäjällä, joka mahdollistaa C++ ja Objective-C -ohjelmakoodin sekoituksen (The Qt Company 2016).

Qt on avoimen lähdekoodin projekti, mutta The Qt Company myy Qt:n kaupallisia lisenssejä ja johtaa Qt:n kehitystä. Qt:ta on mahdollista siis käyttää maksutta, mutta tällöin toteutetut sovellukset tulee myös julkaista avoimena lähdekoodina ja LGPL- tai GPL-lisenssillä, jota yritykset eivät yleensä halua tehdä.

2.1.4 Xamarin

Xamarin on nykyään Microsoftin omistama Mono-projektiin pohjautuva kehys, joka mahdollistaa kehityksen Android, iOS, Mac ja Windows -alustoille. Mono taas on monialustatoteutus Microsoftin .NET kehyksestä, joka mahdollistaa .NET kehityksen muillakin kuin Windows-alustalla.

Myös Xamarinin toiminta on täysin alustariippuvaista, vaikkakin sama ohjelmakoodi toimii kaikilla alustoilla. Android-alustalla Xamarinin toiminta perustuu ajonaikaiseen kääntämiseen (engl. Just-In-Time, JIT) ja Xamarin-ajoympäristöä ajetaan rinnakkain Java-virtuaalikoneen kanssa. iOS-alustalla JIT kääntäminen ei ole sallittua, joten Mono on kehittänyt erilaisen kääntäjän, joka kääntää koodin ennenaikaisesti (engl. Ahead-Of-Time, AOT), joka mahdollista Xamarinin toimimisen myös iOS-alustalla. Windows Phone -alustalle kehitetään muutenkin C#-kielellä ja näin ollen Xamarin-sovellukset toimivat täysin natiiveina Windows Phonella.

Mobiilikehitys Xamarinilla jäljittelee natiivia kehitystä ja näin ollen lähes ainoa eroavaisuus on käytössä oleva C#-kieli. Käyttöliittymiä voidaan toteuttaa natiivikehitykseen tarkoitetuilla työkalulla tai Xamarinin tarjoamilla vastineilla.

2.2 Kehyksen valinta

Kehyksen valinta suoritettiin keväällä 2016, kun yrityksen sisällä tuli pyyntö pitää esittely varteenotettavista mobiiliohjelmointikehyksistä. Tätä esittelyä varten selvitettiin soveltuvat vaihtoehdot, valittiin niistä parhaat ja toteutettiin pienet demot kaikista. Tähän esittelyyn valikoituivat yllä esitellyt Apache Cordova, React Native, Qt ja Xamarin. Nämä vaihtoehdot valittiin perustuen niiden soveltuvuudesta yritystason hyötysovelluksiin ja koska yrityksen ei ole tarkoitus tehdä pelejä jätettiin pelien kehitykseen suuntautuneet kehykset suosiolla pois. Kehyksen valinnassa käytetyt mittarit olivat jo aiemmin mainitut suorituskyky, natiivi tyyli ja tuntuma, ohjelmointikieli sekä kulut (taulukko 1).

TAULUKKO 1. Kehyksien vertailu.

	Apache Cordova	Qt	React Native	Xamarin
Natiivi tyyli ja tuntuma	Ei	Ei	Kyllä	Kyllä
Suorituskyky	Web	Natiivi	Lähes natiivi	Natiivi
Ohjelmointikieli	JavaScript	C++	JavaScript	C#
Hinta	-	350$ / kk / kehittäjä	-	499$ / lisenssi

Sovellusten suorituskyky muodostuu oleelliseksi asiaksi edullisemmissa ja vanhemmissa puhelimissa sekä suuria datamääriä käsittelevissä sovelluksissa. Tavoitteena oli löytää

kehys, jonka suorituskyky olisi mahdollisimman lähellä natiivien sovellusten suorituskykyä. Vertailluista kehyksistä Apache Cordovan suorituskyky on kauimpana natiivista, koska Cordova-sovelluksia suoritetaan WebView-komponentissa ja kaikki logiikka toimii tällöin käyttöliittymäsäikeessä. React Native ajaa omaa logiikkaansa erillisessä JavaScript-säikeessä ja pääsee näin hyvin lähelle natiivien sovellusten suorituskykyä. Qt taas kääntyy natiiviksi koodiksi alustoille ja näin ollen toimii natiivina sovelluksena. Myös Xamarin kääntyy natiiviksi koodiksi ja toimii täsmälleen samalla tavalla kuin natiivit sovellukset. Xamarinilla on kuitenkin joissain tilanteissa pientä suorituskykyongelmaa Android-alustalla käsiteltäessä suuria olioita, koska tällöin Xamarin varaa muistista tilaa C#-oliolle ja tätä vastaavalle Java-oliolle.

Natiivi tyyli ja tuntuma ovat lähes tärkeimmät asiat kehyksen valinnassa. Nämä määrittelevät sovelluksen käyttökokemuksen. Jotta sovellusten käyttö olisi helppoa ja johdonmukaista, tulisi niiden toteuttaa käytössä olevan alustan tyylistandardeja. Vertailluista kehyksistä React Native hyödyntää natiiveja käyttöliittymäkomponentteja ja Xamarinilla käyttöliittymät ovat täysin natiiveja. Qt käyttää käyttöliittymissä omaa QML-kieltä ja käyttöliittymän muokkaus natiivin kaltaiseksi jää kehittäjän murheeksi. Qt kuitenkin tarjoaa joitain valmiita tyylejä natiivien tyylien saavuttamiseksi, mutta päävastuu jää kehittäjälle. Apache Cordova -sovellusten käyttöliittymät taas ovat HTML ja CSS -pohjaisia ja näissäkin tyylin luonti jää kehittäjälle. Eli yhteenvetona React Native ja Xamarin pystyvät tuottamaan natiivin tyylin ja tuntuman, mutta Qt ja Apache Cordova eivät siihen ainakaan täysin kykene.

Jos natiivi tyyli ja tuntuma ovat käyttäjälle tärkeintä, niin kehyksen käyttämä ohjelmointikieli taas on kehittäjille tärkeintä. Esitellyistä kehyksistä React Native ja Apache Cordova käyttävät JavaScript-ohjelmointikieltä, Xamarin C#-ohjelmointikieltä ja Qt C++-ohjelmointikieltä. Evolvitin kannalta C#- ja JavaScript-kielet ovat yrityksessä käytössä lähes kaikissa projekteissa, joten näiden kanssa toimiminen on hyvin hallussa. Myös C++-kokemusta löytyy useilta työntekijöiltä, mutta intoa C++-ohjelmointiin on harvassa. Mainituista kielistä käytetyin on varmasti C#, koska palvelinpuolen sovellukset yrityksessä toteutetaan lähes poikkeuksetta .NET-tekniikoilla ja C#-ohjelmointikielellä.

Vertailluista kehyksistä kaikki ovat maksuttomia yksityiseen käyttöön ja avoimen lähdekoodin projekteihin, mutta sekä Xamarin että Qt ovat maksullisia kaupalliseen käyttöön. Xamarin-alustaa tosin voidaan käyttää kaupallisessa käytössä maksutta, mikäli toiminta

on pientä ja kehittäjiä on vähän. Myös Qt on maksuton kaupalliseen käyttöön, mikäli projekti julkaistaan GPL- tai LGPL-lisenssillä. Evolvitin tapauksessa toiminta on kuitenkin sen verran suurta, että Xamarin-alustan käyttö ei onnistu maksutta, eikä Qt:n tapauksessa GPL- tai LGPL-lisensointi tule kysymykseen. Microsoft kuitenkin muutti Xamarinin lisensointia keväällä 2016 niin, että Xamarin-lisenssi tulee Visual Studio -lisenssin mukana ja näin ollen tulee Evolvitille maksuttomana, koska yrityksellä on jo olemassa Visual Studio -lisenssit. Eli ainoastaan Qt:n käyttö toisi yritykselle lisää kuluja.

Nyt kun kehykset on käyty läpi ja tarkasteltu niiden toimintaa määriteltyjen mittareiden mukaan niin voidaan todeta, että voiton tässä vertailussa vei Xamarin. Xamarinin valintaan päädyttiin kehyksen natiivin suorituskyvyn, tyylin ja tuntuman sekä C#-ohjelmointikielen takia. React Native olisi pystynyt täyttämään lähes samat vaatimukset ja myös JavaScript olisi soveltunut ohjelmointikieleksi, mutta React Native on vielä liian kypsymätön alusta ja työkalut toimivat toistaiseksi täysin ainoastaan OS X -käyttöjärjestelmällä. Apache Cordova jätettiin pois lopullisesta vertailusta, koska kehyksen ei katsottu riittävän yrityksen tarpeisiin. Qt taas sai liian paljon negatiivista palautetta kehittäjiltä käyttökokemusten sekä C++-ohjelmointikielen takia, joten siitäkin ideasta luovuttiin nopeasti. Xamarin taas soveltuu hyvin Evolvitin käyttöön, sillä Microsoftin tekniikat ja työkalut ovat yrityksellä jo päivittäisessä käytössä.

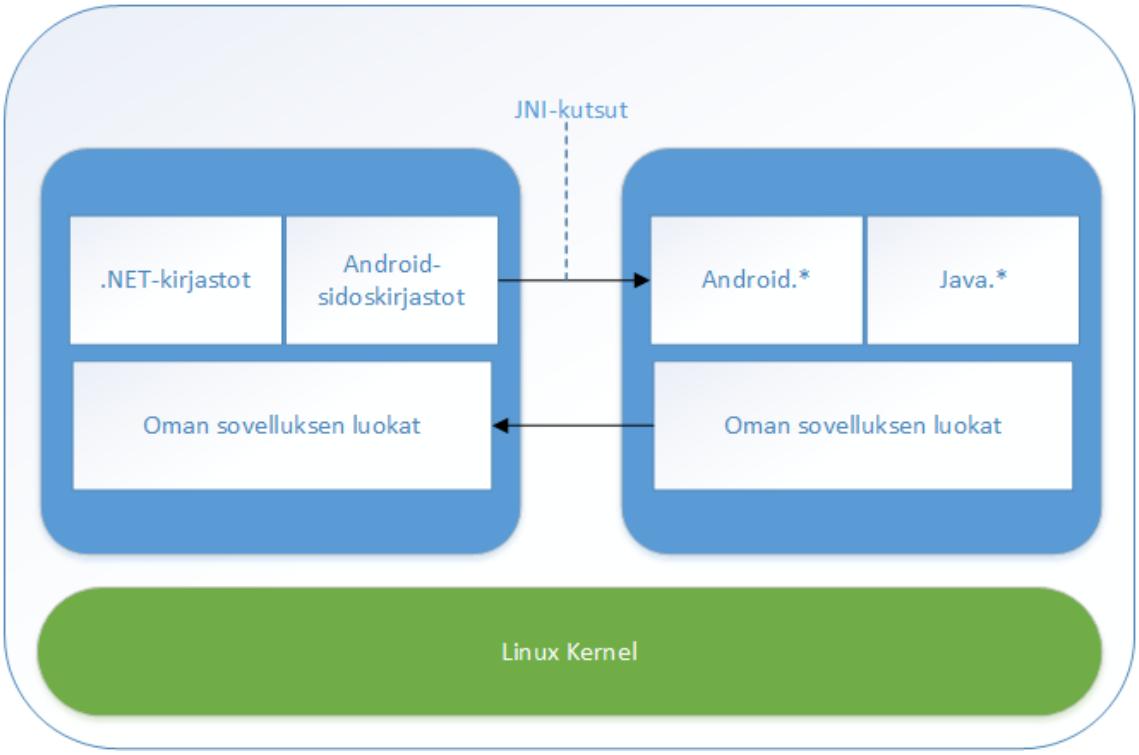
3 XAMARIN-KEHYS

Kuten aiemmassa lyhyessä esittelyssä mainittiin, on Xamarin Mono-projektiin pohjautuva ohjelmointikehys ja Mono taas on .NET-alustaan pohjautuva avoimen lähdekoodin kehitysalusta. .NET on kehitysalusta, jolla voidaan toteuttaa sovelluksia pilvipalvelusta esineiden internettiin ja peleihin. .NET-kehyksellä on mahdollista kehittää sovelluksia Windows, Linux, Android, Mac OS ja iOS -alustoille. (Microsoft 2106.) ECMA:n C# ja Common Language Infrastructure (CLI) -standardit toimivat pohjana Monon toteutukselle .NET-alustasta (Mono Project 2016). CLI tarkoittaa infrastruktuuria, jossa usealla korkean tason kielellä kirjoitettua sovellusta voidaan ajaa useassa erilaisessa järjestelmäympäristössä ilman, että ohjelman tarvitsee ottaa kantaa järjestelmäriippuvaisiin piirteisiin (ECMA International 2012).

Xamarin tarjoaa mahdollisuuden ohjelmoida mobiilialustoille natiiveja sovelluksia käyttäen .NET-alustaa ja C#-kieltä. Pinnan alla toiminta kuitenkin poikkeaa eri alustoilla. iOS-alustalla Xamarinin AOT -kääntäjä kääntää Xamarin iOS-sovellukset suoraan natiiviksi ARM assembly -koodiksi. Android-alustalla taas Xamarinin kääntäjä kääntää sovelluksen välikielelle (engl. Intermediate Language), jonka Just-in-Time (JIT) kääntäjä kääntää natiiviksi assembly-koodiksi, kun sovellus käynnistyy. Molemmissa tapauksissa sovellukset käyttävät Xamarinin ajoympäristöä, joka hoitaa automaattisesti mm. muistinhallinnan, roskien keruun ja alla olevan järjestelmän yhteensovittamisen. (Xamarin Developer 2015.)

3.1 Xamarin Android

Natiivit Android-sovellukset suoritetaan Dalvik-virtuaalikoneessa, joka on samankaltainen kuin Java-virtuaalikone mutta optimoitu laitteille, joissa on rajallisesti resursseja. Xamarin taas pohjautuu Monoon, joka käyttää omaa virtuaalikonetta nimeltään Common Language Runtime (CLR). Xamarin Android sovelluksia ajetaankin samanaikaisesti molemmilla virtuaalikoneilla. Virtuaalikoneiden toiminta perustuu vertaisolioiden (engl. Peer Object) toimintaan Java Native Interface-kehyksen (JNI) läpi. (Reynolds 2014a.)



Kuvio 3. Xamarin.Android sovellusta ajetaan samanaikaisesti kahdessa virtuaalikoneessa (Reynolds 2014a.).

Vertaisoliot ovat Mono-ympäristössä sijaitsevien hallittujen olioiden ja Dalvik-virtuaalikoneessa sijaitsevien Java-olioiden muodostamia oliopareja. Nämä vertaisoliot yhdessä mahdollistavat Xamarin Android -kehyksen toiminnan. Xamarin Android -kehys sisältää sidoskirjastot, joiden sisältävät luokat vastaavat Android sovelluskehyksen luokkia. Sidoskirjastojen metodit toimivat sovittimina, jotka kutsuvat vastaavia Java-luokkien metodeita. Nämä sidoskirjastot mahdollistavat Java-luokkien periytyksen C#-luokiksi ja mahdollistavat käytännössä C#-ohjelmoinnin Android alustalla.
(Reynolds 2014b)

Android-alustalla on käytössä AndroidManifest.xml-tiedosto, joka määrittelee sovelluksen vaatimia oikeuksia, aktiviteetteja, palveluita ja oikeastaan kaikkea sovellukseen liittyvää, joka jollain tavoin vaikuttaa myös sovelluksen ulkopuolelle. Xamarin-kehityksessä voidaan myös käyttää manifest-tiedostoa, mutta Xamarin Android -kehyksellä manifest-tiedosto voidaan korvata C#-annotaatioilla (koodiesimerkki 1). Xamarin Android -kehykseen on toteutettu attribuutit, joiden pohjalta alusta luo manifest-tiedoston ja yhdistää sen projektista löytyvään AndroidManifest.xml tiedostoon. (Xamarin Developer 2016.)

```
// C#-annotaatio
[Activity(Label = "Login", MainLauncher = true,
Icon = "@drawable/ic_launcher",
WindowSoftInputMode = SoftInput.AdjustNothing,
Theme="@style/LoginTheme")]

<!-- AndroidManifest.xml -->
<activity
   android:label="Login"
   android:icon="@drawable/ic_launcher"
   android:windowSoftInputMode="adjustNothing"
   android:theme="@style/LoginTheme">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
</activity>
```

Koodiesimerkki 1. AndroidManifest.xml-tiedoston korvaaminen C#-tyylisillä annotaatioilla.

Android-kehityksessä Xamarinilla saattaa olla hieman suorituskykyongelmia johtuen muistinhallinnasta. Joissain tapauksissa Xamarin Android -sovellus varaa muistia sekä Javaolioille että C#-olioille mahdollistaakseen Xamarin-sovellusten toiminnan, joka aiheuttaa suurempaa muistinkäyttöä (Reynolds 2014). Xamarinilla toteutettu sovelluspaketti on myös suurempi kuin natiivi vastaava johtuen Xamarinin vaatimista ylimääräisistä kirjastoista, jotka tulevat mukaan sovelluspakettiin.

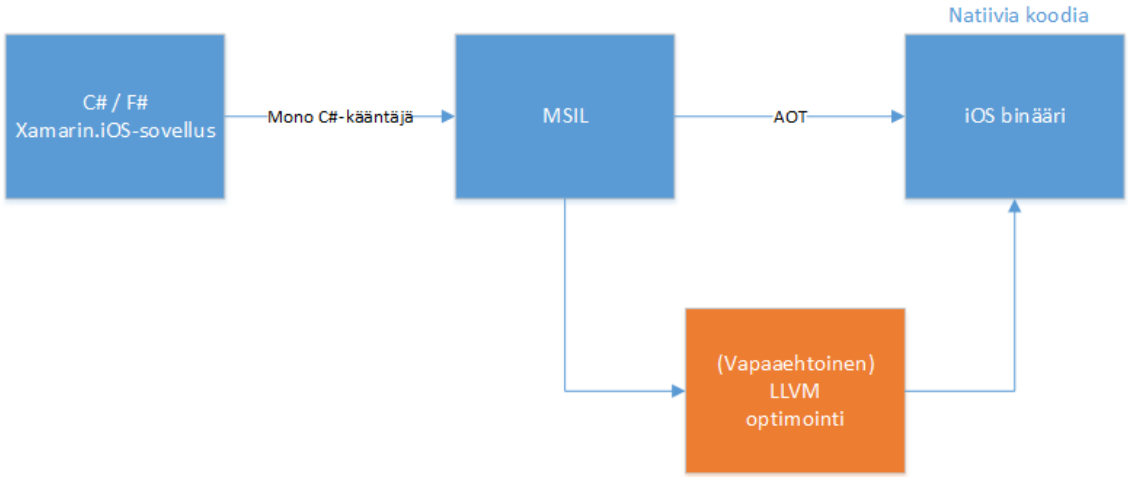
3.2 Xamarin iOS

Xamarin iOS mahdollistaa natiivien iOS-sovellusten kehityksen C#-kielellä ja .NET-alustaa käyttäen. Xamarin käyttää täsmälleen samoja käyttöliittymäkomponentteja kuin natiivissa kehityksessä. Myös Xamarin iOS tarjoaa sidoskirjastot, joiden avulla C#-luokat voivat periytyä iOS:n Objective-C luokista.

Kuten aiemmin jo mainittiin, niin Monon ajoympäristö perustuu JIT -kääntäjään ja siihen, että .NET kielet käännetään ensin välikieleksi ja vasta ajon aikana natiiviksi konekieleksi. Apple on kuitenkin asettanut rajoitteita koskien dynaamisesti luotua koodia ja näin ollen JIT -kääntäminen ei ole sallittu iOS-alustalla.

Koska JIT -kääntäminen ei ole mahdollista iOS-alustalla ja joissain sulautetuissa järjestelmissä, on Mono Project kehittänyt uuden vaihtoehdon, AOT (engl. Ahead-Of-Time)

eli ennenaikaisen kääntäjän. AOT-kääntäjä kääntää kaiken ohjelmakoodin valmiiksi, jolloin ohjelmakoodia ei tarvitse enää kääntää ajon aikana. AOT-kääntäminen on muuten hyvin yksinkertaista paitsi geneerisen koodin tapauksissa. Geneerisen koodin kanssa Mono AOT -kääntäjä suorittaa staattisen analyysin ohjelmakoodista ja päättelee, mitä kaikkia mahdollisuuksia ohjelma käyttää geneeristen luokkien ilmentymissä ja luo käännetyn koodin kaikista mahdollisuuksista (Mono Project 2016). AOT kääntäjä siis mahdollistaa .NET alustan käytön iOS-alustalla ja samalla lyhentää sovellusten käynnistymisaikoja ja mahdollistaa jopa paremman suorituskyvyn. AOT kuitenkin aiheuttaa joitain rajoitteita geneeriseen koodiin, koska joissain tapauksissa kääntäjä ei voi mitenkään päätellä, mitä mahdollisia ilmentymiä geneerisistä olioista voi syntyä. (Peppers 2014, s. 145.)



Kuvio 4. AOT-käännösprosessin kuvaus. (Xamarin Developer 2015)

3.3 Jaettava ohjelmakoodi

Xamarinilla toteutetuissa projekteissa voidaan jakaa jopa 80% ohjelman lähdekoodista eri alustojen välillä (Xamarin Developer 2015). Ohjelmakoodia voidaan jakaa useilla eri tavoilla, mutta nykyään käytetään lähinnä kahta erilaista tapaa. Nämä vaihtoehdot ovat Portable Class Library (PLC) eli siirrettävä luokkakirjasto ja jaetut projektit. Näiden vaihtoehtojen suurin ero on se, että siirrettävät luokkakirjastot voivat sisältää ainoastaan alusta riippumatonta koodia, eikä siirrettävistä luokkakirjastoista näin ollen voida kutsua alustakohtaisia ohjelmointirajapintoja.

Siirrettävien luokkakirjastojen suurin etu piilee siinä, että koodi on kokonaisuudessaan alustariippumatonta. Näin ollen koodin jakaminen voidaan toteuttaa keskitetysti, refaktorointi vaikuttaa kaikkialle ja muiden projektien on helppo käyttää kirjastoa. Varjopuolena

on kuitenkin se, ettei siirrettävistä luokkakirjastoista voida viitata alustariippuvaisiin kirjastoihin, eivätkä kaikki alustoilla muuten käytössä olevat luokat ole käytettävissä siirrettävissä luokkakirjastoissa. (Xamarin Developer 2015.)

Siirrettävät luokkakirjastot soveltuvat hyvin ohjelmien niihin osiin, joiden ei tarvitse olla tietoisia käytettävästä alustasta. Esimerkiksi webrajapintojen käyttö voidaan helposti toteuttaa siirrettävänä luokkakirjastona, jolloin rajapintamuutoksista johtuvat korjaukset voidaan toteuttaa keskitetysti kaikille alustoille samalla kertaa. Siirrettävissä luokkakirjastoissa voidaan myös määritellä tiedonsiirto-oliot ja näin ollen hoitaa myös tietojen serialisointi ja jäsentely olioksi siirrettävissä luokkakirjastoissa. Siirrettäviin luokkakirjastoihin voidaan myös toteuttaa rajapintoja, jotka alustakohtaiset projektit toteuttavat ja näin ollen on mahdollista käyttää alustakohtaisia ominaisuuksia ja kirjastoja. Tämä kuitenkin vaatii Provider-mallin tai riippuvuusinjektiomallin käyttämistä. (Xamarin Developer 2015.)

Jaetut projektit taas mahdollistavat yleisen koodin kirjoittamisen samaan paikkaan, johon viitataan alustakohtaisista projekteista. Ohjelmakoodi käännetään viittaavan projektin mukana ja jaetuissa projekteissa voidaan käyttää esikääntäjän ohjeita, joka mahdollistaa alustakohtaisten ohjelmointirajapintojen käytön. (Xamarin Developer 2015.)

Esikääntäjälle voidaan antaa useita erilaisia komentoja, mutta Xamarin projekteissa tärkeimmät ovat #if- ja #endif-ohje. Näillä voidaan kertoa esikääntäjälle, mitkä osiot tulee kääntää millekin alustalle. Esikääntäjäkomennot lähtevät kuitenkin helposti paisumaan sekä vaikeuttavat luettavuutta, mikäli projektin parissa työskentelee suurempi tiimi tai tiimin jäsenillä ei ole paljoa kokemusta. Jaetut projektit eivät myöskään käänny .dll tiedostoiksi, joten tällaisia projekteja ei voida jakaa muuten kuin lähdekoodin muodossa. (Peppers 2015.)

Riippuvuusinjektion ja IoC-toteutuksen (engl. IoC Container) avulla jaetuissa projekteissa voidaan toteuttaa yhtenäinen rajapinta, jota kaikki projektiin viittaavat toteutukset käyttävät. Tämän kaltainen malli mahdollistaa sen, ettei ohjelman tarvitse olla tietoinen varsinaisesta toteutuksesta ja toteutukset voidaan räätälöidä täysin alustakohtaisiksi. Tällainen malli kuitenkin vaikeuttaa muutosten toteuttamista, koska mikäli muutos koskee alustariippuvaista osiota, joudutaan muutokset tekemään useampaan paikkaan. Tämä myös altistaa alustariippuvaisille ohjelmavirheille.

Jaetut projektit ja siirrettävät luokkakirjastot ovat parhaat käytännöt ohjelmakoodin jakamiseen Xamarin-alustalla. On mahdollista myös jakaa ohjelmakoodia tiedostolinkityksen avulla tai kloonatuilla projekteilla, mutta nämä molemmat ovat jo hieman vanhentuneita tapoja jaettujen projektien julkistamisen jälkeen. (Peppers 2015.) Näistä vaihtoehdoista siirrettävät luokkakirjastot tuovat selkeimmän ja parhaan rakenteen monialustaisille sovelluksille. Siirrettävien luokkakirjastojen käyttö vaatii hieman enemmän työtä ja säätöä, jotta pystytään yhdistämään alustakohtainen ja täysin jaettu koodi toisiinsa, mutta lopputulos on selkeämpi ja paremmin hallittu.

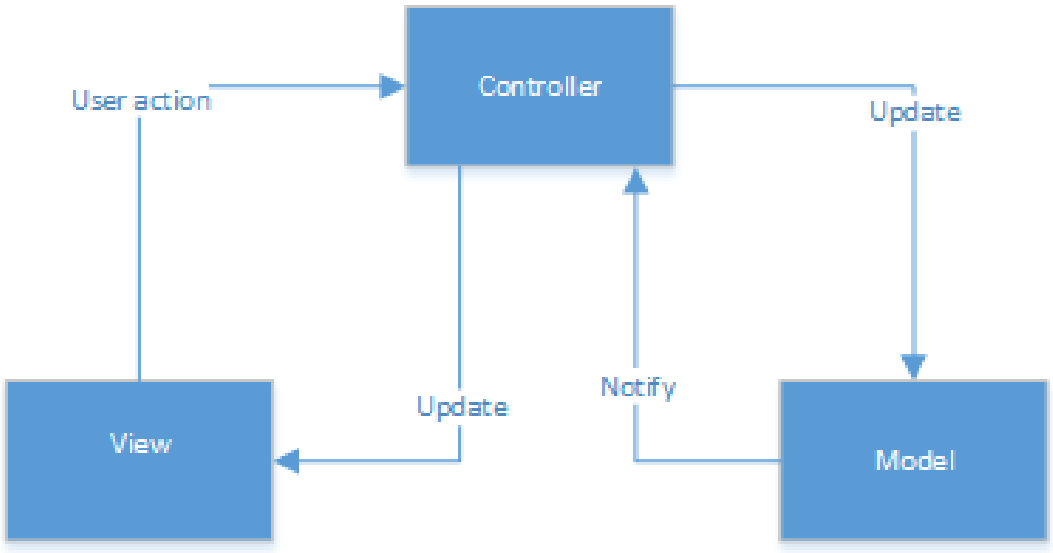
Hyvä tapa hallita jaettua ohjelmakoodia onkin toteuttaa hyvin organisoitu siirrettävä luokkakirjasto, jonka ulkopuolelle tarjoamat palvelut on toteutettu rajapintojen taakse. Kun näin toimitaan, voidaan tarvittaessa toteutukset muokata alustakohtaisiksi ja esim. riippuvuusinjektion avulla hallita mitä toteutusta sovelluksessa käytetään.

3.4 Arkkitehtuuri

Monialustaisen sovelluksen arkkitehtuurin suunnittelun lähtökohtana tulee olla ohjelmakoodin jakaminen useille alustoille. Kun alusta asti suunnittelussa huomioidaan ohjelmakoodin jakaminen, on tuloksena yleensä selkeä ja tehokas sovellus sekä mahdollisimman vähän koodia. Hyvään arkkitehtuuriin päästään noudattamalla hyviä ohjelmointikäytäntöjä, joita tällaisissa sovelluksissa ovat esimerkiksi kotelointi, vastuiden jaottelu ja polymorfismi. (Xamarin Developer 2016).

3.4.1 iOS

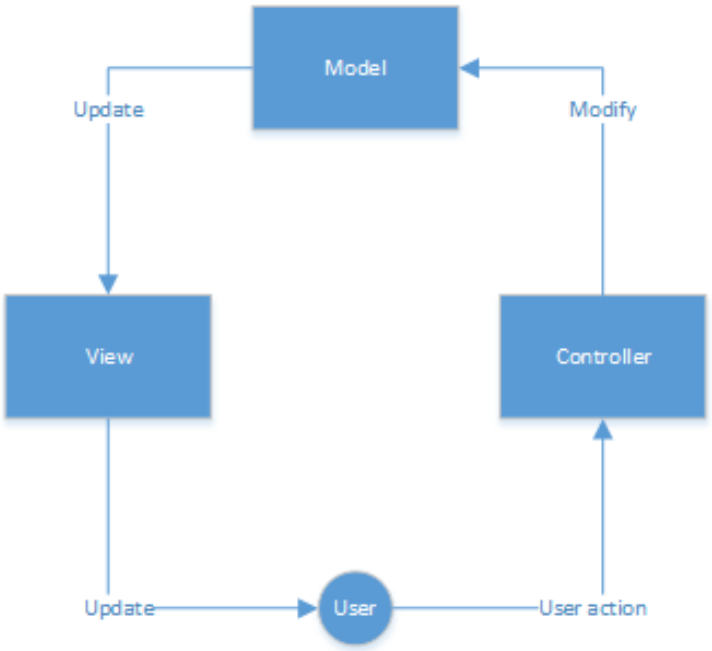
iOS-sovellukset noudattavat arkkitehtuuriltaan Applen johdannaista Malli-Näkymä-Ohjain-suunnittelumallista (engl. Model-View-Controller, MVC), joka määrittelee olioiden roolit ja kommunikaation olioiden välillä (kuvio 5). (Apple 2015.) MVC-malli on jo hyvin vanha suunnittelumalli, jonka variaatioita on käytetty jo 70-luvulla Smalltalk-ohjelmointikielellä toteutetuissa sovelluksissa. MVC-malli parantaa olio-ohjelmoinnin periaatteilla toteutettujen sovellusten uudelleenkäytettävyyttä, selkeyttää rajapintoja ja parantaa valmiuksia muutoksille. (Apple 2012.)



Kuvio 5. Applen MVC-malli.

3.4.2 Android

Android-sovellukset eivät noudata selkeästi mitään tiettyä arkkitehtuurimallia. Voidaan kuitenkin katsoa Android-sovellusten mukailevan MVC-mallia (kuvio 6) jolloin näkymä- ja ohjaintasot ovat jossain määrin sulautuneet yhteen Android-sovelluksissa. Toisten mielestä Android kuitenkin noudattaa enemmänkin Malli-Näkymä-Esittäjä-suunnittelumallia (engl. Model-View-Presenter) MVP, joka on johdannainen MVC-mallista.

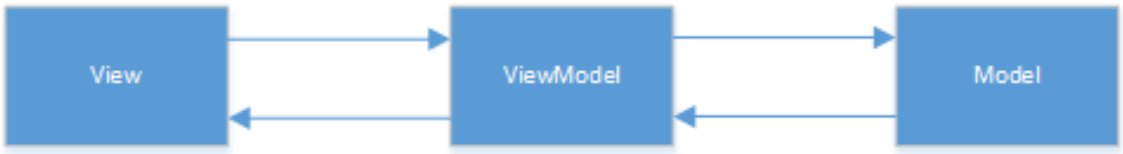


Kuvio 6. Perinteinen MVC-malli.

3.4.3 MVVM-malli

MVVM eli Malli-Näkymä-Näkymämalli (engl. Model-View-ViewModel) (kuvio 7) on MVC-mallin variaatio, joka jakaa perinteisen näkymäkerroksen logiikan kahteen erilliseen kerrokseen, näkymäkerrokseen sekä näkymämallikerrokseen. Näkymän tehtävänä on tarjota käyttöliittymä ja näkymämalli on vastuussa esityslogiikasta. Esityslogiikka pitää sisällään tiedon muuttamisen mallista käyttöliittymälle sopivaksi ja käyttäjän syötteen siirtämisen mallille. (Reynolds 2014a)

MVVM-malli on hieman monimutkaisempi toteuttaa, eikä Android ja iOS -alustaa ole suunniteltu suoraan MVVM-arkkitehtuuriin käytölle, mutta tällaisen arkkitehtuurin toteuttamiseen on olemassa kolmannen osapuolen kirjastoja. Esimerkiksi MvvmCross-kehys mahdollistaa monialustaisten sovellusten kehittämisen MVVM-mallin mukaan Xamarin Android ja Xamarin iOS -alustoille.



Kuvio 7. MVVM-malli.

3.5 Linker

Linker on osa Xamarin Android -sovelluksen käännös- ja paketointiprosessia. Android-sovelluksissa Linkerin tehtävä on poistaa ylimääräinen ohjelmakoodi ja näin ollen pienentää sovelluksen kokoa. Linker käyttää hyväkseen staattista analyysiä ja poistaa kirjastoja, tyyppejä ja muuttujia joihin ei ole viittauksia.

Linkerille voidaan antaa kolme erilaista vaihtoehtoa. ”Ei linkitystä” (engl. Don’t Link), ”linkitä SDK-kirjastot” (engl. Link SDK Assemblies) ja “linkitä kaikki” (engl. Link All Assemblies). Ei linkitystä vaihtoehto kytkee linkityksen kokonaan pois ja sovelluspakettiin tulee mukaan kaikki Xamarinin kirjastot ja kaikki muutkin projektissa viitatut kirjastot, jolloin sovelluksen koko voi olla hyvinkin suuri. Linkitä SDK-kirjastot vaihtoehto toteuttaa linkityksen Xamarin kirjastoille, mutta jättää ohjelmoijan oman koodin ja kolmansien osapuolien kirjastot rauhaan. Linkitä kaikki vaihtoehto taas tekee linkityksen

kaikelle koodille, myös ohjelmoijan omalle ja kolmansien osapuolien kirjastoille. (Xamarin Developer, 2016.) Linkerille voidaan myös suoraan C#-annotaatioattribuuteilla kertoa, mikäli linkityksen ei haluta esim. jotain tiettyä luokkaa, metodia tai assemblyä. Eksplisiittinen linkerin konfigurointi on kuitenkin hankalaa ja virhealtista, näinpä linkerin konfiguraatioon ei pääsääntöisesti ole tarvetta koskea.

3.6 Yhteenveto

Kehitys Xamarin-alustalla jäljittelee siis Android ja iOS-alustoiden natiiveja ohjelmointityylejä ja ohjelmointi alustan avulla on hyvin samankaltaista, kuin natiivien sovellusten ohjelmointi. Xamarin-alusta tuo kuitenkin lisää abstraktiota ja hieman sekavuutta natiivikehityksen päälle, joka vaatii opettelua ja sisäistämistä. Xamarin-alusta ei siis ole sellaisenaan helpompaa ja yksinkertaisempaa kehitystä. Xamarin-alusta vaatii kuitenkin ainoastaan yhden ohjelmointikielen sisäistämisen ja sopivien luokkakirjastojen löytämisen ainoastaan yhdelle alustalle, mikä hieman helpottaa mobiilikehityksen opettelua.

Xamarin-alustan voima perustuu nimenomaan jaettavaan ohjelmakoodiin ja ohjelmointiin yhdellä kielellä. Myös mahdollisuus viedä olemassa oleva .NET-osaaminen mobiilikehitykseen lisää alustan vetovoimaa yrityksille, joissa .NET-työkalut ovat olleet käytössä.

Vaikka Apple julkaisi Swift-ohjelmointikielen korvaamaan Objective-C -ohjelmointikielen vuonna 2014, niin C# on mobiilikehityksessä käytössä olevista kielistä edelleen edistyksellisin. Perusperiaatteiltaan Java, Swift ja C# ovat hyvin samankaltaisia ohjelmointikieliä ja kaikissa on omat hyvät ja huonot puolensa. Kuitenkin varsinkin mobiilikehityksessä C#-ohjelmointikieli helpottaa ikävien ja hankalien asioiden toteutusta.

Hyviä esimerkkejä C#-ohjelmointikielen tarjoamista helpotuksista mobiilikehityksessä ovat Lambdafunktiot, LINQ, async-määrite ja delegaatit. Myös Swift ja Javan kahdeksas versio tarjoavat omat toteutukset lambdafunktioista, joten nämä eivät ole sinänsä mitenkään erikoisia. C#-kielen lambdafunktioista erityisiä tekee niiden käyttö LINQ-kyselyiden yhteydessä. LINQ-kyselyt mahdollistavat SQL-kyselyitä muistuttavien kyselyiden kirjoittamisen vahvasti tyypitettyjä kokoelmia vasten (koodiesimerkki 2). Edellä mainittua tapaa on helpotettu vielä lisää mahdollistamalla LINQ-kysely lambdafunktioiden

avulla (koodiesimerkki 3), joka helpottaa huomattavasti käsiteltäessä dataa, jota pitää pystyä suodattamaan, ryhmittelemään tai järjestämään.

```
// Company on tässä tapauksessa olio, joka sisältää kentät Id(int),
// CompanyId(int) ja CompanyName (string)
IEnumerable<Company> filteredCompanies = from company in companies
    where company.CompanyName.StartsWith("a",
        StringComparision.CurrentCultureIgnoreCase)
    orderby company.CompanyName
    select company;
```

Koodiesimerkki 2. Perinteinen LINQ-kysely toteutettuna C#-ohjelmointikielellä.

```
// LINQ Lambdafunktiolla
IEnumerable<TkTask> orderedTasks =
    tasks.OrderBy(n => n.CustomerName)
    .ThenBy(n => n.ProjectName).ThenBy(n => n.Name);
```

Koodiesimerkki 3. LINQ-kysely toteutettuna lambdalauseita käyttäen C#-ohjelmointikielellä.

Aiemmassa kappaleessa mainittiin myös async-määrite. Async-määrite on uusi .NET 4.5 -versiossa lisätty ominaisuus, joka tuo yksinkertaistetun lähestymistavan asynkroniseen ohjelmointiin. Tällä määritteellä voidaan merkitä jokin pitkäkestoinen metodi, anonyymimetodi tai lambdafunktio, esimerkiksi jokin verkon yli tapahtuva kysely, ajettavaksi asynkronisesti, jolloin suoritus ei estä sovelluksen muuta toimintaa (koodiesimerkki 4). Tämän kaltainen asynkroninen ohjelmointitapa helpottaa huomattavasti responsiivisten sovellusten kehitystä.

```
protected HttpClient CreateClient()
{
    HttpClient client = new HttpClient
    {
        BaseAddress = new Uri(BaseUrl)
    };
    client.DefaultRequestHeaders.Authorization =
        new AuthenticationHeaderValue("bearer", _token);
    return client;
}

protected async Task<T> Retrieve(string url)
{
    string jsonString = await RetrieveContent(url);
    if (jsonString == null)
    {
        throw new EmsHttpException(
            "Retrieving content failed. Content was empty.");
    }
    return JsonConvert.DeserializeObject<T>(jsonString);
}
```

```
private async Task<string> RetrieveContent(string url)
{
    HttpClient client = CreateClient();
    HttpResponseMessage response = await client.GetAsync(url);
    if(response.StatusCode == HttpStatusCode.OK)
    {
        return await response.Content.ReadAsStringAsync();
    }
    throw new EmsHttpException(
        "Retrieving content failed.", response.StatusCode);
}
```

Koodiesimerkki 4. Asynkronisten metodien toiminta async- ja await-määritteiden kanssa.

Myös delegaatit mainittiin C#-kielen etuna. Delegaatti on tyyppi joka edustaa viittauksia metodeihin, joilla on tietyt parametrit ja paluuarvo (MSDN 2015). Delegaatteja käytetään välittämään metodeita argumentteina toisille metodeille, eli delegaatit metodit ovat usein tapahtumakäsittelijöitä tai takaisinkutsumetodeita. Koska Java ei tunne delegaatteja, takaisinkutsufunktioita tai funktio-osoittimia, helpottaa delegaattien käyttö Android-ohjelmointia ja selkeyttää ohjelman rakennetta. Delegaateilla voidaankin Android-sovelluksilla korvata Java-tyyliset yhden metodin sisältävät kuuntelijarajapinnat (koodiesimerkki 5).

```
// Java-tyylinen kuuntelijarajapinta
public interface OnClickListener {
   void onClick(View v);
}

// Käyttö

buttonX.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // Käsitellään click-tapahtuma
    }
});

// C#-tyylillä käyttäen delegaattia
buttonX.Click += delegate(View v) {
    // Käsitellään click-tapahtuma
};

// Tai vielä lyhempi vaihtoehto
buttonX.Click += (View v)  => {
    // Käsitellään click-tapahtuma
};
```

Koodiesimerkki 5. Java-tyylinen kuuntelijatoteutus vastaan C#-tyylinen delegaattitoteutus.

Xamarinin etu piilee yhdessä ohjelmointikielessä ja alustassa sekä C#-kielen tehokkaassa ilmaisussa. Näin ollen voidaan summata, että Xamarin-alusta soveltuu loistavasti kehittäjille ja yrityksille, joilla on jo kokemusta .NET-alustasta eikä välttämättä ole vielä muuta kokemusta mobiilikehityksestä. Muussa tapauksessa Xamarin vaatii natiivien sovellusten toimintaan perehtymisen lisäksi vielä .NET-alustan sisäistämisen ja näin ollen lähtökohdista ja tarpeista riippuen natiivikehitys tai jokin muu monialustakehys voi tarjota paremmat puitteet kehitykselle. Evolvitin tapauksessa Xamarin kuitenkin on yllä mainittujen perusteiden pohjalta selkeä vaihtoehto mobiilikehityksen edistämiseen.

4 TUNTIKIRJAUSSOVELLUS

Opinnäytetyön osana toteutettiin Evolvitille tuntikirjaussovellus Android ja iOS-alustoille. Evolvitilla oli jo käytössä yrityksen sisällä kehitetty tuntikirjausjärjestelmä, joka on toteutettu .NET-tekniikoilla. Mobiilisovelluksia varten vanhaan järjestelmään muokattiin mobiiliystävälliset REST-rajapinnat, koska vanhat WebService-rajapinnat eivät mobiilikäyttöön enää taipuneet.

Evolvitin vanhalla järjestelmällä tuntien kirjaus tapahtuu pääosin yrityksen sisäverkossa toimivalla verkkosovelluksella. Verkkosovellus toimii hyvin henkilöstöllä, joka istuu pääosin toimistolla ja on koko ajan tietokoneen ääressä ja yhteydessä sisäverkkoon. Evolvitilla kuitenkin suuri osa henkilöstöstä on paljon asiakkaiden luona ja reissussa, jolloin tuntien kirjaus verkkosovelluksella voi olla haastavaa tai vähintäänkin epämiellyttävää. Matkalla töistä kotiin esimerkiksi junassa tai autossa ei mielellään enää kaiva tietokonetta esiin ja yhteydetkin usein ovat epävakaat, joten tuntien kirjaaminen jää usein seuraaville päiville. Näissä tilanteissa taas on jo hankala muistaa, mitä aiemmin onkaan tehnyt ja kuinka paljon tunteja pitikään merkitä. Näihin ongelmiin haluttiin vastata toimivalla mobiilisovelluksella.

Tärkeimmät kriteerit sovellukselle olivat: Android ja iOS-tuki, helppo ja nopea tuntien kirjaus, sujuva matkalaskujen kirjaus sekä toimivuus heikoillakin yhteyksillä. Näitä ongelmia lähdettiin ratkaisemaan Xamarin-kehyksellä toteutetulla mobiilisovelluksella.

4.1 Kehitysympäristö

Kehitystä Xamarin-alustalla voidaan tehdä joko Microsoftin Visual Studio tai Xamarin Studio ohjelmointiympäristöillä. Visual Studio on saatavilla ainoastaan Windows käyttöjärjestelmälle, kun taas Xamarin Studio on saatavilla myös OS X -käyttöjärjestelmälle. Visual Studio -ympäristöön pitää myös asentaa Xamarin For Visual Studio -lisäosa, jotta Xamarin-kehitys on mahdollista (liite 1). Lisäksi iOS-kehityksen mahdollistamiseksi tarvitaan Mac-tietokone, jossa on OS X Yosemite tai uudempi käyttöjärjestelmä, Xamarin iOS SDK, iOS SDK ja Xcode 7 tai uudempi ohjelmointiympäristö sekä SSH-etäkäytön on oltava sallittuna (liite 2).

Tuntikirjaussovellusten kehittämiseen käytettiin Visual Studio -ohjelmointiympäristöä sekä jossain määrin Xcode-ohjelmointiympäristöä käyttöliittymäkehitykseen, koska Visual Studion iOS -editori ei kyennyt aivan kaikkien käyttöliittymäkomponenttien luontiin. Näissä tapauksissa paras ratkaisu oli viedä storyboard-tiedosto Macille ja tehdä tarvittavat muutokset Xcodella ja viedä storyboard-tiedosto takaisin kehitysympäristöön.

4.2 Arkkitehtuuri

Arkkitehtuuriksi sovelluksille valikoitui karkeasti MVC-mallia mukaileva arkkitehtuuri. Näin näkymä- ja ohjaintasot voidaan toteuttaa alustakohtaisesti ja sovellusvuo vastaa täysin natiivia. Tällainen arkkitehtuuri mahdollisti myös mallitason toteuttamisen täysin siirrettävänä luokkakirjastona, jolloin malliin tulevat muutokset saadaan hyvin keskitettyä. Käyttöliittymämuutokset joudutaan joka tapauksessa toteuttamaan aina molemmille alustoille erikseen.

4.3 Ohjelmakoodin jakaminen useille alustoille

Tuntikirjaussovelluksessa ohjelmakoodin jakaminen eri alustoille toteutettiin kahdella siirrettävällä luokkakirjastolla (kuvio 8). Toinen kirjasto sisältää kaikki dataoliot ja datansiirto-oliot ja toinen kirjasto kaiken jaettavan logiikan. Eli karkeasti nämä siirrettävät luokkakirjastot muodostavat MVC-mallin mukaisen malli-kerroksen. Tämä jako ei ole toiminnallisuuden kannalta välttämätöntä, mutta tämän kaltainen rakenne helpottaa hahmottamaan projektin rakennetta paremmin.

Logiikkaprojekti toteutettiin niin, että kaikki julkiset luokat toteuttavat niihin liittyvän rajapinnan, jotta alustakohtaisissa toteutuksissa voidaan IoC-säiliötä hyväksikäyttäen toteuttaa tarvittavat alustakohtaiset muutokset ja silti pitää huolta siitä, että sovellukset kaikilla alustoilla toteuttavat samat toiminnot. Tämä myös auttaa muutoksien tapahtuessa, koska rajapintojen muuttuessa myös kaikki alustakohtaiset toteutukset on päivitettävä, jotta ohjelmat kääntyvät.

```
EvolvitEMS
|
|___Android
|      |___Activities
|      |___Services
|      |___Utils
|      |___Views
|      |___Resources
|
|___iOS
|      |___Controllers
|      |___Services
|      |___Utils
|      |___Resources
|
|___Shared
|      |___Extensions
|      |___Utils
|      |___Services
|            |___Implementations
|
|___Data
       |___Dto
       |___Models
```

Kuvio 8. Xamarin-projektin rakenne.

Android ja iOS sovelluksissa käytettiin Autofac-kirjastoa IoC- ja riippuvuusinjektiototeutukseen. Autofac-kirjasto valikoitui siitä syystä, että kyseistä kirjastoa on Evolvitilla käytetty onnistuneesti useissa projekteissa, joten kirjaston toiminta oli tuttua sekä hyväksi havaittua. Autofac myös mahdollisti erilaiset riippuvuusinjektiototeutukset Androidille ja iOS:lle. IoC-säiliön luonti kuitenkin toimii molemmilla alustoilla samalla tavalla (koodiesimerkki 6). Androidilla IoC-säiliön luonti tulee tehdä Application-olion luonnin yhteydessä, jotta voidaan varmistua säiliön olemassaolosta kaikissa tapauksissa ja iOS:lla luonti taas tapahtuu AppDelegate-luokassa.

```
// Luodaan IoC-säiliö
var builder = new ContainerBuilder();

// Rekisteröidään toteutukset rajapinnoille
builder.RegisterType<LoginService>().As<ILoginService>();
builder.RegisterType<TaskService>().As<ITaskService>();
builder.RegisterType<AndroidAppStorage>().As<IAppStorage>();

// Tallennetaan rakennettu IoC-säiliö
// ServiceFactory on staattinen luokka, joka mahdollistaa
// IoC-säiliön saatavuuden.
ServiceFactory.Container = builder.Build();

// IoC-säiliön kautta voidaan hakea rekisteröity toteutuks
// käyttäen Resolve()-metodia.
ILoginService loginService =
    ServiceFactory.Container.Resolve<ILoginService>();
```

Koodiesimerkki 6. IoC-säiliön luonti ja rajapintoja vastaavien toteutuksien rekisteröinti Autofac-kirjastolla Android-alustalla.

iOS-alustalla riippuvuusinjektio toteutettiin rakentajainjektiolla, joka mahdollistaa tarvittavien riippuvuuden määrittelyn suoraan olion rakentajaan ja Autofac-kirjasto pitää huolen riippuvuuksien välityksestä oikeisiin paikkoihin (koodiesimerkki 7). Rakentajainjektiota käytetään myös esimerkiksi .NET-pohjaisissa verkkosovelluksissa ja Googlen AngularJS-kirjastossa.

Androidilla rakentajainjektion käyttö ei ole mahdollista johtuen siitä, että Android-alusta ei anna pääsyä Activity-olioiden rakentajiin. Tästä johtuen Androidilla riippuvuusinjektion sijaan käytettiin palvelun paikallistajamallia (engl. Service Locator pattern), jonka perusideana on se, että paikallistaja-oliolla on tieto kaikista käytettävissä olevista palveluista. (Fowler, 2004) Tämä toteutettiin niin, että Android-sovellukseen luotiin staattinen luokka, johon asetettiin Autofac-kirjaston luoma IoC-säiliö (koodiesimerkki 6). Näin ollen sovelluksessa riippuvuudet voitiin hakea IoC-säiliöstä säiliön tarjoamalla Resolve-metodilla.

4.4 Alustakohtaisten komponenttien kehitys

Kehitys Visual Studio -ympäristössä pyrkii jäljittelemään natiivia kehitystä ja pääsääntöisesti onnistuukin siinä, varsinkin Androidin tapauksessa. Jopa OS X ja iOS-kehitys onnistuu Visual Studiolla, mutta vaatii aktiivisen SSH -yhteyden OS X -käyttöjärjestelmään, jotta kääntäminen sekä käyttöliittymäkehitys ovat mahdollisia.

Xamarin-alustan heikoin kohta on kuitenkin käyttöliittymäkomponenttien kehitys. Visual Studiolla käyttöliittymien kehittäminen on kankeaa ja esikatselu on useissa tapauksissa täysin mahdotonta, jolloin muutoksien vaikutukset näkyvät vasta simulaattorissa tai laitteessa kokeillessa. Kuten jo kehitysympäristöä kuvatessa todettiin, niin joissain tapauksissa parasta on vain siirtää käyttöliittymäkomponentit joko Xcode tai Android Studio -ympäristöön ja toteuttaa tarvittavat muutokset näillä työkaluilla, jonka jälkeen muokatut komponentit voidaan viedä takaisin oikeaan kehitysympäristöön.

Android-sovelluksen käyttöliittymä toteutettiin käyttäen kolmea aktiviteettia (engl. Activity), jotka ovat Android-sovellusten peruskomponentteja ja joita on jokaisessa sovelluksessa oltava vähintään yksi. Aktiviteetit toimivat perustana kirjautumiselle, tuntien kirjauksella ja matkalaskuille. Näiden aktiviteettien päälle toiminnallisuudet rakennettiin käyttöliittymäpalasista (engl. Fragment) ja siirtyminen aktiviteettien välillä toteutettiin sivusta vedettävällä valikolla (engl. Navigation Drawer). Aktiviteetit ja käyttöliittymäpalaset muodostavat arkkitehtuurin näkymä- ja ohjaintasot.

iOS-sovellukset rakentuvat natiivisti MVC-mallin mukaan ja tuntikirjaus sovelluksessa noudatettiin tätä mallia. Sovellus toteutettiin kolmen näkymäohjaimen avulla (engl. View Controller), joiden jako on samanlainen kuin Android-sovelluksessa. Käyttöliittymä toteutettiin iOS:n käyttämällä storyboard-tiedostolla. Näkymäohjaimet sijoitettiin välilehtiohjaimen sisälle, joka mahdollistaa navigaation välilehtipalkin avulla. iOS-toteutuksessa storyboard muodostaa arkkitehtuurin näkymätason ja näkymäohjaimet muodostavat ohjaintason.

Kuten voidaan huomata, on Android ja iOS -sovelluksien kehittäminen erilaista myös Xamarin-alustalla, mutta voidaan nähdä myös yhtäläisyyksiä. Löyhästi voidaan ajatella näkymäohjainten ja aktiviteettien vastaavan toisiaan ja näihin sijoitetaankin hyvin samanlaista ohjelmalogiikkaa. Kuitenkin Android-sovelluksien toiminta on hieman sekavampaa ja aktiviteettien ja käyttöliittymäpalasten sekoitus käyttöliittymän toteutuksessa tekee käyttöliittymän hallinnasta hankalampaa, eikä täysin selkeää linjaa siitä kumpia tulisi suosia ole. iOS-sovelluksien rakenne on selkeämpi ja osien vastuut ovat myös paremmin selvillä.

4.5 Lopputulos

Loppujen lopuksi sovelluksen toteuttaminen Xamarin-alustalla oli odotettua haastavampaa. Huonot käyttöliittymätyökalut hidastivat varsinaista kehitystä ja aiheuttivat turhautumista. Jälkikäteen pohdittuna sovellusten toteuttaminen natiiveina ratkaisuina olisi voinut tässä tapauksessa olla nopeampaa. Toisaalta Xamarin valittiin toteutus alustaksi nimenomaan sen takia, että alustan käyttö helpottaa jatkossa sovelluksien ylläpitoa ja jatkokehitystä.

Xamarin-alusta on myös varmasti Evolvitin kannalta helpompi ottaa haltuun kuin natiivi Android ja iOS-kehitys. Vaikka Xamarin Android ja Xamarin iOS -kehykset vaativatkin tuntemusta alustakohtaisesta kehityksestä, niin C#-kehittäjät pääsevät nopeammin kiinni todelliseen kehitykseen näillä kehyksillä kuin kehittäessä natiiveja Android ja iOS-sovelluksia. Kehityksestä jäi kokonaisuudessaan positiivinen kuva yritykselle ja mobiilisovelluksia tullaan jatkossakin toteuttamaan Xamarin-alustaa käyttäen.

Sovellukset jäivät tämän työn jälkeen vielä testausvaiheeseen ja virallinen julkaisu Evolvitin sisällä tapahtuu loppuvuodesta 2016. Sovellukselle tulee varmasti vielä jatkokehitystarvetta prosessien muuttuessa ja uudistuessa. Yksi todennäköisistä jatkokehitettävistä asioista on matkalaskujen kirjausprosessi, jonka sujuvuuteen panostettiin jo työn aikana. Ongelmakohdat ja tarpeet nousevat kuitenkin esille vasta, kun henkilöstö todella pääsee käyttämään näitä toiminallisuuksia.

Vasta jatkokehityksen kanssa voidaan arvioida tarkemmin kuinka ylläpidettäviä Xamarin-sovellukset ovat, mutta uskon vakaasti, että valittujen ratkaisujen ansiosta sovellusta on helppo kehittää ja ylläpitää myös jatkossa.

5 YHTEENVETO JA LOPPUTULOKSET

Työssä on nyt ensin pohdittu Evolvitin käyttöön sopivaa monialustaisen kehityksen mahdollistavaa alustaa tai kehystä, jonka lopputuloksena päädyttiin Xamarin-alustaan. Xamarinia on tutkittu ja selvitetty alustan mahdollisuuksia sekä mahdollisia ongelmakohtia. Työ huipentui mobiilisovellusten toteuttamiseen Xamarin-alustalla. Matkalla prosessi on opettanut suhtautumaan kriittisesti eri ohjelmointikehysten lupauksiin ja nostanut esille kehyksien vertailun tärkeyden sekä kannattavuuden. Lisäksi prosessi on herättänyt kysymyksiä ja huomioita eri kehysten soveltuvuudesta erilaisiin käyttötarkoituksiin.

Ensimmäisenä huomiona nousee esille se, että mahdollisuudet monialustaiseen mobiilikehitykseen ovat hyvin monipuoliset. Kehyksen voi valita täysin omien lähtökohtien ja kiinnostusten mukaan ja yhtä hyvään lopputulokseen on mahdollista päästä monella eri tavalla. Kehys kannattaakin valita olemassa olevan kokemuksen pohjalta, jolloin opetteluun käytetty aika pienenee. Lisäksi tulee miettiä, onko kehykseen investointi ja kehyksen opettelu kannattavaa, vai voivatko natiivit ratkaisut olla jopa parempia.

Toisena nousee kysymys siitä, oliko Xamarin oikea ratkaisu sovellusten kehitykseen. Kuten aiemmin mainittiin, niin natiivien ratkaisuiden toteuttaminen olisi tässä tilanteessa voinut olla jopa nopeampaa. Tämä johtuu pitkälti siitä, että Xamarin-alusta oli uusi ja tuntematon ennen projektin aloittamista. Lisäksi projektin toteuttaminen Xamarinilla vaati tarkempaa suunnittelua ja arkkitehtuurin hiomista, jotta alustan hyvistä puolista hyödyttiin täysin. Natiivitoteutuksissa oltaisiin voitu oikaista arkkitehtuurin suunnittelussa aiheuttamatta kohtuuttomia ongelmia sovellusten ylläpidolle. Nyt toteutetut sovellukset ovat joka tapauksessa helpommin ja paremmin ylläpidettävissä sekä jatkokehitettävissä. Kokonaisuuden kannalta pohdittuna Xamarinin valinta oli hyvä päätös eikä sovelluksia olisi ollut kannattava toteuttaa natiiveina.

Kolmantena herää kysymys, olisiko pitänyt käyttää jotain muuta kehystä. Vaikka Xamarin toikin omat haasteensa kehitykselle, ei toistaiseksi ole toista yhtä laajasti käytössä olevaa alustaa, jolla voitaisiin luoda natiiveja Android ja iOS-sovelluksia. Suurin osa kehyksistä, jotka eivät perustu HTML5-tekniikkaan ja WebView-komponentin käyttöön, on suunnattuja pääosin pelien kehitykseen. React Native ja Qt olisivat voineet olla vertailukelpoisia kehyksiä Xamarinin kanssa, mutta myös näissä molemmissa on täysin erilainen lähestymistapa kehitykseen, jonka opettelu olisi varmasti ollut yhtä työlästä. Lisäksi

React Native ei olisi sopinut Evolvitin käyttöön huonon Windows-käyttöjärjestelmätuen takia.

Näiden pohdintojen pohjalta uskallan suositella Xamarin-alustaa Evolvitille myös tuleviin mobiilihankkeisiin.

LÄHTEET

Apple Inc, 2012. Model-View-Controller in Cocoa. Luettu 06.10.2016 https://developer.apple.com/library/content/documentation/General/Conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html

Apple Inc, 2015. Model-View-Controller. Luettu 06.10.2016 https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

Blomfelt, E. 2013. Anatomy of a Qt 5 for Android application. Luettu 31.10.2016 http://blog.qt.io/blog/2013/07/23/anatomy-of-a-qt-5-for-android-application

ECMA International, 2012. Standard ECMA-335 Common Language Infrastructure (CLI) 6th edition. Luettu 16.09.2016. http://www.ecma-international.org/publications/standards/Ecma-335.htm

Eisenman, B. 2015. Learning React Native. Iso-Britannia: O'Reilly Media Inc. E-kirja http://www.safaribooksonline.com.

Facebook Inc, 2016. React Native dokumentaatio. Luettu 17.10.2016 http://facebook.github.io/react-native/docs/

Martin Fowler, 2004. Inversion of Control Containers and the Dependency Injection pattern. Luettu 07.10.2016 http://www.martinfowler.com/articles/injection.html

Mono Project, 2016. AOT. Luettu 16.09.2016. http://www.mono-project.com/docs/advanced/aot/

MSDN, 2015. Delegates. Luettu 07.10.2016. https://msdn.microsoft.com/fi-fi/library/ms173171.aspx

Peppers, J. 2014. Xamarin Cross-platform Application Development. Iso-Britannia: Pact Publishing

Peppers, J. 2015. Xamarin Cross-platform Application Development – Second Edition. Iso-Britannia: Pact Publishing. E-kirja http://www.safaribooksonline.com.

Reynolds, M. 2014a. Xamarin Essentials. Iso-Britannia: Pact Publishing. E-kirja http://www.safaribooksonline.com.

Reynolds, M. 2014b. Xamarin Mobile Application Development for Android. Iso-Britannia: Pact Publishing. E-kirja http://www.safaribooksonline.com.

The Apache Software Foundation, 2015. Overview. Luettu 20.10.2016 https://cordova.apache.org/docs/en/latest/guide/overview/index.html

The Linux Information Project, 2005. Cross-platform Definition. Luettu 02.09.2016. http://www.linfo.org/cross-platform.html

The Qt Company, 2016. About Qt. Luettu 20.10.2016 http://wiki.qt.io/About_Qt

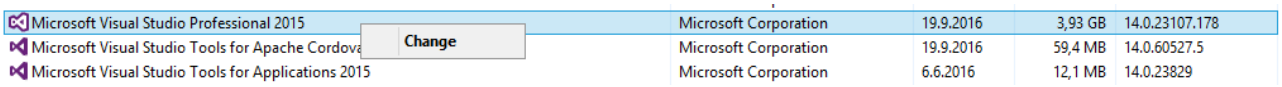
Xamarin Developer, 2016. Xamarin Dokumentaatio. Luettu 09.09.2016 https://developer.xamarin.com/

LIITTEET

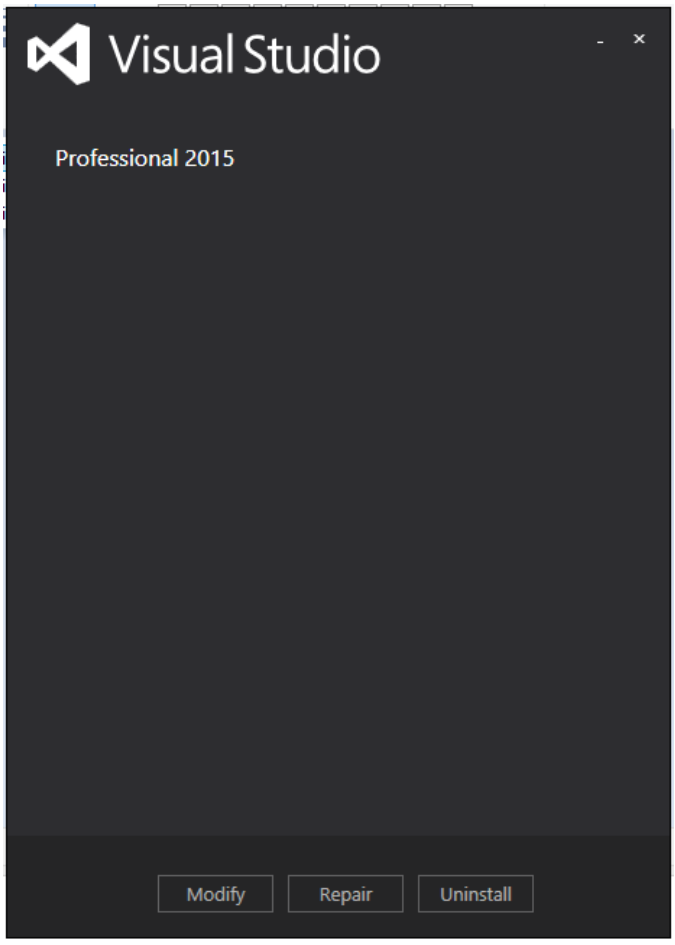
Liite 1. Visual Studion ja Xamarin-lisäosan asennus

1 (2)

Xamarin kehitys vaatii Visual Studio 2013 -version tai uudemman. Mikäli Visual Studiota ei ole asennettu tietokoneelle, tulee Visual Studio ladata Microsoftin latauspalvelusta. Microsoft tarjoaa Visual Studiosta maksuttoman Community-version avoimenlähdekoodin kehittämiseen, yksittäisille kehittäjille ja pienille ryhmille (Microsoft 2016). Mikäli Visual Studio on kuitenkin jo asennettu, niin Xamarin-lisäosan asennus tapahtuu Ohjelmat ja Ominaisuudet -toiminnon (engl. Programs and Features.) kautta, joka löytyy Windowsin ohjauspaneelista (engl. Control Panel.). Listasta valitaan Visual Studio ja oikealla hiiren klikkauksella saadaan Muuta-asetus (engl. Change.) auki (kuva 1). Seuraavasta ikkunasta valitaan Modify-vaihtoehto, joka mahdollistaa laajennusten ja lisäosien asennuksen (kuva 2).

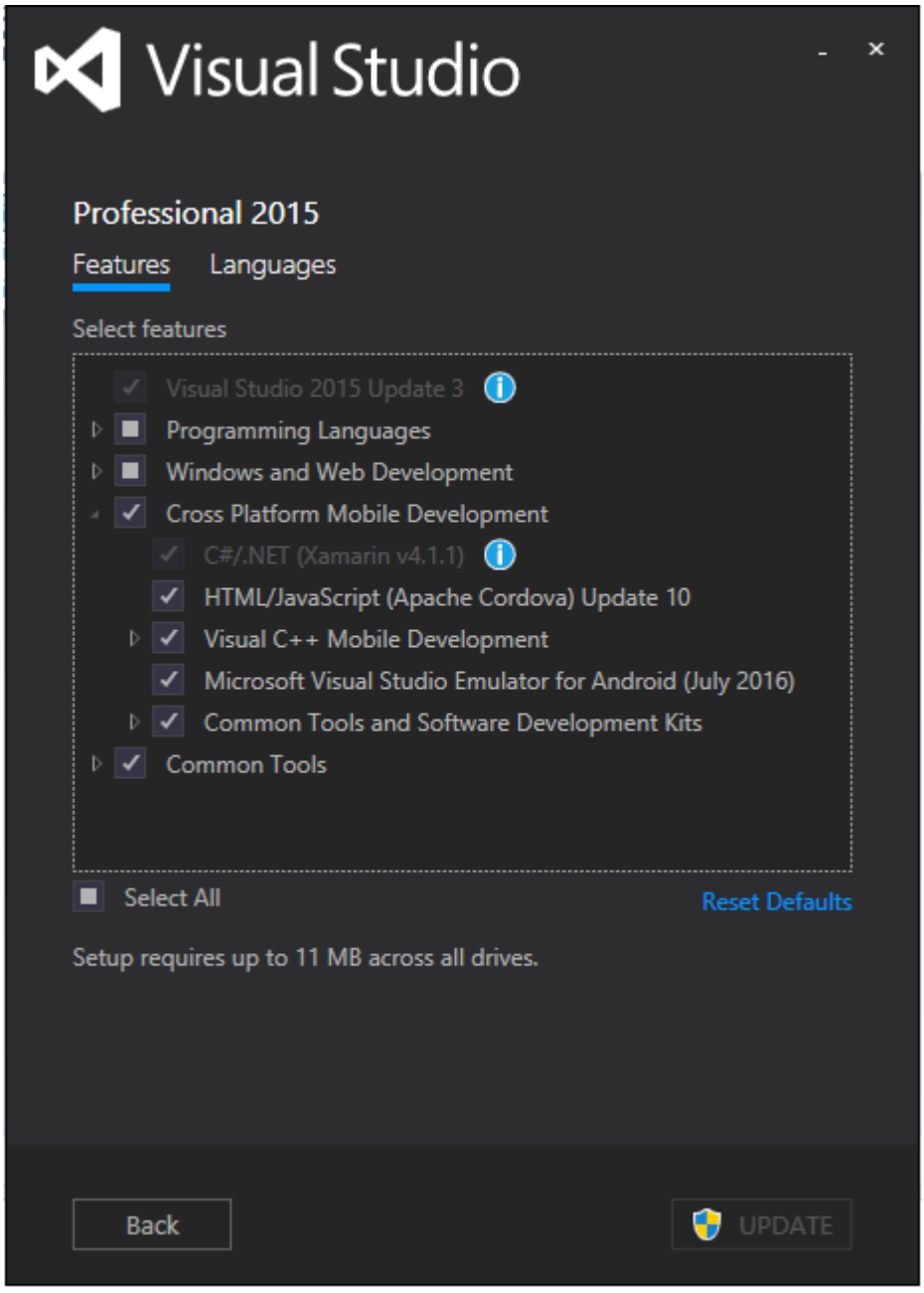


Kuva 1. Visual Studion lisäosien asentaminen Ohejlmat ja Ominaisuudet -toiminnon kautta.



Kuva 2. Visual Studion asennussovellus.

Seuraavassa ikkunassa on lista saatavilla olevista ominaisuuksista ja tästä listasta valitaan Cross Platform Mobile Development -kohdan alta C#/.NET (Xamarin) -valinta (kuva 3). Lisäksi on hyvä asentaa Common Tools and Software Development Kits -vaihtoehdon alta Android-kehitystyökalut ja Microsoft Visual Studio Emulator for Android (kuva 3). Kun valinnat on tehty, painetaan Update- tai Install-painiketta ja valitut ominaisuudet asennetaan Visual Studioon.



Kuva 3. Valinnat Xamarin-lisäosan asentamiseksi.

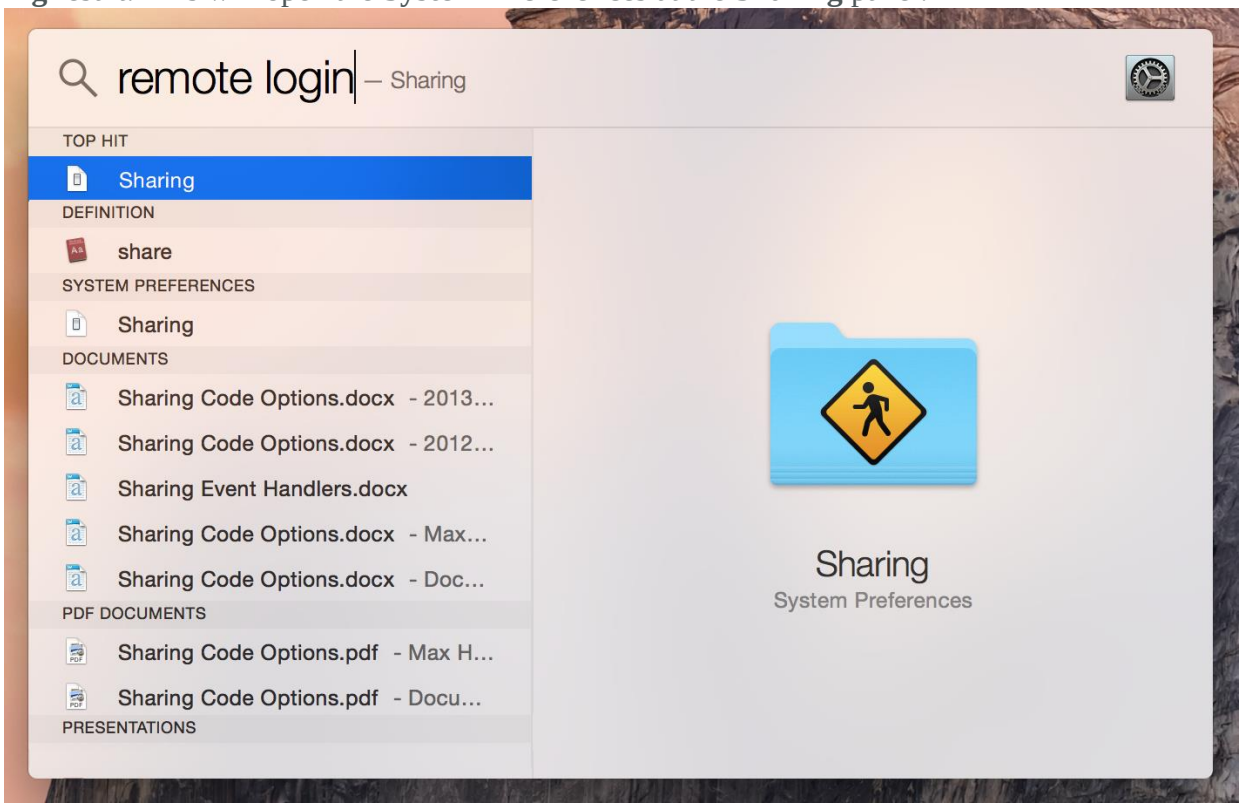
Liite 2. Xamarin konfiguraatio iOS-kehitystä varten
Lähde: https://developer.xamarin.com/guides/ios/getting_started/installation/windows/#Configuring_The_Mac

1 (2)

Configuration

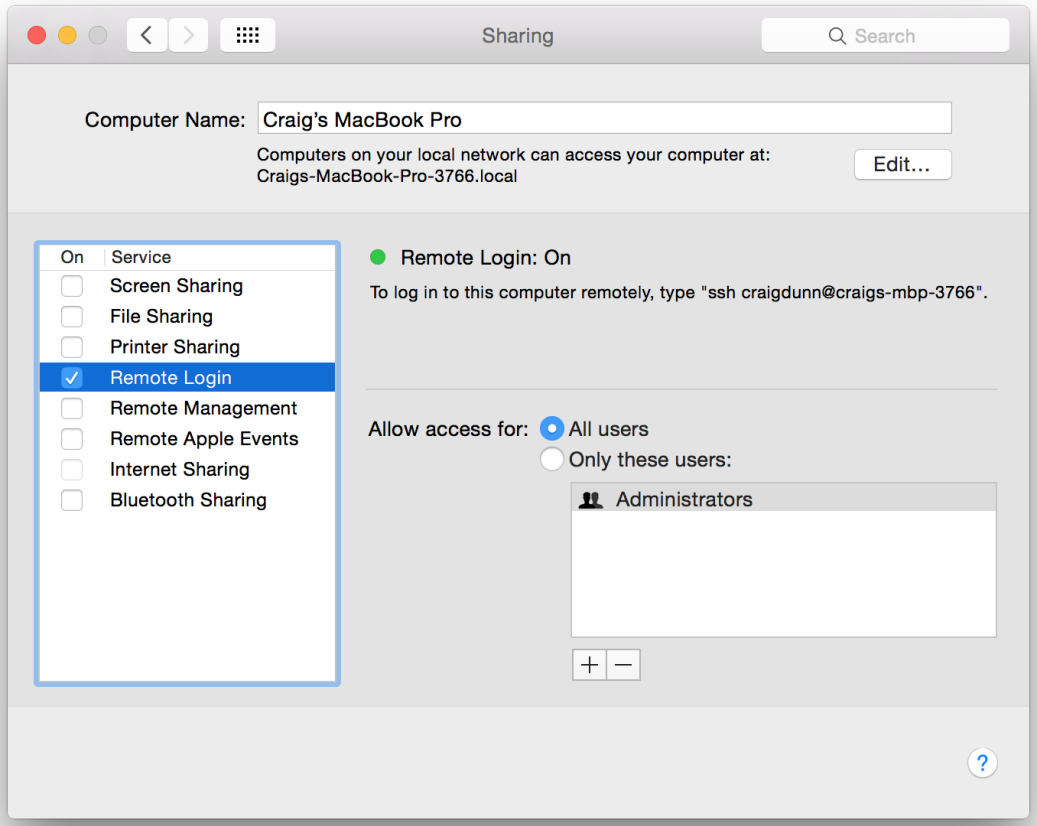
To access communication between the Xamarin extension for Visual Studio and your Mac, you will need to allow **Remote Login** on your Mac. Follow the steps below to set this up:

1. Open *Spotlight* (Cmd-Space) and search for **Remote Login** and then select the **Sharing** result. This will open the **System Preferences** at the **Sharing** panel.



2. Tick the **Remote Login** option in the **Service** list on the left in order to allow Xamarin for Visual Studio to connect to the Mac.

2 (2)



3. Make sure that **Remote Login** is set to allow access for **All users**, or that your Mac username or group is included in the list of allowed users in the list on the right. Your Mac should now be discoverable by Visual Studio if it's on the same network.

   In addition to this, if you have the OS X firewall set to block signed applications by default, you may need to allow mono-sgen to receive incoming connections. An alert dialog will appear to prompt you if this is the case.