Tampereen ammattikorkeakoulu

Xamarin.iOS-sovelluksen julkaisuautomaatio

Riku Kinnunen

OPINNÄYTETYÖ
Kesäkuu 2025

Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

KINNUNEN, RIKU:
Xamarin.iOS-sovelluksen julkaisuautomaatio

Opinnäytetyö 37 sivua, joista liitteitä 8 sivua
Kesäkuu 2025

Tämän opinnäytetyön tarkoituksena oli tutkia nykyaikaisia CI/CD-periaatteita ja niitä hyödyntäen toteuttaa Xamarin.iOS-sovelluksen julkaisuautomaatio. Työn tavoitteena oli kehittää luotettava ja tehokas automatisoitu julkaisuputki, joka yksinkertaistaa ja nopeuttaa mobiilisovelluskehitystä.

Työ jakautui teoriaosuuteen ja käytännön toteutukseen. Teoriaosuudessa tarkasteltiin jatkuvan integraation ja jatkuvan toimituksen periaatteita ja käytäntöjä sekä perehdyttiin Xamarin.iOS-sovelluksen julkaisuprosessiin ja esiteltiin siinä vaadittavat teknologiat ja laitteet.

Käytännön osuudessa toteutettiin julkaisuautomaatioputki sovellukselle käyttäen Fastlane automaatiotyökalua. Ratkaisu integroitiin osaksi projektissa jo käytössä ollutta GitLab CI/CD -alustaa. Toteutukseen lisättiin myös sovelluksen allekirjoitusavainten ja jakeluprofiilien hallintaa helpottavia ominaisuuksia.

Tuloksena saatiin toimiva automaattinen julkaisuputki, joka vähensi merkittävästi manuaalisen työn määrää sovelluksen julkaisuprosessissa. Ratkaisu paransi kehittäjätyytyväisyyttä ja mahdollisti sovelluksen nopeammat ja luotettavat julkaisut.

Asiasanat: xamarin.ios, julkaisuautomaatio, jatkuva integraatio, jatkuva toimitus

ABSTRACT

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

KINNUNEN, RIKU:
Xamarin.iOS Application Release Automation

Bachelor's thesis 37 pages, appendices 8 pages
June 2025

The purpose of this thesis was to research modern CI/CD-principles and implement a release automation pipeline for a Xamarin.iOS application. The objective was to develop a reliable and efficient automated release pipeline that simplifies and speeds up the mobile application development process.

The work was divided into theoretical and practical sections. The theoretical part examined the principles and processes of continuous integration and continuous delivery, the Xamarin.iOS release process and the required technologies and devices.

The practical section covered the implementation of an automated release pipeline using the Fastlane automation platform. The solution was integrated into the project's existing GitLab CI/CD platform. The implementation also included features to simplify the management of the signing keys and distribution profiles of the application.

The result was an automated release pipeline that significantly decreased the amount of manual work in the application release process. The solution improved developer satisfaction and enabled faster and more reliable application releases.

Key words: xamarin, ios, release automation, continuous integration, continuous delivery

SISÄLLYS

LYHENTEET JA TERMIT

AOT	Ahead-Of-Time, ohjelmakoodi käännetään konekielelle etukäteen, ennen ohjelman ajoa
CD	Continuous Delivery, jatkuva toimitus
CI	Continuous Integration, jatkuva integraatio
DevOps	ohjelmistokehityksen toimintamalli, jossa yhdistetään kehitys (Development) ja operointi (Operations). Tarkoituksena tehostaa sovelluksen kehitystä, testausta, julkaisua ja ylläpitoa
DevSecOps	jatkokehitetty malli, jossa DevOpsiin lisätään tietoturva (Security). Tarkoituksena integroida tietoturvan ylläpitö kehitysprosessin vaiheisiin
ipa	iOS App Archive, iOS-sovelluksen asennuspaketti
JIT	Just-In-Time, ohjelmakoodin ajonaikaisesti tapahtuva käännös
MSIL	Microsoft Intermediate Language, korkean tason välikieli, jolle .NET-sovellukset käännetään ennen ajonaikaista suorittamista
SDK	Software Development Kit, ohjelmistokehityspaketti, jonka avulla voidaan kehittää sovelluksia tietylle alustalle, järjestelmälle tai palvelulle

1 JOHDANTO

Teknologian ja digitalisaation kehitys nostaa sovelluskehityksen tärkeyttä ja merkitystä sekä käyttäjille että yrityksille. Erityisesti mobiilisovelluskehityksen merkitys on kasvanut avainasemaan. Älylaitteiden määrä lisääntyy myös vauhdilla ja yhä useampi henkilö käyttää pääasiallisena laitteenaan mobiililaitetta. Internetin selaamisesta lähes kaksi kolmasosaa tapahtuu nykyään puhelimilla. Mobiililaitteita käytetään yhteydenpitoon, viihteeseen, sekä jokapäiväisien asioiden hoitoon, niin töissä kuin vapaa-ajallakin. Tämä nostaa sekä kysyntää että tarvetta tarjota korkealuokkaisia, käyttäjäystävällisiä mobiilisovelluksia.

Nykyaikaisessa sovelluskehityksessä haetaan yhä parempaa tehokkuutta, helppokäyttöisyyttä, kehitysnopeutta sekä laajaa laitetukea. Ne mahdollistavat suurimman mahdollisen käyttäjämäärän, -tyytyväisyyden sekä nopeuttavat sovelluksen julkaisua. Tuettavista käyttöjärjestelmistä Android sekä Applen iOS kattavat yli yhdeksänkymmentäyhdeksän prosenttia laitteista. Alustarajat ylittävät teknologiat, kuten Xamarin ja React Native, mahdollistavat yhden koodipohjan käyttämisen molempien sovellusten kehittämiseen, näin nopeuttaen kehitystä sekä sen ylläpitoa. Näiden teknologioiden kehittyessä, sovelluksen julkaisuprosessi on edelleen aikaa vievä prosessi, jossa on merkittävä osa manuaalista työtä.

Julkaisuautomaation tehtävä on mahdollistaa sovelluksen siirtymä kehitysvaiheesta tuotantoympäristöön sujuvaksi, helposti toistettavaksi ja virheettömäksi. Tämän lisäksi se säästää kehittäjien aikaa ja parantaa kehittäjäkokemusta vähentämällä manuaalista, toistuvaa työtä. CI/CD-prosessien sekä automaatiotyökalujen, kuten Fastlane, GitHub Actions sekä GitLab CI, tarjoavat ratkaisuja ja alustoja näiden automaatioiden toteuttamiseen. Ne mahdollistavat sovelluksen automaattisen kääntämisen, testauksen sekä julkaisun alustakohtaisiin sovelluskauppoihin.

Tässä opinnäytetyössä perehdytään mobiilisovellusten julkaisuprosessiin, sen automatisointiin sekä siihen tarjolla oleviin työkaluihin. Työssä käydään läpi Xamarin.iOS teknologialla kehitetyn sovelluksen julkaisun työvaiheet sekä esitel-

lään GitLab CI:llä ja Fastlanella toteutettu automatisoitu julkaisuprosessi. Sen lisäksi tarkastellaan miten julkaisuprosessin automatisointi vaikuttaa kehitysprosessin nopeuteen, luotettavuuteen sekä kehittäjäkokemukseen.

2 CI/CD MOBIILIKEHITYKSESSÄ

2.1 Yleiskuva

Jatkuva integraatio (CI) ja jatkuva toimitus (CD) ovat DevOps-prosesseja, joiden on tarkoitus nopeuttaa sovelluskehitystä ja julkaisua, parantaa koodin laatua sekä vähentää manuaalista työtä. Tämä mahdollistetaan automatisoimalla koodimuutosten integrointi, testaus ja sovellusten jakelu. (Red Hat 2023)

Jatkuva integraatio tarkoittaa koodimuutosten yhdistämistä keskeiseen repositorioon ja haaraan säännöllisesti, jopa useita kertoja päivässä. Se mahdollistaa sovellusten laadunparantamisen, julkaisusyklin nopeuttamisen ja tuottavuuden nousun. Tämä saadaan aikaan automatisoimalla sovelluksen kääntäminen ja testaaminen. (Shahin, Ali Babar, Zhu 2017.)

Jatkuva toimitus vie jatkuvan integraation periaatteen askeleen pidemmälle. Jatkuvan toimituksen prosessissa sovellus pidetään julkaisuvalmiissa tilassa. Se saavutetaan automatisoimalla jatkuvan integraation prosessien lisäksi sovelluksen julkaisuprosessi. (Laukkanen, Itkonen, Lassenius 2017.)

2.2 Huomioitavat haasteet

Suurimpia haasteita mobiilikehityksessä on pirstoutuminen (Ahmad ym. 2019) Sitä tapahtuu monella tasolla. Käyttöjärjestelmiä on useita ja niillä on jokaisella hieman eri rajapinnat ja toiminnot. Eri käyttöjärjestelmäversioita on myös käytössä yhdellä hetkellä monia, joka voi johtaa siihen, että samasta sovelluksesta joudutaan julkaisemaan samalle alustalle useampi versio. Erilaisia laitteita on valtava määrä. Sovelluksen toiminta pitää voida taata monella eri ruutukoolla ja suorituskyvyllä. (Asfour ym. 2019). Kaikki tämä vaikeuttaa sovelluksen kehitystä ja kattavaa testaamista.

Mobiilisovelluksia jaetaan eri alustoille useassa sovelluskaupassa. Eri sovelluskaupoille on oma julkaisuprosessinsa, jotka tuovat vaatimuksensa sovelluksen

jakeluprosessiin. Esimerkiksi iOS-alustalle julkaistava sovellus vaatii macOS-alustan sovelluksen kääntämiseen. Myös sovellusten allekirjoitustavoissa on eroja alustojen välillä. Julkaisuprosessiin kuuluu myös manuaalisia tarkistus- ja hyväksyntävaiheita, jotka tuovat pidentävät sovelluksen julkaisuun kuluvaa aikaa. Vain yhdeksän prosenttia tutkituista sovelluksista käytti CI/CD-ratkaisuja julkaisuprosessissa (Ghaleb, Abduljalil & Hassan 2025).

Laadunvarmistamiseksi kattava testaaminen on välttämätöntä, mutta mobiilisovelluksille sen toteuttaminen on hankalaa, johtuen lukuisista tuettavista laitteista ja alustoista. Mobiilitestaukseen vaikuttavia seikkoja ovat mobiiliyhteydet, laitteiden ja alustapirstoutuminen sekä kosketuskäyttöliittymät. Testauksessa pitää myös ottaa huomioon sovelluksen resurssien- ja virrankulutus. (Kirubakaran & Karthikeyani 2013.) Käytännössä kattavaa testausta pelkästään fyysisiä laitteita käyttäen on mahdotonta ja avuksi on otettava emulaattoreita. Cruzin, Abreun & Lon (2019) mukaan 41 % tutkituista sovelluksista sisälsi testejä. Yksikkötestejä oli 39 % sovelluksista, mutta käyttöliittymätestejä vain 15 %. Sovellusten testaaminen on yleistynyt huomattavasti, mutta silti suurin osa sovelluksista ei käytä automaattitestausta. Tutkimuksessa havaittiin, että vain 26 % sovellusprojekteista käytti CI/CD-prosesseja ja työkaluja. (Cruz, Abreu & Lo 2019.)

Mobiilisovellusten CI/CD-putkia määriteltäessä täytyy ottaa kaikki edellä mainitut seikat huomioon. Niiden täytyy huomioida eri alustojen vaatimukset sovelluksen kääntämiselle, testaamiselle, allekirjoittamiselle ja jakelulle. Tämä nostaa putkien kompleksisuutta ja eri alustoille täytyy usein rakentaa oma putkensa, joka nostaa implementoinnin työmäärää.

2.3 CI/CD-automaation edut

Automatisoimalla sovelluksen julkaisuprosessi CI/CD-työkaluja käyttäen saavutetaan useita etuja. Prosessi parantaa koodin laatua, nopeuttaa julkaisuaikaa, vähentää manuaalista työtä ja lisää kehittäjä- ja asiakastyytyväisyyttä. (Cruz, Abreu & Lo 2019; Bhimanapati & Goel, 2023).

CI/CD:n käyttöönoton jälkeen Bhimanapatin & Goelin (2023) tutkimuksessa sovelluksen muutosten läpimenoaika väheni huomattavasti, viikoista päiviin. Kaikissa tutkituissa ryhmissä tyytyväisyys nousi, erityisesti kehittäjien ja loppukäyttäjien keskuudessa. Sovelluksen julkaisutahti myös kiihtyi huomattavasti. Ennen 75 % julkaisuista oli kuukausittain ja jälkeen 85 % joko päivittäin tai viikoittain. (Bhimanapati & Goel 2023.) Sovelluksen julkaisutahti myös vaikuttaa käyttäjätyytyväisyyteen, useimmin päivitetyt sovellukset saavat vähemmän negatiivisia arvosteluita ja ovat kategorioissaan suositumpia (McIlroy, Ali, Hassan 2015).

Bernardo, Da Costa, Kulesza, & Treude (2023) eivät tutkimuksessaan havainneet, että CI-automaation lisääminen projektiin olisi vaikuttanut julkaisuiden tiheyteen. Huomattavaa kuitenkin oli, että automaation lisäämisen jälkeen jokaisessa julkaisussa oli 3,43 kertaa enemmän pull requestejä. (Bernardo, Da Costa, Kulesza, & Treude 2023).

Rahman, Agrawal, Krishna & Sobran (2018) raportoivat, että commitien määrä ja koko kasvoivat merkittävästi avoimen lähdekoodin projekteissa CI:n implementoinnin jälkeen. Sama havainto tehtiin tehtävien ja bugikorjausten valmistumisen määrissä. Tiimin jäsenten välinen yhteistyö kasvoi molemmissa verrokkiryhmissä, avoimen lähdekoodin projekteissa enemmän. Löydökset viittaavat siihen, että saadaakseen odotetun hyödyn irti CI:n käytöstä, vaaditaan myös kehitystiimiltä sitoutumista CI:n käytäntöihin, kuten tiheään commit-tahtiin. (Rahman, Agrawal, Krishna & Sobran 2018).

CI/CD-prosessien käyttöönoton jälkeen sovelluksen epäonnistuneiden käyttöönottojen määrä väheni. Käyttöönottojen määrä nousi, joka samalla lisäsi automaattitestien ajokertoja. Kvalitatiivisesta perspektiivistä tarkastellen, kehittäjien henkinen kuorma väheni automaation vähentäessä manuaalisia työvaiheita. Jatkuvan integraation ja toimituksen käytännöt vähentävät kehittäjien kognitiivista kuormaa ja nopeuttavat nopeamman palautekierteen. (Fluri, Fornari, Pustulka 2023.)

3 KÄYTETYT TEKNOLOGIAT, TYÖKALUT JA LAITTEET

3.1 macOS

Alun perin Mac OS on Applen kehittämä UNIX-pohjainen käyttöjärjestelmä. Se on julkaistu alun perin vuonna 1984, kun yhtiön Macintosh-laitteet tulivat markkinoille. Vuonna 1997 Apple osti NeXT Software Inc., jonka NeXTSTEP-käyttöjärjestelmään pohjautuen Apple julkaisi päivitetyn käyttöjärjestelmän vuonna 2001 Mac OS X -nimellä. Vuodesta 2016 eteenpäin se on tunnettu nimellä macOS. (Encyclopædia Britannica n.d.)

Sovellusten julkaisu Applen käyttöjärjestelmille ja laitteille vaatii macOS-käyttöjärjestelmän. Sovelluksen kääntäminen ja allekirjoittaminen käyttää Xcode komentorivityökaluja. Komentorivityökalut voidaan asentaa joko yksittäisenä pakettina tai Xcode-sovelluksen asennuksen mukana. Xcode ja komentorivityökalut ovat saatavilla vain macOS-käyttöjärjestelmälle (Xcode - Support - Apple Developer n.d.). Tästä syystä sekä sovelluksen kehittämiseen käytetty, että sovelluksen kääntämistä varten konfiguroitu, GitLabiin yhdistetty, tietokone olivat macOS-pohjaisia.

3.2 iOS

iOS on Applen mobiililaitteille kehittämä käyttöjärjestelmä, joka esiteltiin vuonna 2007 ensimmäisen iPhonen julkaisun myötä. iOS, kuten myös macOS, pohjautuu Darwiniin, joka on Applen avoimen lähdekoodin Unix-kaltainen käyttöjärjestelmä. iOS toimii pohjana muiden Applen pienempien laitteiden käyttöjärjestelmille, kuten iPadOS, WatchOS ja tvOS. (Volle 2025.)

Mobiilikehittäjän näkökulmasta iOS on merkittävä alusta. Vaikka globaalisti sen markkinaosuus on hieman alle kolmekymmentä prosenttia, iOS-käyttäjät kattavat lähes 70 % sovellusostoista (Howarth 2025). Koska vain Apple valmistaa iOS-laitteita, huomioonotettavien laitekonfiguraatioiden ja eri näyttökokojen määrä on myös pienempi.

3.3 Xamarin

Xamarin on ohjelmistokehitysalusta, joka antaa kehittäjille mahdollisuuden luoda sovelluksia Android, iOS, Windows ja macOS käyttöjärjestelmille käyttäen yhtä koodipohjaa, joka on kirjoitettu C# ohjelmointikielellä ja hyödyntää .NET-ohjelmistokehystä. Xamarin on abstraktiokerros, joka hallitsee kommunikointia jaetun koodin sekä alustakohtaisen koodin välillä. Jopa 90 % koodista on mahdollista jakaa alustojen välillä. (Johnson ym. 2020.)

Xamarin pohjautuu avoimen lähdekoodin Mono-projektiin. Mono on Ximianin 2001 alkaen kehittämä ohjelmistokehys, jonka tarkoitus oli luoda Microsoftin .NET-kehyksestä avoin versio Linuxille (About Mono n.d.). Ensimmäinen versio Monosta julkaistiin vuonna 2004. Alkuperäisestä suunnitelmasta poiketen, Monon tukemien alustojen määrä laajeni huomattavasti. Mono mahdollistaa .NET-kehyksellä luotujen sovellusten ajamisen eri käyttöjärjestelmillä ja alustoilla, kuten Linuxilla, mobiililaitteilla ja pelikonsoleilla (Supported Platforms n.d.). Monoa käyttäen kehitettiin MonoTouch sekä Mono for Android projektit, jotka ovat Monon sovelluksia mobiililaitteilla. Xamarinin perustamisen jälkeen näiden projektien kehitystä jatkettiin ja niistä muodostuivat Xamarin.iOS ja Xamarin.Android. Vuonna 2016 Microsoft osti Xamarinin ja sitoutui jatkamaan sen integroimista vahvemmin omiin tuotteisiinsa, kuten Visual Studioon (Guthrie 2016). Microsoft lopetti Xamarinin virallisen tuen toukokuussa 2024, jonka jälkeen esimerkiksi uusia Android- ja iOS-versioita ei tueta. Xamarinilla kehitettyjä sovelluksia kehotetaan siirtymään Microsoftin .NET MAUI -ympäristöön. (Xamarin Support Policy 2024.)

Sovelluskehityksessä Xamarinin etuina on jaettu koodipohja, joka mahdollistaa liiketoimintalogiikan koodin kirjoittamisen vain kerran. Lisäksi voidaan käyttää C#-kirjastoja, kuten .NET:n lukuisia Microsoftin kehittämiä ja ylläpitämiä, sekä kolmannen osapuolen tekemiä. Natiiveja kirjastoja on mahdollista käyttää Xamarin-sovellusten kehityksessä linkityksen kautta (Strakh 2020a, 2020b). Myös sovelluksen palvelimet voidaan kehittää käyttäen .NET-ohjelmistokehystä, mahdollis-

taen samojen kehittäjien työskentelyn sekä mobiilisovellusten, että niiden käyttämien palvelimien kehityksessä. Xamarin on myös vahvasti sidottu mukaan Microsoftin Visual Studio -kehitysympäristöön. (Johnson ym. 2020.)

Xamarinin heikkouksia natiiviin mobiilisovelluskehitykseen verrattuna on sovellusten suurempi koko. Tämä johtuu siitä, että sovelluspaketissa on mukana Mono-ajoympäristö sekä .NET Base Class -kirjastot (BCL). Sovelluspaketin kokoa on mahdollista pienentää ottamalla Linker käyttöön. Linker analysoi koodin ja jättää loppullisesta paketista käyttämättömät kirjastot ja tyypit pois. Kokoa voi myös pienentää optimoimalla sovelluksen käyttämiä resursseja, esimerkiksi pakkaamalla kuvatiedostot. Vaikka Xamarin-sovellukset kirjoitetaan C#-kielellä, vaatii niiden kehittäminen silti alustaspesifiä osaamista, koska esimerkiksi käyttöliittymät rakennetaan käyttäen kyseisten alustojen natiivikomponentteja. Lisäksi uusien käyttöjärjestelmäversioiden ja laiterajapintojen tuki tulee Xamariniin saataville hieman myöhemmin. (Vidjikant 2024.)

3.4 Xamarin.iOS

Xamarin.iOS on Xamarin-kehyksen toteutus Applen iOS-alustalle. Xamarin.iOS tarjoaa pääsyn iOS:n natiiveihin rajapintoihin, kuten UIKitiin, CoreGraphicsiin ja CoreDataan. Tämä mahdollistaa esimerkiksi käyttöliittymien rakentamisen täysin natiiveja komponentteja käyttäen.

Muilla alustoilla Xamarinin käyttää Just-in-Time (JIT) -kääntämistä, eli koodi käännetään ensin Microsoft Intermediate Languageksi (MSIL), joka sovellusta käynnistäessä käännetään natiiviksi koodiksi. Apple estää dynaamisesti generoidun koodin suorittamisen iOS-alustalla, joka sulkee pois JIT-käännöksen käytön. Tämän takia Xamarin.iOS käyttää Ahead-of-Time (AOT) -kääntämistä. Ensin Monon C#-kääntäjä kääntää koodin Microsoft Intermediate Languageksi (MSIL), joka käännetään natiiviksi iOS-binääriksi, jota voidaan ajaa Applen ARM-pohjaisella prosessorilla. (Ortinau ym. 2017b.) Kääntämisessä on myös mahdollista ottaa käyttöön optimoiva LLVM-kääntäjä, joka tuottaa nopeammin toimivaa koodia, mutta kääntäminen kestää kauemmin. Kehitystyössä suositellaan käytettävän

normaalia käännöstä ja julkaisua tehdessä optimoivaa LLVM-kääntäjää. (Ortinau ym. 2017a.)

3.5 MSBuild

Microsoft Build Engine, eli MSBuild on Microsoftin kehittämä alusta sovellusten kääntämiseen. Se tarjoaa XML skeeman, jolla määritellään miten alusta kääntää sovelluksen. Visual Studiossa projektitiedostot, esimerkiksi .csproj-tiedostot, sisältävät XML-koodia, joka ajetaan MSBuildilla projektia käännettäessä. Tiedostossa voi määritellä muun muassa eri profiilit sovelluksen kääntöön kehitystä ja julkaisua varten. Julkaisua varten olevassa Release-profiilissa kannattaa esimerkiksi käyttää sovelluksen kokoa ja suorituskykyä parantavaa LLVM-optimointia. Kehitysvaiheessa sen käyttö pidentää sovelluksen kääntämiseen kuluvaa aikaa. (Ortinau ym. 2017c; MSBuild 2024).

Xamarin käyttää sovelluksen kääntämiseen MSBuildia. Sitä voi käyttää joko Visual Studion kautta tai komentoriviltä komennoilla msbuild.exe tai dotnet build. Parametreinä sille voidaan antaa projektitiedosto tai useamman projektin sisältävä Visual Studio solution (.sln) -tiedosto. Mikäli tiedostoa ei määritellä, etsii MSBuild kansiosta proj-päätteisen tiedoston. Tiedoston lisäksi sille voidaan antaa lippuparametreja, joilla voidaan esimerkiksi valita mitä .csproj-tiedostossa määriteltyä profiilia ja alustaa sovelluksen kääntämiseen käytetään. (MSBuild command-line reference 2024; Build properties for iOS... 2025).

3.6 GitLab

GitLab on web-pohjainen DevSecOps-alusta. Sen tarkoituksena on tehostaa sovellusten kehitystä, jakelua ja turvallisuutta. GitLabin ominaisuuksiin kuuluu muun muassa

- lähdekoodien hallinta ja koodikatselmoinnit
- CI/CD-putkien luonti ja hallinta
- koodin analyysi ja turvallisuustestaus

- sovelluksen suunnittelutyökalut, kuten tiketöinti ja iteraatioiden seuranta. (GitLab Features n.d.).

GitLab tarjoaa kolme eri palvelutasoa, ilmaisesta alkaen. Maksullisilla tasoilla tarjolla on tekoälyominaisuuksia, kuten koodin ja testien generointi. Alusta tarjoaa myös kolme eri hallintavaihtoehtoa. Palvelua voi käyttää GitLab.comin kautta, heidän ylläpitämänsä dedikoidun palvelininstanssin kautta tai sen voi asentaa ja ylläpitää omalla palvelimellaan. GitLabilla on maaliskuuhun 2025 mennessä arviolta yli 50 miljoonaa rekisteröityä käyttäjää. (About GitLab, 2025; GitLab Pricing n.d.)

Opinnäytetyössä käsiteltävän sovelluksen hallintaan käytettiin omalla palvelimella ajettavaa GitLab-instanssia. GitLabissa oli myös muiden tiimin sovellusten repositoriot sekä niille rakennetut CI/CD-putket. Joten luonnollista oli integroida mobiilisovelluksen julkaisuautomaatio GitLabiin.

3.7 Fastlane

Fastlane avoimen lähdekoodin työkalu mobiilisovellusten julkaisun automatisointiin. Sitä voidaan käyttää

- sovelluksen kääntämiseen
- sovelluksen allekirjoittamiseen ja allekirjoitusavainten hallintaan
- testien ajamiseen
- kuvankaappausten ottamiseen sovelluskauppaa varten
- testi- ja tuotantoversioiden jakeluun. (Fastlane docs, 2025.)

Fastlanen työnkuluista käytetään nimitystä lane. Ne määritellään Fastfile-tiedostossa, jossa jokainen lane voidaan koota Fastlanen tarjoamista ja kehittäjän itse määritellyistä toiminnoista. Fastlanen tarjoamia toimintoja ovat esimerkiksi sovelluksen jakelu Google Play Storeen sekä Applen App Storeen. Fastlane on myös rakennettu helposti eri CI/CD-alustoihin integroitavaksi.

Käsitellyn sovelluksen automaatioon valittiin Fastlane sen hyvän GitLabin CI/CD-integraation, laajan ja selkeän dokumentaation sekä helpon muokattavuuden ja

laajennettavuuden takia. Natiiviin iOS-sovellukseen verrattuna Xamarin-sovelluksen kääntäminen käyttää eri työkaluja. Fastlanen muokattavuus mahdollisti näiden kääntötyökalujen yhdistämisen muuhun jakeluputkeen. Fastlanea on myös mahdollista ajaa lokaalisti, joka nopeuttaa automaation testaamista ja iterointia.

3.8 Apple Developer -portaali

Apple Developer -portaali on web-pohjainen hallintatyökalu kehittäjille. Sen kautta hallitaan kehittäjätiimin jäseniä ja oikeuksia sekä sovellusten allekirjoittamiseen ja jakeluun vaadittavia sertifikaatteja.

App Store Connect on Developer-portaalin alla toimiva hallintatyökalu. Sen kautta kehittäjät voivat hallita Applen alustoille julkaistavia sovelluksia. Sen kautta määritellään esimerkiksi sovelluksen tiedot, hinnat sekä TestFlight ja App Store julkaisut. App Store Connect tarjoaa myös analytiikkaa, esimerkiksi sovellusten latausmääristä sekä suorituskyvystä ja käyttäjä- ja myyntitilastoja. Sen kautta hallitaan myös verotustietoja ja hyväksytään sopimukset ja käyttöehdot, jotka vaaditaan sovelluksen jakeluun, kuten Apple Developer Program Licence Agreement.

Developer-portaali on pakollinen osa iOS-alustan sovellustenjakelua, jotta sovelluksen saa Applen App Storeen. Käyttäjän pitää liittyä ja Applen maksulliseen kehittäjäohjelmaan, Apple Developer Programiin, jotta saa pääsyn Developer-portaaliin (Membership Details n.d.).

Projektissa Developer-portaalia sekä App Store Connectia käytettiin sovelluksen tietojen hallintaan ja sen jakeluun TestFlightissa ja App Storessa. Portaalin kautta hallittiin myös sovellusten jakelu- ja allekirjoitussertifikaatteja. Automatisoinnin myötä portaaleja siirryttiin käyttämään kutsumalla niiden rajapintoja Fastlanen kautta.

3.9 package_ipa_default.sh -skripti

Xamarin ei tarjonnut virallista tukea Swift-bindingeille, joten alun perin Swiftillä kirjoitettuja kirjastoja käyttäessä piti varmistaa, että sovelluspakettiin tulee mukaan SwiftSupport-kansio, jossa on oikea Swift-ajoympäristö sekä sen standardikirjastot.

Package_ipa_default.sh (Teixeira, 2019) on alunperin BQ:n julkaisemasta skriptistä Lucas Teixeiran päivittämä versio. Skriptille ajetaan macOS-alustalla ja sille annetaan parametrina .app-sovellustiedosto sekä uuden .ipa-tiedoston sijainti. Skripti paketoi sovelluksen ja sisällyttää .ipa-tiedostoon tarvittavan SwiftSupport-kansion.

4 JULKAISUAUTOMAATION TOTEUTUS

4.1 Alkutilanne

Projektin Android- ja iOS-sovellukset oli julkaistu ja aktiivisessa jakelussa sovelluskaupoissa. GitLabissa oli määritelty pipeline, jossa käännettiin Android- ja iOS-projektit, jonka jälkeen ajettiin niiden testit. GitLabiin oli yhdistetty sekä Windows että macOS laitteet, joille oli asennettu GitLab Runner, joka mahdollisti niiden käytön CI/CD-putkissa. Android-sovellukselle oli lisäksi tehty testi- ja julkaisuversioiden luonti. iOS-julkaisut tehtiin täysin manuaalisesti.

iOS-sovelluksen jakelu- ja allekirjoitussertifikaattien hallinta oli myös manuaalinen prosessi. Ne luotiin, tarvittaessa uusittiin ja ladattiin Applen Developer-portaalin Certificates, Identifiers & Profiles -osion kautta.

4.2 Fastlanen asennus

Fastlanen asennus suositellaan tehtäväksi Bundlerin kautta. Bundler on Ruby-paketeille tarkoitettu riippuvuuksienhallintatyökalu. Vaaditut riippuvuudet määritellään Gemfile-tiedostossa. Projektin repositorion juureen lisättiin Gemfile-tiedosto (Liite 1). Tiedostoon määriteltiin riippuvuudeksi Fastlane ja sille lähteeksi rubygems.org. Määrittelyn jälkeen Fastlane asennettiin komennolla bundle install.

4.3 Fastlanen App Store Connect -autentikaatio

Fastlanelle piti määritellä yhteys App Store Connectiin, jotta se voi hallita sovelluksen allekirjoitusavaimia ja jakeluprofiilea ja jotta se pystyi lataamaan sovelluspaketit TestFlight- ja App Store -jakeluihin. Autentikaation toteutukseen on kaksi vaihtoehtoa, Fastlanelle voi joko antaa Apple ID -tunnuksen ja salasanan tai luoda App Store Connect API -avaimen. Eroina on, että tunnuksen käyttö vaatii

kaksivaiheisen tunnistautumisen käyttöä ja kaikki toiminnot eivät ole käytettävissä API-avaimella. Projektissa otettiin käyttöön autentikointi käyttäen API-avainta, koska toteutetut toiminnot eivät vaatineet tunnuksen käyttöä. Sillä saatiin myös karsittua pois kaksivaiheinen tunnistautuminen, joka pitäisi tehdä joko manuaalisesti joka kerta tai tallentaa Fastlanen käytettäväksi manuaalisesti autentikoitu sessio.

API-avain luotiin App Store Connect -portaalin kautta. Etusivulta navigoitiin Users and Access -osioon, jonka jälkeen valittiin Integrations-välilehti. Sivuvalikosta valittiin App Store Connect API ja luotiin Team Key, jolle annettiin nimi ja käyttöoikeustasoksi Developer. Sen jälkeen ladattiin API key -tiedosto. Tiedoston lataus on mahdollista vain sillä hetkellä, mikäli sivu päivittyy, lataus ei ole enää saatavilla. Tässä vaiheessa otettiin myös talteen sivulla näkyvä Issuer ID, jota tarvittiin myöhemmässä vaiheessa.

Jotta Fastlane osaa käyttää avaimen tietoja autentikointiin, täytyy sille antaa avaimen sisällön lisäksi aiemmin mainittu Issuer ID ja avaimen ID. Nämä annettiin JSON-tiedostossa, jonka formaatti näkyy liittessä 2. Key ID on App Store Connect API -avaimen ID, joka näkyy samassa paikassa, jossa avain luotiin. Issuer ID näkyy samalla sivulla. Koska App Store Connect API:n kautta Fastlane ei saanut tietoa onko sinne rekisteröity tiimin tili Apple Developer vai Enterprise -ohjelmassa, tieto annettiin JSON-tiedoston ”in-house”-arvona, jossa false tarkoittaa App Storea. Enterprise-tilillä ei saa julkaistua sovelluksia App Storeen. Luotua JSON-tiedostoa voitiin myöhemmin käyttää Fastlanen laneja määritellessä. Tiedosto tallennettiin sekä kehittäjätiimin käytössä olleeseen Bitwarden-salaisuuksienhallintaohjelmaan että GitLabin CI/CD-muuttujiin, josta se on CI/CD-putken saatavilla. Näistä tarkemmin salaisuuksienhallinta-osiossa.

4.4 Allekirjoitusavainten ja jakeluprofiilien hallinta

Allekirjoitusavainten ja jakeluprofiilien hallintaan valittiin Fastlanen match-työkalu. Matchin kautta hoidettiin sertifikaattien luominen, säilytys sekä niiden uusiminen. Sen käyttöönotto alkaa komennolla fastlane match init, joka suoritettiin

projektin hakemistossa. Ensin valittiin missä sertifikaatit säilytetään. Vaihtoehtoina on Git-repositorio ja Google Cloud tai Amazon S3 -pilvipalvelut. Koska mobiili- ja palvelinsovellusten lähdekoodien hallinta tapahtui GitLabin kautta, oli luonnollinen valinta tehdä sertifikaateille GitLabiin oma repositorio ja säilyttää ne siellä. Git-repositorion osoitteen syöttämisen jälkeen työkalu loi projektikansion juuressa olevaan fastlane-kansioon Matchfile-tiedoston (Liite 3). Koska tallennussijainniksi valittiin Git-repositorio, match pyysi määrittelemään sille salasanan. Matchin repositorioon tallentamat sertifikaatit salataan openssl:n avulla. Salasana lisättiin GitLabissa sovellus-projektin CI/CD-asetuksiin Variables-kohdan alle nimellä MATCH_PASSWORD. Sillä nimellä match osaa käyttää sitä automaattisesti.

Koska sovelluksesta julkaistiin kolmea eri versiota, kahteen testiympäristöön sekä tuotantoon, määriteltiin Matchfile-tiedostoon kaikkien kolmen sovelluksen tunnisteet app_identifier-kohtaan. Team_id arvoksi laitetaan kehitystiimin id, joka löytyi Apple Developer -portaalista, kohdasta Account ja Membership Details. Type-tiedolla määriteltiin minkä tyyppisiä sertifikaatteja match käsittelee oletuksena. Arvoksi asetettiin development, muita tyyppejä, kuten appstore, käsiteltäessä pitää sertifikaattien tyyppi antaa matchille erikseen.

Matchin määrittelyn jälkeen ajettiin fastlane match nuke -komento, joka poistaa olemassaolevat allekirjoitusavaimet. Sen jälkeen ajetut komennot fastlane match development sekä fastlane match Appstore luovat uudet allekirjoitusavaimet ja jakeluprofiilit ja ne tallennetaan määriteltyyn Git-repositorioon.

4.5 Fastfilen määrittely

4.5.1 Yleiset määrittelyt

Fastfilen (liite 4) alussa määriteltiin default_platform-arvoksi ios. Koska suuri osa Fastlanen toiminnoista tukee useampaa alustaa, default_platform-arvon määrittelyn jälkeen jokaiselle toiminnolle ei tarvinnut erikseen antaa platform-arvoa.

Seuraavaksi määriteltiin Fastlanen ios-alustalle muuttujia, kuten Git-repositorion osoite, joka sisältää sovelluksen allekirjoitusavaimet ja jakeluprofiilit sekä aiemmin mainittu JSON-tiedosto, joka sisältää App Store Connectin API:n autentikointiin vaaditut tiedot. Tiedot määriteltiin erikseen sen perusteella, ajetaanko Fastlanea lokaalissa kehittäjäympäristössä vai GitLabin CI/CD-putkessa. Sertifikaattien git-repositorioon määriteltiin pääsy vain tiimin jäsenille, joten sitä ei oletuksena voinut käyttää sovellus-repositorion CI/CD-putken sisältä. Sertifikaatti-repositorion asetuksista (Settings) navigoitiin CI/CD-kohdan alta löytyvään Job token permissions -asetuksiin. CI/CD job token allowlistille piti lisätä sovelluksen GitLab-projekti. GitLabin CI/CD-putki luo ajon alussa CI/CD job tokenin, jota käyttämällä ajettava CI-job autentikoituu sertifikaatti-repositorioon. Tokenin käyttö määriteltiin CI-koneella käytettävään sertifikaatti-repositorion osoitteeseen muodossa https://gitlab-ci-token:${CI_JOB_TOKEN}@sertifikaatti-repositorion-osoite.git. GitLabiin yhdistetyn macOS-laitteen Keychain, joka sisältää allekirjoitusavaimet, piti myös avata. Se onnistui Fastlanen unlock_keychain-toiminnolla. Kaikki CI/CD-putkelle määriteltyjen muuttujien arvot konfiguroitiin sovellus-repositorion asetuksissa CI/CD-osion Variables-kohtaan.

4.5.2 Julkaisu-lanen määrittely

Jokaiselle sovellusversiolle määriteltiin oma lane, jotka nimettiin julkaisuympäristön mukaan, esimerkiksi ios_dev_release-niminen lane suorittaa kehitysympäristöä vasten ajettavan sovelluksen kääntämisen, allekirjoittamisen sekä latauksen TestFlightiin.

Ensimmäisenä vaiheena ajettiin match-toiminto. Match haki määritellystä sertifikaatti-repositoriosta sovellustunnisteen sekä sertifikaattityypin perusteella oikeat allekirjoitusavaimet ja jakeluprofiilit. CI/CD-alustalla ajettaessa konfiguroitiin match ajettavaksi vain luku -tilassa, jolloin se ei tee muutoksia sertifikaatteihin.

Fastfilessa voi määritellä komentorivikäskyjä ajettavaksi. Se tehdään Fastlanen sh-toiminnolla. Ennen sovelluksen kääntämistä ladattiin kaikki sen käyttämät paketit komennolla nuget restore ../sovellus.sln. Sovelluksen kääntäminen suoritettiin MSBuildilla, jolle annettiin parametreiksi konfiguraatio, alusta ja target sekä

sovelluksen iOS-projekti. Komennolla käännettiin ja paketoitiin sovellus .app-tiedostoksi. Esimerkiksi kehitysympäristöön julkaistava sovellus kääntyi iOS-projektin bin/iPhone/DevRelease -hakemistoon.

Projektissa käytettiin kuvaajien piirtoon kirjastoa, joka oli alun perin kirjoitettu Swiftillä. Sille oli tehty Xamarin-sidokset, jotta sitä pystyi käyttämään Xamarin.iOS-projektissa. Koska Xamarin ei sillä hetkellä tarjonnut virallista tukea Swift-sidoksille, sovelluspaketista puuttui SwiftSupport-kansio sisältöineen. Ongelman korjaamiseksi käytettiin package_ipa_default.sh -skriptiä, joka paketoi msbuildin käyttämän, allekirjoitetun .app-tiedoston mukaan SwiftSupport-kansion ja paketoi sovelluksen .ipa-tiedostoksi.

Sovelluksen kääntämisen, allekirjoittamisen ja paketoimisen jälkeen ladataan sovelluspaketti TestFlightiin. Lataus suoritettiin Fastlanen pilot-toiminnolla. Parametreilla määriteltiin, että Pilot ei julkaise sovellusta TestFlightissa automaattisesti. Testaajat on TestFlightissa määritelty kahteen eri kategoriaan, sisäisiin ja ulkoisiin. Sisäiset testaajat ovat App Store Connectissa kehitystiimiin kuuluvat henkilöt. Ulkoiset testaajat ovat erikseen testiryhmään kutsuttuja ja lisättyjä käyttäjiä. Ulkoisia testaajia varten sovelluksen pitää julkaista TestFlightiin, jolloin se käy läpi App Storen hyväksyntätarkistukset. Pilotille määriteltiin myös skip_waiting_for_build_processing -arvo. Arvon ollessa true, pilot ei jää odottamaan, että sovelluksesta ilmestyy uusi versio näkyviin App Store Connectin builds-osioon. Tämä säästi aikaa ja nopeutti CI/CD-putken ajamista. Notify_external_testers -arvolla konfiguroitiin, ettei App Store Connectista lähtisi viestiä ulkoisille testaajille uudesta versiosta, koska sovellusversiota ei automaattisesti julkaistu ulkoiseen testiin.

Jokaiselle sovellusversiolle toteutettiin samanlainen lane. Eroina oli eri komennoille annettavat parametrit, esimerkiksi sovelluksen tunniste sekä msbuildille annettava konfiguraatio.

4.5.3 Lanet allekirjoitusavainten ja jakeluprofiilien hallintaan

Fastfileen määriteltiin kaksi lanea sovelluksen allekirjoitusavainten ja sovellusprofiilien hallinnan helpottamiseksi. Näiden kautta vanhentuneiden sertifikaattien uusiminen helpottui. Lisäksi esimerkiksi tiimiin tulevan uusi kehittäjä saisi helposti vaadittavat sertifikaatit asennettua tietokoneelleen.

Lanet määriteltiin nimillä nuke_ios_certs ja get_ios_certs. Näistä ensimmäinen kutsuu kahdesti match_nuke-toimintoa eri parametreilla. Sitä kutsutaan ensin development-tyypillä ja sen jälkeen appstore-tyypillä. Toiminto poistaa olemassaolevat sertifikaatit Matchfilessa (liite 3) määritellyiltä sovelluksilta. Get_ios_certs lane kutsuu match-toimintoa, jota kutsutaan samoilla parametreilla. Se luo uudet allekirjoitusavaimet sekä jakeluprofiilit sekä tallentaa ne aiemmin määriteltyyn git-repositorioon salattuna.

4.6 Integrointi GitLab CI/CD-putkeen

Fastlanen ajaminen GitLabin CI/CD-putkessa vaatii tiettyjen muuttujien asettamisen. Yhteenvetona, GitLabin CI/CD-muuttujiin lisättiin automaation kehityksen aikana

- MAC_BUILD_HOST_KEYCHAIN_PASSWORD
- APP_STORE_CONNECT_API_KEY_JSON
- MATCH_GIT_URL
- MATCH_PASSWORD.

YAML ankkureita käyttäen (Optimize GitLab CI/CD configuration files n.d.), gitlab-ci.yml-tiedostoon (Liite 5) määriteltiin uudelleenkäytettävä skripti before_fastlane. Skripti asentaa ensin Bundlerin, jonka jälkeen se suorittaa bundle install -komennon. Tämä komento asentaa Gemfile-tiedostossa määritellyt riippuvuudet, eli Fastlanen.

GitLabin CI/CD-putkelle oli määritelty kolme työvaihetta: build, test ja deploy. iOS-julkaisuille lisättiin yksi suoritettava työnkulku jokaiselle julkaisuympäristölle. Työnkulut määritelttiin suoritettavaksi deploy-vaiheessa. Niille annettu mac-tagi

määritteli millä GitLab Runnerilla työ ajetaan. LC_ALL ja LANG -muuttujat asetettiin arvoon en_US.UTF-8. Fastlane vaatii, että työtilalle on määritetty käyttämään UTF-8 (Fastlane 2025).

Projektissa oli käytäntönä, että julkaisusta tehtävälle commitille annetaan tagiksi alusta/versionumero, esimerkiksi ios/1.0.1. Molempien alustojen julkaisutyönkulut määriteltiin GitLabiin saataville vain, kun commitissa oli kyseiseisen työnkulun alustaa vastaava tagi.

iOS-julkaisutyönkulkujen before_script-kohdassa määriteltiin ajettavaksi aiemmin määritelty skripti. Skriptin tarkoitus oli varmistaa, että Fastlane olisi aina asennettuna GitLab Runnerin työtilassa. Script-osiossa määriteltiin suoritettava skripti, joka iOS-julkaisuja tehdessä oli kutsua Fastlanen Fastfilessa (liite 4) määriteltyjä laneja.

Artifacts-osiossa määriteltiin mitä tiedostoja halutaan GitLabin jättävän saataville työnkulkujen valmistuttua. Paths-osioon konfiguroitiin minkä tiedoston GitLab otti talteen ja name-osiossa millä nimellä artefakti tallentuu. Tallennettuja artefakteja voi selata ja ladata GitLabissa valitsemalla projektin valikosta Build ja Artifacts.

4.7 Manuaaliseksi jätetyt työvaiheet

Sovelluksen repositoriossa pidettiin listaa eri julkaisuihin tehdyistä muutoksista. Näiden päivitys jäi edelleen käsin tehtäväksi. Julkaisumuistiinpanoja tehtäessä päivitettiin myös käsin sovelluksen versio- ja build-numerot Info.plist-tiedostoihin. Nämä muutokset puskettiin GitLabiin ja commitille merkittiin tagiksi alusta ja versionumero, esimerkiksi iOS/1.0.1, jolloin GitLabin CI/CD-putki mahdollisti julkaisutyövaiheiden suorittamisen. Olemassa olevan Android-julkaisuputken lailla, iOS-julkaisuputken käynnistys jätettiin myös manuaalisesti tehtäväksi.

Käsin tehtäväksi jätettiin myös sovelluksen julkaisu App Storeen. Julkaisu tehdään App Store Connectissa, jossa navigoidaan Apps-välilehdelle ja valitaan julkaistava sovellus. Sivupalkista valitaan Prepare for Submission. Build-kohdassa valitaan mikä sovellusversio halutaan julkaista. Sovelluspaketin voi ladata tässä

kohtaa itse tai valita jo automaation TestFlightiin lataamista paketeista. Valitaan millä tavalla julkaisu tapahtuu Applen tarkistuksen jälkeen, automaattisesti, tiettynä määritettynä ajankohta tai manuaalisesti. Tämän jälkeen painetaan Add to Review, joka toimittaa sovelluksen Applen tarkistusjonoon. Tarkastuksen läpäisemisen jälkeen sovellus julkaistaan aiemmin tehdyn valinnan mukaisesti.

5 POHDINTA

Julkaisuautomaation tutkiminen ja toteuttaminen oli mielenkiintoista ja opin paljon uutta sekä CI/CD-prosesseista ja työkaluista. Oli myös hienoa helpottaa kehitystiimin toimintaa ja vapauttaa työaikaa varsinaiseen kehitystyöhön. Allekirjoitusavainten ja jakeluprofiilien hallinnan yksinkertaistaminen myös vähensi päänvaivaa. Julkaisuautomaatioputkelle saatiin toteutettua kaikki työn alussa määritellyt työvaiheet ja tavoitteet.

Sovelluksen julkaisuun jäi vielä manuaalisia työvaiheita, esimerkiksi julkaisuputken käynnistys ja sovelluksen lopullinen julkaisu App Storeen. Jatkokehityksenä voisi tarkastella näiden manuaalisten vaiheiden poistoa ja toteuttaa täysin automaattisen sovellusjulkaisun. Myös sovelluksen Android-version julkaisuputken siirtämistä Fastlanella suoritettavaksi voisi tarkastella.

Työn käytännön osuus tehtiin, kun Xamarin oli vielä tuettu. Luonnollinen jatkokehitysidea olisi muokata julkaisuautomaatiota toimimaan sen seuraajalla, .NET MAUI:lla. Merkittävästi tämä ei eroaisi työssä kuvatusta toteutuksesta. Erot tulisivat sovelluksen kääntämisen yhteydessä. Xamarin vaati Visual Studion ja Xamarin SDK:n asennuksen, kun taas .NET MAUI vaatii .NET SDK:n sekä MAUI workloadin asentamisen. Xamarin sovellukset käännettiin msbuild.exe:llä, kun taas MAUI käyttää MSBuildia dotnet build -komennon kautta.

Lopputuloksena saatiin toimiva julkaisuautomaatio Xamarin.iOS-sovellukselle, joka säästi useita tunteja aikaa jokaista julkaisua kohden. Julkaisuprosessi sekä erityisesti allekirjoitusavainten ja jakeluprofiilien hallinta selkeytyi huomattavasti. CI/CD-automaation toteutuksella saavutetut edut ovat juurikin niitä etuja, joita työn teoriaosuudessa tarkasteltiin. Yhteenvetona voidaan todeta, että automatisoitu CI/CD-työnkulku on ehdottoman tärkeä osa modernia mobiilisovelluskehitystä.

LÄHTEET

About GitLab. 2025. GitLab. Verkkosivu. Viitattu 29.05.2025. https://about.gitlab.com/company/

About Mono. n.d. Mono Project. Verkkosivu. Viitattu 27.5.2025. https://www.mono-project.com/docs/about-mono/

Ahmad, A., Li, K., Feng, C., Asim S. M., Yousif, A. Ge, S. 2019. An Empirical Study of Investigating Mobile Applications Development Challenges. Viitattu 02.06.2025. DOI: 10.1109/ACCESS.2018.2818724

Asfour, A., Zain, S., Salleh, N., & Grundy, J. (2019). Exploring agile mobile app development in industrial contexts: A qualitative study. Viitattu 03.06.2025. https://www.researchgate.net/publication/329569528_Exploring_Agile_Mobile_App_Development_in_Industrial_Contexts_A_Qualitative_Study

Bernardo, J. H., Da Costa, D. A., Kulesza, U. Treude, C. 2023. The Impact of a Continuous Integration Service on the Delivery Time of Merged Pull Requests. Viitattu 03.06.2025. DOI: 10.48550/arXiv.2305.16365

Bhimanapati, V. B. R., Goel, O. 2023. Implementing CI/CD for Mobile Application Development in Highly Regulated Industries. Viitattu 03.06.2025. DOI: 10.1234/jis.2020.0301

Build Properties for iOS, Mac Catalyst, macOS and tvOS. 2025. Microsoft. Verkkosivu. Viitattu 28.5.2025. https://learn.microsoft.com/en-us/dotnet/ios/building-apps/build-properties

Cruz, L., Abreu, R., Lo, D. 2019. To the Attention of Mobile Software Developers: Guess What, Test your App! Viitattu 03.06.2025. DOI: 10.48550/arXiv.1902.02610

Encyclopædia Britannica. n.d. macOS. Verkkosivu. Viitattu 26.5.2025. https://www.britannica.com/technology/macOS

Fastlane. 2025. Continous Integration. Verkkosivu. Viitattu 01.06.2025. https://docs.fastlane.tools/best-practices/continuous-integration/#environment-variables-to-set

Fastlane docs. 2025. Fastlane. Verkkosivu. Viitattu 29.05.2025. https://docs.fastlane.tools/

Fluri, J., Fornari, F., Pustulka, E. 2023. Measuring the Benefits of CI/CD Practices for Database Application Development. DOI: 10.1109/ICSSP59042.2023.00015

Ghaleb, T., Abduljalil, O., Hassan, S. 2025. CI/CD Configuration Practices in Open-Source Android Apps: An Empirical Study. Viitattu 03.06.2025. DOI: 10.48550/arXiv.2411.06077

GitLab Features. n.d. GitLab. Verkkosivu. Viitattu 29.5.2025. https://about.gitlab.com/features/

GitLab Pricing. n.d. GitLab. Verkkosivu. Viitattu 29.05.2025. https://about.gitlab.com/pricing/

Guthrie, S. 2016. Microsoft to acquire Xamarin and empower more developers to build apps on any device. Official Microsoft Blog. Verkkosivu. Viitattu 27.5.2025. https://blogs.microsoft.com/blog/2016/02/24/microsoft-to-acquire-xamarin-and-empower-more-developers-to-build-apps-on-any-device/

Howarth, J. 2025. iPhone vs Android User Stats (2025 Data). Exploding Topics. Verkkosivu. Viitattu 04.06.2025. https://explodingtopics.com/blog/iphone-android-users

Johnson, J., Britch, D., Buck, A., Arya, H., Dunn, C. 2020. What is Xamarin? Microsoft. Verkkosivu. Viitattu 27.5.2025. https://learn.microsoft.com/en-us/previous-versions/xamarin/get-started/what-is-xamarin

Kirubakaran, B., Karthikeyani, V. 2013. Mobile Application Testing – Challenges and Solution Approach through Automation. Viitattu 03.06.2025. DOI: 10.1109/icprime.2013.6496451

Laukkanen, E., Itkonen, J., Lassenius, C. 2017. Problems, Causes and Solutions When Adopting Continuous Delivery - A Systematic Literature Review. Viitattu 02.06.2025. DOI: 10.1016/j.infsof.2016.10.001

McIlroy, S., Ali, N., Hassan, A. E. 2015. Fresh apps: an empirical study of frequently-updated mobile apps in the Google play store. Viitattu 03.06.2025. DOI: 10.1007/s10664-015-9388-2

Membership Details. n.d. Apple. Verkkosivu. Viitattu 29.05.2025. https://developer.apple.com/programs/whats-included/

MSBuild. 2024. Microsoft. Verkkosivu. Viitattu 28.5.2025. https://learn.microsoft.com/en-us/visualstudio/msbuild/msbuild

MSBuild command-line reference. 2024. Microsoft. Verkkosivu. Viitattu 28.5.2025. https://learn.microsoft.com/en-us/visualstudio/msbuild/msbuild-command-line-reference

Optimize GitLab CI/CD configuration files. n.d. GitLab Docs. Verkkosivu. Viitattu 29.05.2025. https://docs.gitlab.com/ci/yaml/yaml_optimization/#anchors

Ortinau, D., Britch, D., Sherer, T., Dunn, C., Schonning, N. 2017a. Compiling for different devices in Xamarin.iOS. Microsoft. Verkkosivu. Viitattu 28.5.2025. https://learn.microsoft.com/en-us/previous-versions/xamarin/ios/deploy-test/compiling-for-different-devices

Ortinau, D., McClister, C., Britch, D., Coulter, D., Dunn, C., Schonning, N., Pakala, Y. 2017b. iOS App Architecture. Microsoft. Verkkosivu. Viitattu 27.5.2025. https://learn.microsoft.com/en-us/previous-versions/xamarin/ios/internals/architecture

Ortinau, D. Britch, D. Sherer, T. Dunn, C. Schonning N. 2017c. iOS Build Mechanics. Microsoft. Verkkosivu. Viitattu 28.5.2025. https://learn.microsoft.com/en-us/previous-versions/xamarin/ios/deploy-test/ios-build-mechanics

Rahman, A., Agrawal, A., Krishna, R., Sobran, A. 2018. Characterizing The Influence of Continuous Integration: Empirical Results from 250+ Open Source and Proprietary Projects. DOI: 10.48550/arXiv.1711.03933

Red Hat. 2023. What is CI/CD? Verkkosivu. Viitattu 01.06.2025. https://www.redhat.com/en/topics/devops/what-is-ci-cd

Shahin, M., Ali Babar, M., Zhu, L. 2017. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. Viitattu 02.06.2025. DOI: 10.1109/ACCESS.2017.2685629

Strakh, A. 2020a. Binding Android Kotlin Libraries. Microsoft. Verkkosivu. Viitattu 28.5.2025. https://devblogs.microsoft.com/xamarin/binding-android-kotlin-libraries/

Strakh, A. 2020b. Binding iOS Swift Libraries. Microsoft. Verkkosivu. Viitattu 28.5.2025. https://devblogs.microsoft.com/xamarin/binding-ios-swift-libraries

Supported Platforms. n.d. Mono Project. Verkkosivu. Viitattu 27.5.2025. https://www.mono-project.com/docs/about-mono/supported-platforms/

Teixeira, L. 2019. ipa-packager. GitHub. Verkkosivu. Viitattu 31.05.2025. https://github.com/Flash3001/ipa-packager

Vidjikant, S. 2024. Xamarin App Development: Advantages and Disadvantages. Softjourn. Verkkosivu. Viitattu 28.5.2025. https://softjourn.com/insights/xamarin-app-development-advantages-and-disadvantages

Volle, A. 2025. iOS. Encyclopædia Britannica. Verkkosivu. Viitattu 05.06.2025. https://www.britannica.com/technology/iOS

Xamarin Support Policy. 2024. Microsoft. Verkkosivu. Viitattu 27.5.2025. Verkkosivu. https://dotnet.microsoft.com/en-us/platform/support/policy/xamarin

Xcode - Support - Apple Developer n.d. Apple. Verkkosivu. Viitattu 26.5.2025. https://developer.apple.com/support/xcode/

LIITTEET

Liite 1. Gemfile

```
source "https://rubygems.org"

gem "fastlane"
```

Liite 2. JSON-tiedosto Fastlanen App Store Connect -autentikointiin

```
{
    "key_id": "$KEY_ID",
    "issuer id": "$ISSUER_ID" ,
    "key": "$KEY-TIEDOSTON SISÄLTÖ",
    "in_house": false
}
```

Liite 3. Matchfile

```
storage_mode ("git")
type ("development") # The default type, can be: appstore, adhoc, enterprise
or development
team_id ("ABCD1234")
app_identifier (["fi.sovellus.ios.dev", "fi.sovellus.ios.staging", "fi.
sovellus.ios"])
```

Liite 4. Fastfile


```
# This file contains the fastlane.tools configuration
# You can find the documentation at https://docs.fastlane.tools
#
# For a list of all available actions, check out
#
#     https://docs.fastlane.tools/actions
#
# For a list of all available plugins, check out
#
#     https://docs.fastlane.tools/plugins/available-plugins
#

# Uncomment the line if you want fastlane to automatically update itself
# update_fastlane

default_platform(:ios)

platform :ios do

  if is_ci
    api_key_path =  ENV['APP_STORE_CONNECT_API_KEY_JSON']
    match_git_url = ENV['MATCH_GIT_URL']
    unlock_keychain(
      password: ENV['MAC_BUILD_HOST_KEYCHAIN_PASSWORD']
    )
  else
    api_key_path = "./fastlane/app_store_connect_api.json"
    match_git_url = "git@gitlabin.osoite:sovellus/sovellus-certificates.git"
  end

  lane :ios_dev_release do

    match(
      app_identifier: "fi.sovellus.ios.dev",
      api_key_path: api_key_path,
      git_url: match_git_url,
      type: "appstore",
      readonly: is_ci
    )

    sh 'nuget restore ../Sovellus/Sovellus.sln'

    sh 'msbuild /p:Configuration=DevRelease /p:Platform=iPhone /t:Build
../Sovellus/Sovellus.Client.iOS/Sovellus.Client.iOS.csproj'

    sh 'sh package_ipa_default.sh
../Sovellus/Sovellus.Client.iOS/bin/iPhone/DevRelease/Sovellus.Client.iOS.app
$PWD/../sovellus-dev.ipa'

    pilot(
```

```
      app_identifier: "fi.sovellus.ios.dev",
      api_key_path: api_key_path,
      skip_submission: true,        #Only need to submit for external testers,
internal testers will get update
      ipa: "sovellus-dev.ipa",
      notify_external_testers: false,
      skip_waiting_for_build_processing: true
    )

  end

  lane :ios_staging_release do

    match(
      app_identifier: "fi.sovellus.ios.staging",
      api_key_path: api_key_path,
      git_url: match_git_url,
      type: "appstore",
      readonly: is_ci
    )

    sh 'nuget restore ../Sovellus/Sovellus.sln'

    sh 'msbuild /p:Configuration=StagingRelease /p:Platform=iPhone /t:Build
../Sovellus/Sovellus.Client.iOS/Sovellus.Client.iOS.csproj'

    sh 'sh package_ipa_default.sh
../Sovellus/Sovellus.Client.iOS/bin/iPhone/StagingRelease/Sovellus.Client.iOS.
app $PWD/../sovellus-staging.ipa'

    pilot(
      app_identifier: "fi.sovellus.ios.staging",
      api_key_path: api_key_path,
      skip_submission: true,
      ipa: "sovellus-staging.ipa",
      notify_external_testers: false,
      skip_waiting_for_build_processing: true
    )

  end

  lane :ios_prod_release do

    match(
      app_identifier: "fi.sovellus.ios",
      api_key_path: api_key_path,
      git_url: match_git_url,
      type: "appstore",
      readonly: is_ci
    )

    sh 'nuget restore ../Sovellus/Sovellus.sln'
```

```
    sh 'msbuild /p:Configuration=ProdRelease /p:Platform=iPhone /t:Build
../Sovellus/Sovellus.Client.iOS/Sovellus.Client.iOS.csproj'

    sh 'sh package_ipa_default.sh
../Sovellus/Sovellus.Client.iOS/bin/iPhone/ProdRelease/Sovellus.Client.iOS.app
$PWD/../sovellus-prod.ipa'

    pilot(
      app_identifier: "fi.sovellus.ios",
      api_key_path: api_key_path,
      skip_submission: true,
      ipa: "sovellus-prod.ipa",
      notify_external_testers: false,
      skip_waiting_for_build_processing: true
    )

  end

  lane :nuke_ios_certs do

    match_nuke(
      api_key_path: api_key_path,
      git_url: match_git_url,
      type: "development"
    )

    match_nuke(
      api_key_path: api_key_path,
      git_url: match_git_url,
      type: "appstore"
    )

  end

  lane :get_ios_certs do

    match(
      api_key_path: api_key_path,
      git_url: match_git_url,
      type: "development"
    )

    match(
      api_key_path: api_key_path,
      git_url: match_git_url,
      type: "appstore"
    )

  end

end
```

Liite 5. gitlab-ci.yml, iOS-sovellusta koskevat osat


```
stages:
  - build
  - test
  - deploy

default:
  cache:
    paths:
      - Sovellus/packages/
  before_script:
    - 'nuget restore Sovellus/Sovellus.sln'

.before_fastlane: &before_fastlane
  - 'gem install bundler'
  - 'bundle install'

build_ios:
  stage: build
  tags:
    - mac
  script:
    - 'msbuild Sovellus/Sovellus.sln -p:Configuration=DevRelease -
p:Platform=iPhoneSimulator -p:CrashlyticsServiceInfo=""'
  artifacts:
    paths:
      - 'Sovellus/*/bin'
    when: on_success
    expire_in: 30 mins
  interruptible: true

test_frontend:
  stage: test
  tags:
    - windows
  script:
    - 'nunit-console
Sovellus/Sovellus.Client.Common.Test/Sovellus.Client.Common.Test.csproj
/config:Release'
  interruptible: true

ios_dev_release:
  stage: deploy
  tags:
    - mac
  dependencies: []
  variables:
```

```
    LC_ALL: "en_US.UTF-8"
    LANG: "en_US.UTF-8"
  rules:
    - if: '$CI_COMMIT_TAG =~ /^(ios)\//'
  before_script:
    - *before_fastlane
  script:
    - bundle exec fastlane ios_dev_release
  artifacts:
    paths:
      - sovellus-dev.ipa
    name: "ios-dev-release-$CI_COMMIT_TAG"

ios_staging_release:
  stage: deploy
  tags:
    - mac
  dependencies: []
  variables:
    LC_ALL: "en_US.UTF-8"
    LANG: "en_US.UTF-8"
  rules:
    - if: '$CI_COMMIT_TAG =~ /^(ios)\//'
      when: manual
  before_script:
    - *before_fastlane
  script:
    - bundle exec fastlane ios_staging_release
  artifacts:
    paths:
      - sovellus-staging.ipa
    name: "ios-staging-release-$CI_COMMIT_TAG"

ios_prod_release:
  stage: deploy
  tags:
    - mac
  dependencies: []
  variables:
    LC_ALL: "en_US.UTF-8"
    LANG: "en_US.UTF-8"
  rules:
    - if: '$CI_COMMIT_TAG =~ /^(ios)\//'
      when: manual
  before_script:
    - *before_fastlane
  script:
    - bundle exec fastlane ios_prod_release
  artifacts:
    paths:
      - sovellus-prod.ipa
    name: "ios-prod-release-$CI_COMMIT_TAG
```