TARTU ÜLIKOOL
LOODUS- JA TÄPPISTEADUSTE VALDKOND
Arvutiteaduse instituut
Informaatika õppekava

Juri Noga

iOS mobiilirakenduste arendamise kursuse õppematerjal Swift keele baasil

Bakalaureusetöö (9 EAP)

Juhendajad: Tauno Palts, MA
Marina Lepp, PhD

Tartu 2016

iOS mobiilirakenduste arendamise kursuse õppematerjal Swift keele baasil

Lühikokkuvõte:

Selles lõputöös antakse ülevaade iOS mobiilirakenduste arendamise kursuse jaoks koostatud õppematerjalidest. Kirjeldatud on aine ülesehitus, põhjendatud konkreetsete teemade valimist ja materjalide õpetamise stiili. Lisaks vaadeldakse mobiilirakenduste põhiprintsiipe ja nende rakendamist reaalsete näidete abil.

Võtmesõnad:

Mobiilirakendused, Swift, haridus, kasutajaliideste kavandamine

CERCS: P175

iOS application development with Swift learning materials

Abstract:

This thesis provides an overview of learning material for iOS mobile application development course. It consists of the description of the course structure, decisions behind specific topics choice and materials teaching approach. The material also includes key principles of mobile application development and application of these in real-world examples.

Keywords:

Mobile applications, Swift, education, user interface design

CERCS: P175

Sisukord

1. Sissejuhatus

Mobiilirakenduste arenduse õppimiseks on mitmeid võimalusi. Tartu Ülikooli arvutiteaduste instituut pakub kursust "Mobiilirakenduste loomine" (MTAT.03.262), mis aga ei keskendu iOS (Apple-i arendatav mobiilsete seadmete operatsioonisüsteem) mobiilirakenduste arendamisele. Kuna nutitelefonide kasutuselevõtt on tõusnud meie elus väga kiiresti – aastal 2009 oli müüdud 172 miljonit seadet, aga aastal 2015 juba 1423 miljonit seadet[1], siis arendades mobiilirakendusi võib oma töö tulemusega jõuda miljonite potentsiaalsete kasutajani.

Mobiilirakenduste arendamisel on tähtis õppida mitte ainult programmeerimiskeeli või tarkvaraarenduspakette, vaid tihti arendaja tegeleb ka kasutajaliideste kavandamisega, ikoonide joonistamisega/valmistamisega ja serveriga suhtlemise seadistamisega. See tähendab, et lisaks nendele kõikidele tehnoloogiatele, peab arendaja protsessi üldiselt ja iga selle osa eraldi hästi mõistma.

Kiirelt kasvava konkurentsiga turul olemine lisab aspekti, et on väga raske jõuda kaugele triviaalse lahendusega, mis ei paku uut lisaväärtust võrreldes olemasolevate rakendustega. Lisaks sellele piisab omandatud teadmistest väga lühikeseks perioodiks, millest järeldub, et peab alati uusi tehnoloogiaid õppima, trende jälgima ja võib-olla ka neid ennustama.

Edukaks teadmiste kättesaamiseks on laialt levinud õppimine loomise ajal (ingl *learn-by-doing*), mis sisuliselt tähendab seda, et parimad teadmised saab lahendades konkreetseid ülesandeid, mitte uurides ainult videoid ja lugedes materjale. Carnegie Mellon's West Coast Campus on läbiviinud tarkvaratehnika kursuse, mis oli rühmatöö ja projektipõhiline ning tulemused olid positiivsed – 92% tudengitest arvasid, et selline lähenemine andis neile konkurentsieelist võrreldes teiste eakaaslastega [1].

Stanfordi kursuses "Developing iOS 8 Apps with Swift"[2] alustatakse õpetamist tavaliselt lühisissejuhatusega, kus selgitatakse põhiprintsiipe: tutvustatakse arendamiskeskonda, õpetatakse keelt ja antakse baasteadmisi rakenduste ülesehituse kohta. Pärast seda tuleb kasutades saadud teadmisi lahendada konkreetne probleem ja saada töötav rakendus.

[1] http://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/

Valmis rakendus tähendab lisaks funktsionaalsetele võimalustele ka kasutajaliideste läbimõtlemist ja visuaalsete elementide valmistamist.

Selles bakalaureusetöös on järgnevad osad:

1. iOS-i ja Swifti ajalugu ning ülevaade õppematerjalides kasutatavatest tehnoloogiatest. Välja on toodud iOS-i ja Swifti lühiajalugu – märkimisväärsed kuupäevad ja vastavad muudatused. Lisaks tutvustatakse põhitehnoloogiaid, mida on vaadeldud õppematerjalides.
2. Metoodika osa, kus on kirjeldatud töö loomise protsessi ja kirjeldatud teatud tehnoloogiate valiku põhjused.
3. Tulemuste osa, mis sisaldab loodud kursuse ja õppematerjalide kirjeldust.
4. Kokkuvõte, kus on toodud töö ülevaade ja kokkuvõte.
5. Lisad, mis sisaldavad programmeerimiskeelte Objective-C ja Swift jõudluse ja süntaksi võrreldust ning kursuse põhjalik kava.

2. iOS-i ja Swifti ajalugu ja õppematerjalides kasutatavad tehnoloogiad

2.1 iOS-i ja Swifti ajalugu

iOS (kuni 2010. aastani iPhone OS) tuli avalikkuse ette koos esimese iPhone-iga aastal 2007. Sellel ajal oli operatsioonisüsteem funktsionaalselt piiratud, näiteks puudusid kopeerimise võimalus (ingl *copy/paste*) ja video salvestamise võimalus. Lisaks puudus ka võimalus paigaldada ja arendada kolmanda osapoole rakendusi – arendajad ja tavakasutajad võisid kasutada ainult piiratud funktsionaalsusega veebirakendusi[2].

Koos iPhone OS 2 ja App Store-i avamisega tehti saadavaks sisseehitatud arenduskeskkond Xcode – arenduskomplekti (SDK), mille abil sai arendada (peamiselt programmeerimiskeeles Objective-C) ja avaldada rakendusi App Store-is.

iPhone OS 3 tõi arendajatele võimaluse saata *push*-teateid ja teha videosalvestusi.

2010. aastal nimetati iPhone OS ümber iOS-iks tänu sellele, et lisaks iPhone-ile töötas operatsioonisüsteem nüüd ka iPad-i peal. iOS 4 põhivõimalusteks olid[2]:

1. Multitegumtöö (ingl *multitasking*) – rakendused ei laadita mälust välja teiste rakenduste avamisel. See võimaldas käivitada rakendusi taustal (asukoht, diktofon, jne) ning ei pidanud taaskäivitama äsja laaditud rakendusi.
2. iPad tugi – tekkis võimalus arendada rakendusi iPad-i jaoks või optimeerida olemasolevaid suurema ekraani jaoks.
3. Kõrge resolutsiooniga ekraanide tugi – koos iPhone 4 ja Retina-ekraanide tulekuga muutus ekraanile pildi kuvamise põhimõte: kui varem üks punkt pildil vastas ühele pikslile ekraanil, siis nüüd, näiteks Retina-ekraani puhul ühe punkti laius vastab kahe piksli laiusele. Seega tuleb programmeerimisel arvestada punktide, mitte pikslitega ja probleeme võivad tekitada erinevate rasterpiltide kuvamised.

iOS 5 ja iOS 6, mis tulid saadavale vastavalt 2011. ja 2012. aastal, tõid endaga kaasa lisaks kasutajale keskendatud võimalustele ka parandusi, mis lihtsustasid rakenduste arendamist. Olulised muutused olid tehtud iOS 7-s, kus tehti ümber terve süsteemi

[2] http://www.theverge.com/2011/12/13/2612736/ios-history-iphone-ipad

välimus, samuti parandati mõned süsteemi mehhanismid nagu multitegumtöö ja traadita võrkudega töö.

iOS 8 tulek oluliselt laiendas arendajate võimalusi. Esitleti kolmanda osapoole klaviatuure, teiste rakenduste laiendusi ja vidinad teavituste keskuses nimega Notification Center.

iOS 9 ei toonud suuri muutusi arendajatele, selles versioonis keskenduti üldiselt operatsioonisüsteemi ja API parandustele.

WWDC (Worldwide Developers Conference)[3] 2014 ajal, kus esitleti iOS 8, avalikustati ka uus programmeerimiskeel Swift.

Peamised Swifti arendamise eesmärgid ja sellele tulevikus ülemineku Objective-C-st põhjused on [3]:

1. Kaasaegsus – Swift toetab kaasaegseid programmeerimise paradigmasid nagu funktsionaalne programmeerimine ja protokoll-orienteeritud programmeerimine.
2. Jõudlus – tänu oluliselt suuremale kompileerimisaja optimeerimisele (ingl *compile-time optimizations*) ja dünaamilisuse vähendamisele, on keele jõudlus parem, kui Objective-C-l.
3. Koodi kirjutamise lihtsus – keele arendajad vabanesid päise-failidest (ingl *header*), keele süntaks muutus selgemaks (näiteks kadusid semikoolonid ridade lõppudest ja käsuvoogude käskude ümbritsevad sulud).

Viimasest punktist võib järeldada, et Swifti õppimine peaks toimuma kiiremini ja lihtsamini, mis omakorda viib kiiremale adaptatsioonile kogenud ja algavate arendajate ringis.

Swift on kompileeritud LLVM (Low Level Virtual Machine) kompilaatoriga bait-koodi, mis on identne Objective-C' ja C' bait-koodiga. See võimaldab vabalt kasutada nendes keeltes kirjutatud faile projektides ilma ümbris-failideta (ingl *wrapper*) ja kolmanda osapoole kompilaatoriteta.

WWDC 2015 üritusel esitleti[4] uut versiooni Swift 2, mille lähtekood avati 2015. aasta lõpus. 3. detsembril 2015[5] Apple avalikustas kompilaatori ja standardteegi lähtekoodi

[3] http://www.apple.com/apple-events/june-2014/

GitHubis. Lisaks porteeriti Swift ka Linux platvormile. Jõudlus, kaasaegsus ja avalikkus olid põhjusteks, miks just Swift oli valitud selle bakalaureuse töö raames loodava kursuse programmeerimiskeeleks.

2.2 Õppematerjalides kasutatavate tehnoloogiate kirjeldus

Swift

Swift on programmeerimiskeel, mis on loodud suuremas osas iOS, OS X ja watchOS rakenduste arendamiseks [3, 4]. Keele arendus algas 2010. aastal, kui projekti juhendajaks ja vanemarendajaks sai Chris Lattner (LLVM-kompilaatori autor)[6]. Swift avalikustati ainult neli aastat hiljem 2. juunil 2014 arendajate konverentsil WWDC 2014. Swifti mõjutasid mitmed teised programmeerimiskeeled. Näiteks Objective-C-lt sai Swift alusteegi (ingl *foundation framework*), Haskellilt sai funktsionaalprogrameerimise võimalused. Mõjutusi on Swift saanud ka keeltest nagu Rust, Ruby, Python, C#.

Swift kasutab LLVM kompilaatori infrastruktuuri ja Objective-C mootorit (ingl *runtime*). Sellest järeldub, et neid kahte keelt saab kasutada ühes samas projektis, kuna mõlemad kompileeritakse kohalikku (ingl *native*) masinkoodi. Swiftil on liigipääs Objective-C klassidele, tüüpidele, funktsioonidele ja muutujatele läbi sillapäisele (ingl *bridging header)* ja kasutades laiendusi saab ligi C ja C++ koodile (mis peab olema kompileeritav LLVM kompilaatoriga). See võimaldab Swiftil kasutada Cocoa, Cocoa Touch ja teisi olemasolevaid Objective-C teeke vabalt oma projektis [5].

Teek Foundation

Selle töö raames loodud õppematerjalis on kesksel kohal põhitehnoloogia, mida kasutatakse iOS rakenduste arendamisel. Tegemist on sisuliselt alusteegiga (ingl *Foundation framework*), mille lisaeeliseks on see, et see teek on sama kõikidel Apple-i platvormidel, ehk saadud teadmisi saab kasutada ka OS X rakenduste arendamisel.

---

[4] http://www.apple.com/live/2015-june-event/

[5] https://swift.org/blog/welcome/

[6] http://www.nondot.org/sabre/

Foundation teek kujutab endast funktsionaalsust, mida läheb vaja pea igas rakenduses.

Hetkel on Foundation ametlikult olemas ainult Objective-C keele jaoks, kuid Swifti jaoks on loodud niinimetatud pakendeid (ingl *wrapper*), mis lubavad seda siiski kasutada. Swifti oma Foundaion teegi arendust on võimalik jälgida GitHubis[7].

Foundation kehtestab lisaks paljudele primitiivsetele klassidele ka mitut paradigmat ja funktsionaalsust, mis standardteegi (ingl *standard library*) kaudu ei avaldu.

Foundationi peamised eesmärgid on [4]:

- Pakkuda suurt hulka põhilisi utiliite.
- Kehtestada järjekindlaid konventsioone ja teha seeläbi tarkvaraarendust lihtsamaks.
- Teha tarkvara kättesaadavaks üle kogu maailma pakkudes erinevate keelte tuge.
- Olla sõltumatu operatsioonisüsteemist ehk võimaldada portimisvõimekust.

Loodud õppematerjalis on toodud ülevaade klassidest, mis täidavad ülaltoodud eesmärke ja jaotuvad kaheksaks alamgrupiks:

1. Andmete hoidmine.
2. Tekst ja sõned.
3. Kuupäev ja aeg.
4. Sorteerimine ja filtreerimine.
5. Rakenduse koordineerimine ja juhtimine.
6. Objektide jaotus ja püsivus.
7. OS teenused.
8. Andmete laadimissüsteem.

UIKit – teek rakenduste loomiseks

Teiseks oluliseks osaks loodud õppematerjalides on UIKit. UIKit on teek, mis pakub kriitilise infrastruktuuri vajaliku iOS rakenduste arendamiseks [6]. Üldised teegi ülesanded on toetada:

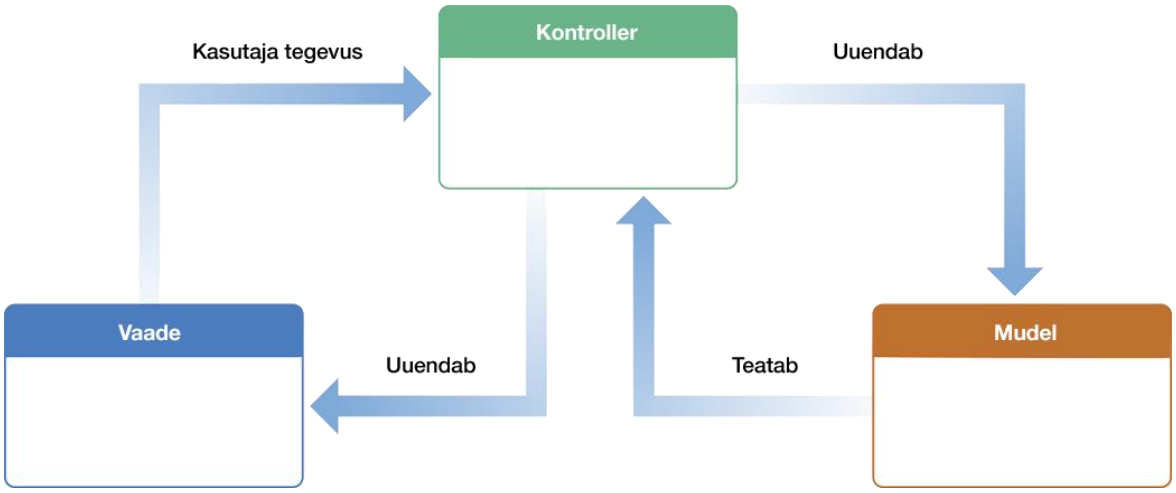
- Akna- ja vaatesüsteeme selleks, et kontrollida rakenduse kasutajaliidest.

[7] https://github.com/apple/swift-corelibs-foundation

- Sündmuste käsitlemisinfrastruktuuri, mis tegeleb kasutaja sisendi töötlemisega ja reageerimisega.
- Rakenduse mudelit, mis tegeleb andmetega ja operatsioonisüsteemiga suhtlemisega.

iOS rakenduste arendamisel on soovituslik järgida MVC (*Model-View-Controller*) mustrit. MVC jagab rakenduse komponentide (klasside) töö kolmeks:

- Mudelklassid (ingl *model*), mis hoiavad andmeid.
- Vaateklassid (ingl *view*), mis loovad ja kuvavad kasutajaliidest.
- Kontrollerklassid (ingl *controller*), mis reageerivad erinevatele sündmustele (näiteks nupuvajutusele või andmete muutmisele) ja panevad tööle vastavad funktsioonid (andmebaasi kirje või vaate uuendamine).



Joonis 1. MVC arhitektuur[8]

Kõikidel UIKiti kuuluvatel klassidel on nime ees "*UI*", et eristada neid teistest klassidest ja vähendada trükivigu.

Adaptiivse kujunduse vahend Auto Layout

Kolmandaks oluliseks osaks loodud õppematerjalides on vahend Auto Layout. Apple Auto Layout giid [7] defineerib Auto Layouti nagu süsteemi, mis dünaamiliselt arvutab kõikide

[8] https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

hierarhias olevate vaadete suurusi ja positsioone kasutades piiranguid (ingl *constraints*), mis on lisatud nendele vaadetele. Näiteks, saab lisada piiranguid (ingl *add constraints)* nupule nii, et selle ülemine serv oleks alati kaheksa punkti võrra all pildi vaate alumisest servast. See tähendab, et kui pildi vaate positsioon muutub, automaatselt muutub ka vastavalt nuppu positsioon. Selline lähenemine võimaldab luua kasutajaliidest, mis dünaamiliselt vastab sisemistele ja välimistele muudatustele.

Näited sisemisest muudatustest [7]:

- Rakenduse sisu muutub.
- Rakendus toetab mitu keelt.
- Rakendus toetab muudetava teksti suurust (ingl *dynamic type).*

Näited välimisest muudatustest [7]:

- iOS seade on pööratud horisontaalsest asendist vertikaalseks või vastupidi.
- On ilmunud aktiivse telefonikõne ülemine riba.
- On avatud mitmikvaade (ingl *split view)* (hetkel toetavad ainult mõned iPad mudelid).

Nimekirjade kuvamine UITableView abil

Neljandaks oluliseks teemaks loodud õppematerjalides on UITableView. Nimekirjade kuvamine on üks tähtsamatest teemadest ja samas on suhteliselt raske, seega selle teema materjal on jaotud õppematerjalis kaheks nädalaks.

Tabeli vaade on üks tihedamalt kasutatud kasutajaliidese elemente läbi kõikide rakenduste. Tihti on vaja ekraanile viia palju rohkem infot, kui sellele üldse võib mahtuda ja nagu lauaarvuti operatsioonisüsteemidel on see probleem lahendatud kerimisega, ehk sisu üles-alla liikumisega. Kuna mobiilseadmetel ekraan on suhteliselt väike, on tabeli vaade tähtsust raske ülehinnata, ja kui vaadata sisse ehitatud iOSi rakendusi, siis võib igal pool leida ühe või mitu tabeli vaadet.

Üks lihtsamatest näidetest on telefoniraamatu rakendus – uuringute järgi on igal Suurbritannia kodanikul keskmiselt 47 telefoninumbrit salvestatud oma seadmesse ja isegi nii palju numbreid on peaaegu võimatu mahutada väiksele ekraanile nii, et kõik oleksid loetavad ja kättesaadavad.

UITableView klass representeerib tabeli vaadet, mis on nagu keritav nimekiri, kus andmed on jaotud ridadeks, mida saab omakorda jagada sektsioonideks.

Rakenduse elutsükkel

Mobiilirakendused on sisuliselt oma koodi ja sisseehitatud teekide koodi segu. Teegid annavad vajaliku infrastruktuuri, millel rakendused töötavad ja programmeerija lisab oma koodi, mis lisab rakendusele need võimalused ja selline välimus nagu oli planeeritud. Selleks, et teha seda efektiivselt tuleb mõista iOS arhitektuuri ja selle rakenduse elutsüklit.

UIApplicationDelegate on klass, mis peab olema igas rakenduses ja see vastutab rakenduse ja operatsioonisüsteemi vahelise suhtlemise eest. Rakenduse delegaat saab teavitusi erinevate süsteemi sündmuste kohta, millest üks tähtsamatest on teavitus, et rakendus on edukalt käima pandud. Seejärel läheb programmeerija kood kontrolli alla, kus saab initsialiseerida vajalikud ressursid ja näidata vastavaid vaateid (näiteks, kui kasutaja ei ole sisse logitud, siis näidata sisselogimise vaadet, aga kui kasutaja juba seda tegi, siis näidata vastavat vaadet).

Võrk

Ilma võrguühenduseta mobiilirakendused võivad ainult kasutada, töödelda ja lõpuks kuvada ainult need andmed, mis olid allalaaditud koos rakendusega või loodud lokaalselt. See oluliselt piirab hulk probleeme, mille saab rakenduse abil lahendada.

Andmete võrgust saamise programmeerimine ei ole triviaalne ülesanne, eriti mobiilsetel seadmel - ühenduse kiirus on tihti aeglasem, kui kaabelühendusega arvutitel, lisaks, kuna rakendust saab kasutada igal pool - ühendus ei pruugi olla stabiilne, ehk mõneks ajaks andmeteedastus võib peatuda.

iOS-is on sisseehitatud paljud mehhanismid, mille abil saab kergelt ühendust seadistada ja andmeid üles- või allalaadida.

Serveritehnoloogia

Tihti juhtub nii, et on vaja rakenduse andmeid salvestada, selleks võivad olla märkmed, tehtud fotod, veebilehitseja järjehoidjad või isegi läbitud mängu progress ja tulemused.

Üheks lahenduseks on hoida andmeid seadmes, aga see esitab märkimisväärseid piiranguid[9]:

1. Kasutajad aeg-ajalt paigaldavad rakendusi uuesti, või seade võib katki minna, mis omakorda võib põhjustada andmete kadu.
2. Paljudel inimestel on rohkem kui üks seade ja oleks mugav omada ligipääsu oma dokumentidele igast seadmest.
3. Tihti on kasutajatel vaja jagada andmeid info edastamiseks või koostööks.
4. Viimase ajal on levinud *push*-teavitused, näiteks, selleks, et kontrollida uute emailide olemasolu ei pea seade kogu aeg serverilt seda küsima, vaid uue emaili saabumisel server lükkab emaili seadmele ja kohaselt teavitab kasutajat.
5. Arendaja ei saa teada, kuidas rakendust kasutatakse, milliste võimaluste peale tuleb tähelepanu pöörata ja vastavalt rakendust parandada.

Seega väga paljud rakendused kasutavad servereid andmete hoidmiseks, teavituste saatmiseks ja kasutamise analüüsimiseks.

Kõikide nende eelistega tulevad ka raskused[10]:

1. Suur ajakulu. Mobiilirakenduse arendus võtab paju aega, aga serveripoolse osa arendamine võib, parimal juhul, võtta samapalju aega.
2. Kõrged nõuded oskustele. Serveri arhitektuuri arendamine täiesti erineb mobiilirakenduste arendamisest ja vajab teisi oskusi ja kogemust.

Selle kursuse jaoks on valitud sobiv kompromiss, mis võimaldab teha mobiilirakendust koos andmebaasi ühendusega.

[9] http://sdsol.com/why-does-my-app-need-a-server/

[10]https://www.raywenderlich.com/20482/how-to-choose-the-best-backend-provider-for-your-ios-app-parse-vs-stackmob-vs-appcelerator-cloud-and-more

3. Metoodika

3.1 Töö loomise protsess

Õppematerjalide loomiseks viidi kõigepealt läbi vajaduste analüüs, kasutades kvaliteetse e-kursuse loomise juhendit [8], selleks, et saada teada:

- Milliseid eelteadmisi peavad tudengid omama, et kursust edukalt läbida.
- Milline on tegelik olukord mobiilirakenduste arendamise õpetamises Tartu Ülikoolis.
- Kas sellise kursuse loomine toob praktilist kasu tudengitele.

Selleks, et alustada õppematerjalide loomist, oli vaja paika panna sihtgrupp. Selleks valiti üliõpilased.

Lisaks pandi paika, millised tehnoloogiad (teemasid) kursus vaatlema hakkab. Esimese asjana on plaanis tutvustada iOS-i arendaja vaatepunktist ja õppida programmeerimiskeelt Swift, millel hakkab kogu kursusepoolne arendus toimuma. Järgmiseks sammuks on iOS tehnoloogiate õpetamine ehk millistest osadest iOS rakendused koosnevad ja kuidas neid peab kasutama.

Lõpuks, et kinnitada saadud teadmisi ja omandada praktilisi oskusi, üliõpilased saavad võimaluse arendada täieliku rakendust, ideest ja prototüübist valmisrakenduseni. Kuna kursuse planeeritav kestvus on 16 nädalat, on igale nädalale määratud konkreetne teema, teemade nimekirja nädalate kaupa saab vaadata vastavas lisas (vt. Lisa 2).

3.2 Kursuse programmeerimiskeele valik

Üheks tähtsamaks küsimuseks oli programmeerimiskeele valik kursuse jaoks. Kuna käesolev kursus vaatleb ainult kohaliku (ingl *native*) rakenduste arendust, siis sellised lahendused nagu Xamarin[11], Cordova[12] jt. jäid vaatlusest ära (vt. peatükki Platvormi valik). Seega on mõistlik kitsendada vaatlust Objective-C ja Swift võrdlusele.

[11] https://www.xamarin.com

[12] https://cordova.apache.org

Microsofti artiklis “Choosing Programming Language” [9] on välja toodud kaheksa programmeerimiskeele omadust, mida tuleb vaadelda keele valimisel.Järgnevalt on esitatud need omadused koos vastava võrdlusega Swifti ja Objective C vahel.:

1. **Õppimise lihtsus** – Swifti arendamise ajal peeti meeles seda, et keel peab omama selget ja puhast süntaksit. Lisaks sellele on Swifti süntaks sarnane teiste laialt levinut keelte süntaksitega nagu Python või JavaScript. Kõik see teeb selle õppimine võrreldes Objective-C-ga kiiremaks ja kergemaks.
2. **Arusaadavus** – nagu eelmises punktis on mainitud, siis Swifti süntaks on puhtam ja selgem võrreldes Objective-C-ga (vt. Lisa 3).
3. **Arenduse kiirus** – Swifti range tüübisüsteem ja eritüüp Optional võimaldavad paljud vead leida kompileerimise ajal, ehk läheb vähem aega vigade otsimisele ja parandamisele.
4. **Korrektse koodi kirjutamise abi** – Swift on staatiliste tüüpidega keel, mis elimineerib paljud tüübivead kompileerimise ajal, lisaks sellele Swiftis puuduvad probleemid, mis esinevad Objective-C-l, sest Objective-C on C keele põhine ja pärib selliseid vigu nagu *goto*, *buffer overflow*, *uninitialized variables* jt. [13]
5. **Koodi jõudlus** – vt. Lisa 1.
6. **Platvormi keskkondade tugi** (ingl *platform environment support*) – mõlemad keeled on täielikult toetatud iOS-i poolt.
7. **Porditavus** (ingl *portability*) – mõlemate keelte kompilaatorid on esindatud mitte ainult Apple-i platvormidel, aga Swift ja selle standardteek on vähem seotud Apple-i ökosüsteemiga ja seda arendatakse avalikult.
8. **Vajaduseks sobilikus** – mõlemad keeled sobivad samaselt hästi iOS rakenduste arendamiseks.

3.3 Platvormi valik

Kate Abrosimova toob oma artiklis[14] järgmised kohalike (ingl *native*) rakenduste eelised:

- *Native*-koodi jõudlus on kõrgem.

[13] https://www.youtube.com/watch?v=w87fOAG8fjk aeg: 1:46:27

[14] https://yalantis.com/blog/native-vs-cross-platform-app-development-shouldnt-work-cross-platform/

- Uued võimalused on saadaval kohe, kui Apple neid valmis saab, ei pea ootama kuna kolmandad osapooled realiseerivad uute võimaluste tugi.
- Kergem ligipääs riistvarale, näiteks žestid või asukoha määramine kasutades GPSi.
- Lisatööristad, nagu Instruments, kus saab oma rakendust profileerida, ehk leida aeglasemad kohad koodis ja mäluleke (ingl *memory leak*).
- Erinevate platvormide kasutajad ootavad erinevaid kasutajaliideseid ja kasutajakogemusi, see tähendab, et üks kasutajaliides ei sobi erinevatele platvormitele.

Arvestades neid eeliseid, ja fakti, et Apple ega Google ei paku ametliku platvormidevahelist (*cross-platform*) teeki, mis tähendab, et *cross-platform* rakenduse arendaja sõltub kolmanda osapoole tarkvarast ja selle toest (ingl *support*). Selle kursuse eesmärgiks on valitud *native*-rakenduste arenduse õpetamine.

3.4 Kursuse koostamise põhimõtted

Kursuse vajaduste analüüsimisel ja eesmärkide sõnastamisel põhineti materjalile “Juhend kvaliteetse e-kursuse loomiseks” [8] ja kursuse ülesehitus ja vaadeldavad tehnoloogiad on valitud arvestades Stanfordi iOS mobiilirakenduse arendamise kursusega [2] mille korraldatakse juba mitu aastat ja mis saab positiivset tagasisidet.

4. Koostatud kursus ja õppematerjalid

4.1 Kursuse kirjeldus

Kuna mobiilirakenduste turg ja arendus muutub väga kiiresti, on edukuse põhiprintsiipiks õppimine loomise ajal (ingl *learn-by-doing*), ehk esitatakse mingi ülesanne ja seda lahendatakse varem saadud ja uusi teadmisi kasutades.

See printsiip võimaldab lühikese ajaga õppida palju materjali, samuti omandada praktilisi oskusi.

iOS rakenduste arendamise kursus Swift keele baasil[15] koosneb kolmest peamisest osas:

1. Nädalad 1–5. Sissejuhatus iOS arendusele, Swifti õppimine. Tudeng saab teada, kuidas Swiftis programme kirjutada. Materjalid on keskendunud rohkem teoreetilisele osale.
2. Nädalad 6–10. SDK (Software Development Kit) õppimine ja mobiilirakenduste arendamise algpõhimõted. Kuna sellel ajavahemikul tudeng saab algteadmisi mobiilirakenduste arendamise põhiprintsiipidest, enamik materjale on segu teoreetilistest ja praktilistest ülesannetest.
3. Nädalad 11–16. Viimase kuue nädala jooksul tudengid saavad arendada tervikliku rakenduse enda väljamõeldud ideest – prototüübist valmislahenduseni. Selles osas vaadeldakse tehniliste lahenduste ja rakenduste arenduse protsesse.

Täpsema materjali ülevaate leiab lisas (vt. Lisa 2).

Materjal on koostatud arvestades mahuhinnangut 3 EAP-d, ehk 78 tundi. Töömaht jaotub järgmiselt:

1. 15 praktikumi, iga praktikum 90 minutit. Iga praktikum koosneb teoreetilisest sissejuhatusest 30–40 minutit ja järgnevast praktilisest osast, kus kasutatakse saadud teadmisi.
2. 15 iseseisvat ülesannet, iga ülesanne 2 tundi. Iga praktikumi lõpus antakse ülesanne, mille tudengid peavad lahendama kasutades saadud teadmisi või otsida lisamaterjali antud allikatest.

[15] http://appdromeda.ee/swift

3. 26 tundi rühmatöö. Viimase 4–5 nädala jooksul õpilased saavad võimaluse rühmades arendada enda väljamõeldud rakendust algusest lõpuni.

4.2 Kursuse teoreetiline osa

Kursuse teoreetiline osa koosneb baasteadmiste omandamisest, mis on vajalikud praktiliste ülesannete lahendamiseks.

Esimese 5 nädala jooksul tegeldakse kursuse esimese osa teemadega. Esimese osa eesmärgiks on tutvustada tudengitele iOS rakenduste arendamist ning pakkuda materjale Swift programmeerimiskeele õppimiseks.

Järgmised viis nädalat, suurim osa ajast, on pühendatud teoreetiliste teadmiste omandamisele, läheb erinevate mini-rakenduste ülevaatusele, mis on realiseeritud kasutades vastava teema tehnoloogiat. Lisaks tuleb tutvustus Xcode-i kasutajaliidese elementidega, mis tulevad kasuks ülesannete lahendamisel. Eriti puudutab see kasutajaliidese ehituse tööriista Interface Builder.

Iga teoreetilise osa alguses antakse aega eelmise nädala materjalide meelde tuletamiseks vastavate küsimuste esitamisega.

Iga praktikum lõpeb sellega, et tudeng peab esitama tagasiside, kus peab vastama küsimustele:

1. Mida sa said täna uut teada?
2. Kui raske oli tänane materjal?
3. Kas on midagi, mida soovid veel teada saada selle teema kohta?

Lisaks iga tudeng võib esitada oma küsimust, millele vastust saab kas kohe, emaili teel või järgmises praktikumis.

Selline lähenemine võimaldab tudengile moodustada oma kokkuvõte praktikumi kohta ja õppejõule saada infot läbiviidud praktikumi kohta.

4.3 Kursuse praktiline osa

Praktikumi põhieesmärgiks on kinnitada teadmisi, mis on saadud teoreetilises osas. Praktikumi ajal esitatakse mini-rakenduse idee, mille lahenduseks kasutatakse vastava nädala õppematerjale.

Praktikumi käigus lisaks lahendatavatele ülesannetele esitatakse ka ülesanded iseseisvaks lahendamiseks. Vastavad ülesande tüübid on erinevad – mõned võivad koosneda valmis rakenduste täiendamisest, aga teised olemasoleva koodi/rakenduse puuduste leidmisest ja parandamisest.

Esiteks, ülesanded iseseisvaks lahendamiseks aitavad veelgi paremini materjali mõista ja teiseks, arendavad probleemilahenduse oskusi. Ülesannete lahenduse otsimise ja lahendamise oskused on väga vajalikud iga tarkvara arendamise ajal ja paljud tööandjad nõuavad neid.

Projekti koostamine praktikumis

Viimase viie õppenädala jooksul tegeldakse praktikumides täieliku rakenduse arendamisega.

Lisaks sellele tudengid saavad rohkem teada rakenduste arendamise protsessist (idee, prototüüp, MVP (Minimum Valuable Product), lõplik otsus), saavad soovitusi õige kujunduse ja kasutajakogemuse realiseerimise jaoks (trükk, värvi valik, kasutatavus ja trendid).

Lisaks teoreetilistele teadmistele saavad üliõpilased saadud projekti teadmisi kasutada enda väljamõeldud rakenduse realiseerimisel.

4.4 Rühmatöö

Rümatöö viiakse läbi paralleelselt praktikumiprojektile. Selle käigul tudengid saavad ülesandeks välja mõelda rakenduse idee, mis peab õppejõu poolt kinnitatud olema, selleks et takistada liiga lihtsad või liiga raskeid ülesandeid.

Lisaks praktilistele teadmistele tudengid teevad paremaks oma meeskonnatöö oskusi, arendavad loovat mõistmist ja lõpuks saavad peaaegu valmis üleslaadimiseks rakenduse, mis võimaldab täita oma isiklikku portfooliot juba ülikoolis.

4.5 Materjalide kättesaadavus

Materjalide kuju

Kogu kursuse materjal on kirjutatud Markdown[16] formaadis, millel on kolm eelist:

1. Kogu materjali saab alla laadida kaustadesse pandud tekstifailidena.
2. Markdown on lihtsustatud teksti vormistamise süntaks, mida saab otse HTMLi (HyperText Markup Language) tõlkida ja igas veebilehitsejas lugeda.
3. On mitu erinevat tekstiredaktorit (näiteks MarkdownPad[17] Windowsi jaoks ja MacDown[18] Maci jaoks), mis võimaldavad Markdown failide ilusal kujul lugemist.

Materjalide asukoht

Lisaks sellele, et materjale saab tekstifailidena alla laadida ja lugeda valitud tekstiredaktoris on olemas materjalide veebiversioon.

Veebiversioon koosneb staatilistest HTML dokumentidest ja CSS (Cascading Style Sheets) stiilifailidest. Need staatilised dokumendid on genereeritud otse Markdown failidest lisades neile stiile. Genereerimisega tegeleb staatiliste veebilehtede generaator Jekyll[19], mille põhiarendajaks on Tom Preston-Werner, kes on ka GitHubi kaasasutaja.

Jekyll ei sobi väga dünaamiliste veebilehtede, ehk veebirakenduste arendamise jaoks, kuna JavaScripti ja muude tehnoloogiate integreerimine on raskem, kui tavalisel arendamisviisil. Aga kuna kursuse materjalid on enam-vähem staatilised, generaatorit saab vabalt kasutada selleks, et teha materjalid saadavaks veebist.

Jekyll projekt koosneb:

1. Sisufailidest, ehk failid, mille sisu ilmub veebilehele.
2. Stiilifailidest, mis sisaldavad koodi, mille järgi kavandatakse töödeldud Markdown.
3. Konfiguratsioonifailidest, mis sisaldavad erinevaid seadeid, põhimõtteliselt, mis moel töödeldakse Markdown-faile.

[16] https://daringfireball.net/projects/markdown/

[17] https://markdownpad.com

[18] http://macdown.uranusjr.com

[19] https://jekyllrb.com

Mugav viis hallata projekti on Git[20] versioonihaldussüsteemi kasutamine, ehk sisu uuendused toimuvad läbi muutuste fikseerimise (ingl *commit)* ja nende üleslaadimise (ingl *push).*

4.6 Tehnilised andmed

Kursuse materjal vaatleb ka iOS rakenduse serveriga ühendust ja suhtlust, et rakendus saaks üles- ja allalaadida andmeid. Põhikriteeriumiteks serveri-tehnoloogia valimisel olid: seadistamise lihtsus ja ajakulud, tasuta plaanide võimalused, mobiilirakendustega integratsioon.

Parimaks lahenduseks valiti Parse Server. See on serveri-tehnoloogia, mida arendab Parse LLC, Facebook'i tütarettevõte. Lisaks andmehoiusele, Parse Server pakub kasutajate haldust (registreerimine, paroolide haldamine jne), rakenduste paigaldamist ja lisa seadistamisega *push*-teavituste saatmise võimalust.

Suureks Parse Serveri eeliseks on see, et selle arendus toimub GitHubis ja see pakub oma SDK-d mobiilirakenduste arendamiseks (iOS kaasarvatud), mis võimaldab realiseerida rakendusele andmebaasiga ühenduse väga väikese vaevaga.

Parse Server ja selle SDK on saadaval avatud lähtekoodiga[21] [22].

Kuna Parse Server kasutab MongoDB-d, on ka andmebaasi jaoks oli valitud mLab (endine MongoLab)[23], selle tasuta plaan võimaldab teha andmebaasi kuni 500MB suurusega.

Hostingu pakkujaks on valitud Heroku[24], põhiliselt Parse Serveri integratsiooni lihtsuse pärast. Tasuta Heroku plaan pakub serverit 512MB operatiivmäluga, mis lülitub välja automaatselt, kui seda pole üle 30 minutit kasutatud ja kokku 24 tunni jooksul võib maksimaalselt 18 tundi aktiivne olla. See piirang ei lase täisväärtusliku rakendust luua, aga samas võimaldab tasuta seadistada serveri ja seda kasutada õppimiseks.

[20] https://git-scm.com

[21] https://github.com/ParsePlatform/parse-server

[22] https://github.com/ParsePlatform/Parse-SDK-iOS-OSX

[23] https://mlab.com

[24] https://heroku.com

5. Kokkuvõte

Käesoleva töö eesmärgiks on koostada kursuse „iOS mobiilirakenduste arendamine Swift keele baasil“ kava ja vastavad õppematerjalid. Materjalid on koostatud nii, et need läbinud tudeng saaks võimalikult täieliku ettekujutuse iOS rakenduste arendusest ja omandaks lisaks teoreetilistele teadmistele ka praktilist kogemust selleks, et enda rakenduste ideid ellu viia.

Töö sissejuhatuses lugeja saab teada töö aktuaalsusest ja miks selline kursus tuleb kasuks Tartu Ülikooli õppekavale.

Teine peatükk tutvustab iOS-i ja Swifti ajalugu ja õppematerjalides kasutatud tehnoloogiate nimekiri ja kirjeldus.

Pühendatud metoodikale peatükk kirjeldab töö loomise protsessi, kursuse koostamise põhimõtteid ja kaks osa, mis seletavad, miks kursuse programmeerimiskeeleks oli valitud Swift ja miks oli valitud *native*-rakenduste arendamise õpetus, aga mitte *cross-platform* arendus.

iOS mobiilirakenduste kursus Swift keele baasil koosneb õppematerjalidest, mis on jaotud 15 nädalaks, kus igal nädalal vaadeldakse konkreetne peatükk. Kursuse alguses tudengid tutvuvad iOS arendusega ja hakkavad õppima Swift programmeerimiskeelt ja tähtsaid klasse, mida tuleb igapäevases iOS arendaja elus vaja. Kursuse järgmine osa koosneb rakendust moodustavate osade õppimisest – näiteks, kuidas sisu ekraanile näidetakse võik kuidas teha nii, et rakendus näeks hästi välja erinevate ekraanisuurustega seadmetel. Lisaks tudengid saavad teada iOS rakenduse elutsüklist, kuidas efektiivselt nimekirju kuvada ning andmeid veebist allalaadida.

Kursuse lõpus tudengitel tekib võimalus rühmades oma rakenduse idee ellu viia, et saadud teadmised kohe tööle panna ning praktilisi oskusi arendada.

Kokkuvõtteks võib öelda, et valminud kursusematerjalid ja korraldus on heaks aluseks vastava kursuse läbiviimiseks ja kursus ise pakub piisavat väärtust selle läbinud tudengitele.

6. Kasutatud materjalid

[1] R. Bareiss ja M. Griss, „A story-centered, learn-by-doing approach to software engineering education,“ ACM, New York, 2008.

[2] P. Hegarty, „Developing iOS 8 Apps with Swift,“ Stanford, 2015. [Võrgumaterjal]. Available: https://itunes.apple.com/us/course/developing-ios-8-apps-swift/id961180099. [Kasutatud 09 05 2016].

[3] Apple Inc., „The Swift Programming Language (Swift 2.2),“ Apple Inc., 2 juuni 2014. [Võrgumaterjal]. Available: https://itun.es/us/jEUH0.l. [Kasutatud 9 mai 2016].

[4] Apple Inc., „The Foundation Framework,“ Apple Inc., 22 oktoober 2013. [Võrgumaterjal]. Available: https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/ObjC_classic/. [Kasutatud 9 mai 2016].

[5] G. Wells, „The Future of iOS Development: Evaluating the Swift Programming Language,“ 19 mai 2015. [Võrgumaterjal]. Available: http://scholarship.claremont.edu/cmc_theses/1179. [Kasutatud 9 mai 2016].

[6] Apple Inc., „UIKit Framework Reference,“ Apple Inc., 17 september 2014. [Võrgumaterjal]. Available: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIKit_Framework/. [Kasutatud 9 mai 2016].

[7] Apple Inc., „Understanding Auto Layout,“ Apple Inc., 21 märts 2016. [Võrgumaterjal]. Available: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/AutolayoutPG/#//apple_ref/doc/uid/TP40010853-CH7-SW1. [Kasutatud 9 mai 2016].

[8] L. Pilt, T. Plank, A. Villems, M. Varendi, M. Kusmin, K. Kusnets, M. Dremljuga-Telk ja E. Koitla, „Juhend kvaliteetse e-kursuse loomiseks,“ 5 märts 2008. [Võrgumaterjal]. Available: http://www.e-ope.ee/images/site_0/FINAL_trykk.pdf. [Kasutatud 9 mai 2016].

[9] C. Britton, „Choosing a Programming Language,“ Microsoft Inc., 10 jaanuar 2008. [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/cc168615.aspx. [Kasutatud 9 mai 2016].

Lisad

I. Objective-C ja Swift jõudluse võrdlus

Jõudlus on arvutatud sorteerimisalgrotimite kiiruse võrreldes.

Kood:

- Objective-C algoritmid: https://github.com/jessesquires/objc-sorts
- Swift algoritmid: https://github.com/jessesquires/swift-sorts

Tarkvara:

- OS X El Capitan 10.11.4, Xcode 7.3.1

Riistvara:

- MacBook Pro (Retina 13-inch, Late 2012), 2.5GHz Intel Core i5, 8GB 1600MHz DDR3 operatiivmälu

Tulemused:

n = 10000	std lib sort	Quick	Heap	Insertion	Selection
Swift	0.001	0.001	0.002	0.144	0.106
Objective-C	0.003	0.007	0.019	0.148	3.626
Vahe	3x	7x	9.5x	1.02x	34.2x

Tulemused on esitatud sekundites, arvutatud 10 jooksutamisest aritmeetilise keskmisena. Testid on toodud väljalülitatud optimisatsioonidega (-O0), kuna testid sisselülitatud optimimisatsioonidega annavad sarnaseid tulemusi.

II. Peatükkide loetelu

Nädal	Teema	Alamteemad
1	Sissejuhatus	• iOS ajalugu • Sissejuhtaus Xcode • Esimene Xcode projekt • Xcode kasutajaliides • Esimene iOS rakendus
2	Swift	• Sissejuhatus Swift • Xcode Playground • Muutujad, tüübid, andmestruktuurid • Käsuvoog ja funktsioonid • Klassid, struktuurid • Optional-tüüp • Protokollid ja laiendused • Vigade käsitlemine
3	Foundation teek	• Foundation teek • Andmehoius • Mäluhaldus • Aja käsitlemine • Teavitused
4	UIKit	• Vaated • Vaatekontrolleid • Storyboard • UITouch • UIControl • UITextField ja UITextView
5	Auto Layout	• Adaptiivne kujudus • Visuaalne Auto Layout • Programmaatiline Auto Layout • Animatsioonid
6	Table View I	• Tabeli vaated - UITableView • Tabeli read - UITableViewCell • Tabeli täiendamine - UITableViewDataSource
7	Table View II	• Tabeli sündmused - UITableViewDelegate • Custom UITableViewCell • Tabeli vaade jõudlus

8	Rakenduse elutsükkel	• Elutsükkel • Application Delegate
9	I/O	• NSUserDefaults • Core Data
10	Võrk	• NSURLSession • JSON • Asünkroonsed päringud
11	Projekt I	• Idee • Tingimused • Prototüüp • Andmebaas
12	Projekt II	• Seadistus • Vaated • Tekst-postitused
13	Projekt III	• Pildi-postitused • Võrk • Profiil
14	Disain	• Disainist • Ikoonid • Kasutajakogemus
15	Kasulikku	• Kasulikud materjalid
16	Projektide esitlus	

III. Objective-C ja Swift näidisprogrammid

Olgu meil on vaja luua klassi Kasutaja, millel on nimi ja vanus ja see peab oskama teretama. Peame arvestama, et võib tekkida probleem kasutaja initsialiseerimisel ja seega vastav muutuja võib tühi (ingl *null*) olla. Lisaks puudub garantii, et iga klass, kes väidab, et oskab teretada tõesti seda teeb.

Objective-C programm

```
// Teretaja.h
@protocol Teretaja <NSObject>
@optional
-(void)utleTere;
@end

//Kasutaja.h
#import <Foundation/Foundation.h>
#import "Teretaja.h"

@interface Kasutaja: NSObject<Teretaja>
@property (strong, nonatomic) NSString* nimi;
@property (assign, nonatomic) int vanus;
- (id) initWithNimi: (NSString*)name andVanus: (int)age;
@end

//Kasutaja.m
#import "Kasutaja.h"
#import "Teretaja.h"

@implementation Kasutaja
@synthesize nimi;
@synthesize vanus;

- (id)initWithNimi:(NSString*)name andVanus:(int)age{
 self = [super init];
 if (self){
  nimi = name;
  vanus = age;
 }
 return self;
}
-(void)utleTere{
 NSLog(@"Tere, maailm");}
```

Programmi kasutamine:

```
Kasutaja *kasutaja = [[Kasutaja alloc]initWithNimi:@"Juri" andVanus:22];
id<Teretaja> teretaja = kasutaja;


if (teretaja != nil){
 if ([teretaja respondsToSelector:@selector(utleTere)]){
  [teretaja utleTere];
 }
}
```

Swift programm

```
import Foundation


@objc
protocol Teretaja {
 optional func ütleTere()
}


class Kasutaja : NSObject, Teretaja{
 var nimi: String
 var vanus: Int
 init(nimi: String, vanus: Int) {
  self.nimi = nimi
  self.vanus = vanus
 }


  func ütleTere(){
  print("Tere, maailm!")
 }
}
```

Programmi kasutamine:

```
var kasutaja = Kasutaja(nimi: "Juri", vanus: 22)
var teretaja : Teretaja? = kasutaja
teretaja?.ütleTere?()
```

Nagu näideprogrammidest saab näha, et kirjutatud Swiftil programm säilitades funktsionaalsust võtab kaks korda vähem koodiridu, mis on selgem lugemiseks ja mõistmiseks.

IV. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Juri Noga**,
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
iOS mobiilirakenduste arendamise kursuse õppematerjal Swift keele baasil,
(*lõputöö pealkiri*)

mille juhendajad on Tauno Palts ja Marina Lepp
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **12.05.2016**