TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDK70LT

Artjom Sinkin 141944

"KEGLER" MOBIILRAKENDUSE ARENDUS

Magistritöö

Juhendaja: Jekaterina Tšukrejeva
Magistrikraad
Õppejõu assistent

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Artjom Sinkin

09.05.2016

Annotatsioon

Antud töös on vaadeldud „Kegler“ rakenduse arendamine mobiilplatvormide jaoks.

„Kegler“ on rakendus mõeldud kegli mängijale, selle peamised funktsioonid on mängude ja võistluste tulemuste salvestamine ja mängija statistika kuvamine.

Töö eesmärk on arendada „Kegler“ rakendust kahe mobiilplatvormide jaoks: Google Android ja Apple iOS.

„Kegler“ rakendus peab olema arendatud võimalikult kiiresti ja limiteeritud finantskuludega. Seda võib saavutada kasutades sobivat mobiilrakenduste arendamise raamistiku, mis on arendamiseks lihtne ja mugav. Valitud raamistik peab kasutama levinud programmeerimiskeelt, arusaadavad ja omandamiseks lihtsad tehnoloogiad ja peab võimaldama rakenduse arendajate hulka vähendama.

Antud töö käigus analüüsitakse levinud mobiilrakenduse arendamise raamistikud. Võrdlemise käigus valitakse kõige sobivamaid variante ja need on praktikas proovitud rakenduse arendamiseks. Kogutud info on analüüsitud ja „Kegler“ rakenduse arendamiseks on valitud kõike sobivaim raamistik, arvestades selle rakenduse omaduste ja nõuetega.

„Kegler“ rakendus on arendatud Android ja iOS platvormide jaoks, arendusprotsess valitud raamistiku kasutamisel on kirjeldatud ja selle tulemus on analüüsitud arvestades projekti nõuetega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 52 leheküljel, 7 peatükki, 45 joonist, 0 tabelit.

Abstract

"Kegler" mobile application development

This work describes the„Kegler“ mobile application development.

The „Kegler“ is a mobile application developed for kegel players, application`s main feautures are game results and statistics tracking.

The aim of this work is to develop „Kegler“ mobile application for both Google Android and Apple iOS platforms.

The „Kegler“ application should be developed in a limited periood of time and money resources invested. This can be achieved using mobile development framework suitable for this project, which enables to simplify development process and minimise amount of developers involved.

In this work popular mobile development frameworks are analyzed and the most suitable framework for „Kegler“ project needs is determined. Using selected framework „Kegler“ application is developed for Android and iOS plarforms to verify the analysis result.

The thesis is in Estonian language and contains 52 pages of text, 7 chapters, 45 figures, 0 tables.

Lühendite ja mõistete sõnastik

IDE	Integrated Development Environment, integreeritud arenduskeskkond
API	Application program interface, rakendustarkvara liides
UI	User Interface, kasutajaliides

Jooniste loetelu

Sisukord

1. Sissejuhatus

Antud töös on vaadeldud „Kegler“ rakenduse arendamine mobiilplatvormide jaoks.

„Kegler“ on rakendus mõeldud kegli mängijale, selle peamised funktsioonid on mängude ja võistluste tulemuste salvestamine ja mängija statistika kuvamine.

Töö eesmärk on arendada „Kegler“ rakendust kahe mobiilplatvormide jaoks: Google Android ja Apple iOS.

„Kegler“ rakendus peab olema arendatud võimalikult kiiresti ja limiteeritud finantskuludega. Seda võib saavutada kasutades sobivat mobiilrakenduse arendamise raamistiku, mis on arendamiseks lihtne ja mugav. Valitud raamistik peab kasutama levinud programmeerimiskeelt, arusaadavad ja omandamiseks lihtsad tehnoloogiad ja peab võimaldama rakenduse arendajate hulka vähendama.

1.1 Taust ja probleem

Mobiilrakenduste turg aktiivselt suureneb, Apple App Store ja Google Play rakenduste ja kasutajate arv kasvab kiiresti. Mobiilsed rakendused on perspektiivne võimalus tulu saamiseks, mõned IT ettevõtted proovivad ennast sellel sfääril.

Apple iOS ja Google Android on kõige populaarsemad mobiilsed platvormid, need koos omavad 98.4% turust (Q42015). [1]

Worldwide Smartphone Sales to End Users by Operating System in 4Q15 (Thousands of Units)

Operating System	4Q15 Units	4Q15 Market Share (%)	4Q14 Units	4Q14 Market Share (%)
Android	325,394.4	80.7	279,057.5	76.0
iOS	71,525.9	17.7	74,831.7	20.4
Windows	4,395.0	1.1	10,424.5	2.8
Blackberry	906.9	0.2	1,733.9	0.5
Others	887.3	0.2	1,286.9	0.4
Total	**403,109.4**	**100.0**	**367,334.4**	**100.0**

Source: Gartner (February 2016)

Joonis 1. Mobiilplatvormidejaotus.

Android ja iOS on sihtplatvormid potentsiaalsete klientide saavutamiseks ja tulu saamiseks.

Rakendus peab olema arendatud mõlemade platvormide jaoks, aga need platvormid on rakenduste arendamise põhimõttest erinevad ja iga platvormi jaoks peab olema arendatud oma rakenduse versioon.

Kiiresti muutuval turul ettevõtted on huvitud mõistlikkudes kuludes arvestades ajapiiritega.

Rakenduse arendamise rahalised ja ajalised kulud sõltuvad raamistikust, sest uued raamistikud lahendavad palju probleeme ja võimaldavad arendada kasutades šabloone, valmis UI elemente ja erinevaid komponente, vähendades sellega rakenduse arenduskud.

1.2 Ülesande püstitus

Töö peamine ülesanne on optimaalse raha- ja ajakuludega „Kegler“ rakenduse arendamine kahe mobiilplatvormide jaoks: Apple iOS ja Google Android.

Sellega kaasneb erinevate mobiilrakenduste arendamise raamistikude analüüs ja kõige optimaalsema varianti valimine.

Arvestades „Kegler“ rakenduse nõuetega valitakse kõige optimaalsem raamistik ja arendatakse rakendust.

„Kegler“ rakenduse arendamiseks valitud raamistik peab võimaldama realiseerida kõiki eelnimetatud nõudeid.

1.3 Metoodika

„Kegler“ rakenduse arendamiseks on vaja valida sobiva mobiilrakenuduse arendamise raamistiku.

Erinevatest infoallikatest otsitakse kõige populaarsemad raamistikud, mis vastavad rakenduse nõuetele. Nendest on koostatud suur nimekiri.

Need raamistikud on antud töös kirjeldatud ja teoreetiliselt analüüsitud. Analüüsides ja võrreldes nende omadused valitakse kõige sobivamaid variante.

Kolm raamistikku on katsetatud „Hello World“ rakenduse arendamisel ja on praktikas kasutamisel analüüsitud.

Analüüsi kokkuvõtteks on valitud kõige sobivaim raamistik arvestades „Kegler“ rakenduse omaduse ja nõuetega.

„Kegler“ rakendus on arendatud kasutades valitud raamistiku.

1.4 Ülevaade tööst

Antud töö esimene etapp on natiivsete arendusviiside ja nende alternatiivide kirjeldus ning teoretiline analüüs. On kirjeldatud erinevare mobiilarenduse raamistikude omadused, nende eelised ja puudused on analüüsitud ja omavahel võrreldud. Analüüsi tulemuseks on valitud kõige sobivaim raamistik „Kegler“ rakenduse aredamiseks.

Teoreetilise analüüsi käigus kolm parimat variante on katsutud praktikas „Hello World“ rakenduse arendamiseks, mis võimaldab proovida arendusprotsessi ja tutvuda raamistikuga ja selle arendusmeetoditega.

Praktilise ja teoreetilise analüüsi tulemused on võrreldud ja kokkuvõttes on valitud kõige sobivaim raamistik „Kegler“ rakenduse arendamiseks.

Valitud raamistiku kasutades on arendatud „Kegler“ rakenduse Android ja iOS versioonid, arendamise protsess on kirjeldatud, selle käigus on välja selgitud raamistiku omadused rakenduse nõuete täitmises.

Tehtud töö tulemused on teoreetilise analüüsiga võrreldud. Kogu töö tulemused on analüüsitud.

2. “Kegler” rakenduse ülevaade

„Kegler“ on rakendus mõeldud kegli mängijale, selle peamised funktsioonid on mängude ja võistluste tulemuste salvestamine ja mängija statistika kuvamine.

“Kegler” on tehtud Concise Systems OÜ tellimusel, see on äriprojekt, rakendus on tasuline Android ja iOS platvormidel. Projekti tasuvus sõltub rakenduse arenduskuludest, Concise Systems OÜ on huvitatud arenduskulude minimiseerimises, et projekti tasuvust tagama.

Saksamaal on olemas umbes 21 miljon kegel mängijaid, kellest 4 miljonit mängivat regulaarselt. 276 tuhad mängijat on registreeritud kegel sprotkluubide liikmed, kes osalevad turniirides ja regulaarselt trennivad. Kegel klubide arv Saksamaal on umbes 10 tuhad. Kogu maailmas on mängijate arv kordades suurem. [1]

„Kegler“ rakenduse sihtrühm on kõik kegel mängijad, kes mängivad regulaarselt ja on huvitatud oma tulemuste organiseerimises.

Konkurents turul on minimaalne, Google Play Store-is ja Apple AppStore-is ei ole Saksamaa-tüübi kegelmängu statistika rakendusi. [3]

Lähimad konkurentrakendused on „Bowling“ mängutüübi jaoks, „Bowling“ on väga populaarne mängutüüb, aga selle reeglid on Saksamaa-tüübi kegelmänguga erinevad ning ei sobi selle tulemuste salvestamiseks ja statistika kuvamiseks.

Antud töö skoopis on arvestatud „Kegler“ rakenduse nõued, mis on mobiilirakendusega arendamise raamistikkudega seotud:

- N1: „Kegler“ rakenduse Android versioon.
- N2: „Kegler“ rakenduse iOS versioon.
- N3: Vasakpoolse menüüga UI.
- N4: Omapärane „Kegler“ UI stiil.
- N5: Mängude nimekiri pealehel.
- N6: Lokaalne andmebaas.

- N7: Mängu tulemuste sisestamine.
- N8: Mängija statistika kuvamine.

Valitud mobiilrakenduste arendamise raamistik peab võimaldama realiseerida kõik eelmainitud nõued.

3. Mobiilarenduse raamistikud

Antud peatükkis on natiivse arenduse raamistikude võrdlus võimalikke multiplatvormsete alternatiividega. Teoreetilise analüüsi käigus valitakse kõige sobivaim variant „Kegler“ rakenduse arendamiseks.

Vastavalt püstitud ülesannele „Kegler“ rakendus peab olema realiseeritud optimaalse raha- ja ajakuludega kahe mobiilplatvormide jaoks: Apple iOS ja Google Android.

Valitud mobiilarenduse raamistik peab vastama püstitud ülesannele, selleks on jargmised kriteeriumid:

- iOS platvormi toetus.
- Android platvormi toetus.
- Arendamiseks nõutud töömaht.
- Kasutud tehnoloogiate lihtsus.
- Arendatud rakenduse kvaliteet.
- Litsensi hind.

iOS ja Android platvormide toetus on “Kegler” rakenduse nõue. Arendamiseks nõutud töömaht ja kasutatud tehtoloogiate lihtsus mõjub projekti raha- ja ajakuludele, väiksem töömaht võimaldab minimiseerida arendajate hulka ja nende töötunnide arvu. Kasutatud tehnoloogiate lihtusus võimaldab värbama ilma spetsiifilise kogemuseta arendajaid, kelle töötasu on võrlemisi madalam. Valitud raamistik peab kasutama levinud programmeerimiskeelt, arusaadavad ja omandamiseks lihtsad tehnoloogiad ja peab võimaldama rakenduse arendajate hulka vähendama. Raamistiku litsensi hind mõjutab projekti rahalise kuludele otseselt.

Lähtudes erinevate raamistikude kriteeriumitest valitakse kõige sobivaima variant “Kegler” rakenduse arendamiseks.

3.1 Natiivne arendus

“Native” arendamine on platvormi spetsiifilise IDE kasutamine. See tähendab, et Google Android platvormi jaoks kasutatakse Android Studio ja Apple iOS jaoks Xcode. Selline lähinemine on nii Google kui ka Apple soovitatav, sest nende oma SDK ja platvormi spetsiifilise arenduskeskonna kasutamine võimaldab realiseerida kõik platvormi spetsiifilisi funktsioone ja kasutada kõik platvormi API-d.

Jargmistes peatükkides käsitletakse Apple iOS ja Google Android natiivsed keskkonad.

3.1.1 Apple iOS

Apple ametlik IDE iOS arendamiseks on Xcode. Viimane versioon on Xcode 7.

Xcode 7 pakkub kõiki mida on vaja iOS arendamiseks. On olemas kõik SDK-d ja tööristad iPhone, iPad, iPod jaoks arendamiseks. Xcode võimaldab realiseerida kõik arendusprotessi sammu, alustades UI disainist, koodi kirjutamisest ja lõpetades testimisega nii emulaatoril kui ka reaalsel seadmel (iPhone, iPod, iPad).

Xcode 7 IDE kombineerib ennas Cocoa raamistiku ja Swift programmeerimiskeelt, mis teeb arendusprotsessi lihtsamaks. Xcode 7 pakkub ka Objective-C keele kompilaatorit, aga Swift on uus Apple jaoks arendamiseks soovitatav programmeerimise keel, mida kasutatakse kõiki Apple rakenduste arendamiseks. [4]

Swift on võimas ja turvaline programmeerimis keel, mida arendab Apple Inc. Swift esimine version ilmus 2 Juunis 2014. Alates versioonist 2.2 Swift omab Apache License 2.0.

Swift keele peamised omadused:

- Turvalisus (süntaks ja tunnusjooned)
- LLVM compiler – võimaldab koodi kiiret kompilatsiooni ja käivitamist.
- Sulgemised on ühendatud funktsiooni viitega
- Mitmeväärtuste tagastamine
- Võimsad ptorokoolid mida võib laiendada oma koodiga. [5]

Xcode 7 omab kasutajaliide loojat, see optsioon võimaldab arendada UI ilma koodi kirjutamist. Saab kasutada standaartsed elementid UI loomiseks ja näha tulemuse live versioonis.

Xcode 7 on mugav ja efektiivne IDE, mis võimaldab arendada ilusaid rakendusi spetsiaalselt iOS jaoks, mis garanteerib et nad töötavad kiiresti ja ilma probleemideta seadme API-de kasutamisel.

Xcode 7 töötab ainult Mac OS-ga, see tähendab et igal arendajal iOS tiimis peab olema Apple arvuti. On vaja osta Apple Developer litsensiooni, mis maksab 99USD aastas. [6]

3.1.2 Google Android

Google Inc. ametlic IDE Android arendamiseks on Android Studio. Esimene versioon ilmus 16 Mail 2013. Android Studio on tasuta IDE „Apache Litcense 2.0“ litsensiga.

Android Studio baseerib JetBrains IntelliJ IDEA IDE-s. Android Studio on arendatud Android arendamiseks. On olemas versioonid Windows, Mac OS X ja Linux jaoks. Android Studio-sse on sisseehitatud „Android Development Tools“ tööriistad ja on olemas kõik vajalikud tööristad arendamiseks, Android seadmete API-de kasutamiseks ja testimiseks.

Android Studio kasutab Java koodi toimetajat, mis on kasutatud koodi kompileerimiseks ja analüüsiks. On olemas iga Android versioonide ja paljude reaalsete seadmete emulaatorid, mis võimaldavad testida rakendust lai seadmete hulgas, isegi sellel juhul, kui ei ole võimalust neid osta reaalseks testimiseks. [7]

3.1.3 Native arenduse analüüs

Lähtudes kvaliteedi ja funktsionaalsuse põhimõttest kõige parem variant on kasutada ametlik IDE-t mis on väljatöötatud konkreetse platvormi jaoks. iOS platvormi jaoks see on Xcode ja Android platvormi jaoks Android Studio.

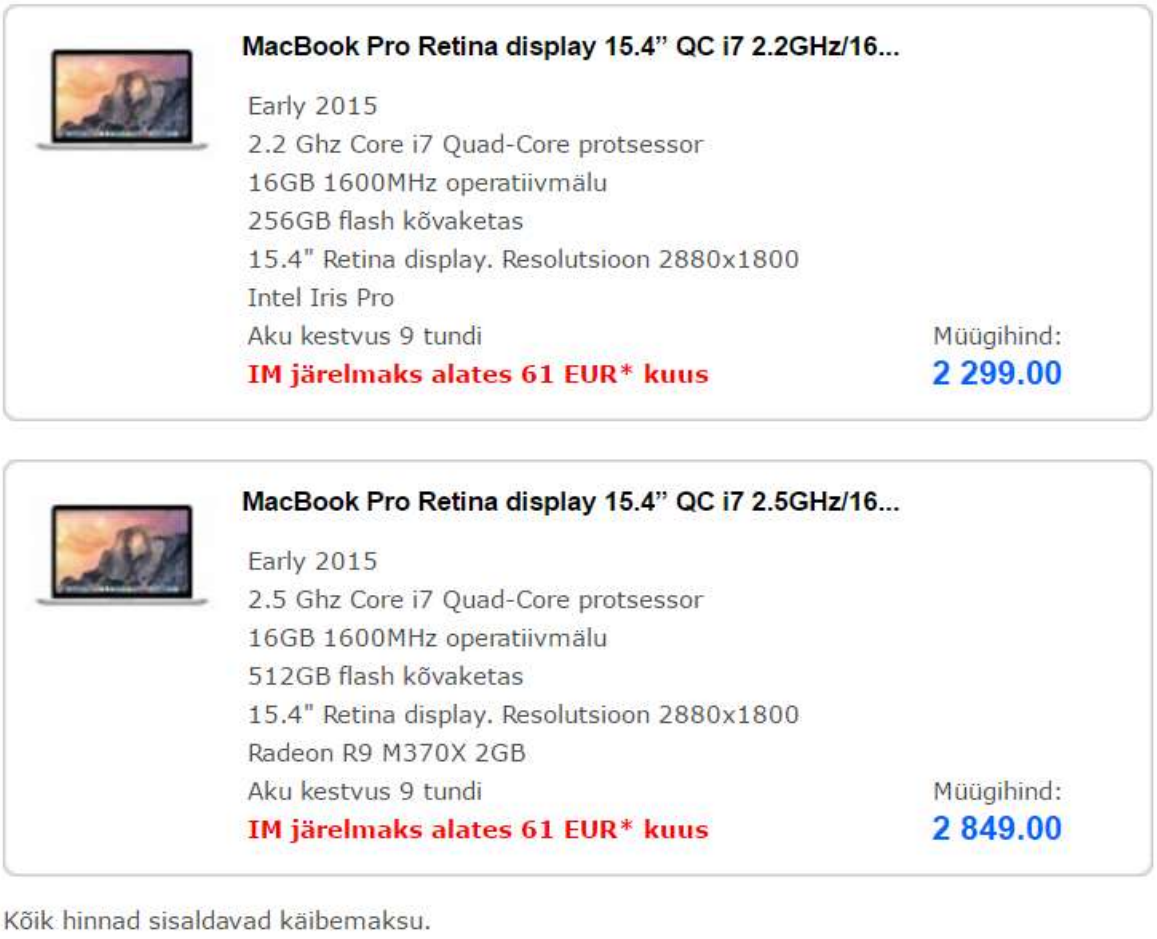
Selline lähenemine tähendab, et rakenduses saab kasutada kõiki platvormi API-d ja platvormi spetsiifilisi UI komponente.

Peab arvestama seda, et selline lähenemine tähendab ühe rakenduse versiooni arendamist iga platvormi jaoks.

“Kegler” projekti nõued N1 ja N2 on iOS ja Android rakenduse versioonid. Projekti raames peab arendama rakenduse kaks erinevat versioone, üks iOS jaoks kasutades Xcode, ja teine

Android jaoks kasutades Android Studio. Selleks peab vormistama kaks tiimi, üks iOS ja teine Android versiooni arendamiseks. See võib kahekordistada programmeerijate arvu ja sellega tuntuvalt suurendada arenduskulud.

iOS tiimis igal arendajal peab olema Apple arvuti Xcode kasutamiseks. On mugavam kasutada MacBook Pro 15” selleks, et arendajatel oli võimalus kodus töötada. MacBook hinnad varjeeruvad suuruses 2 299.00 EUR kuni 2 845.00 EUR (hinnad IM Arvutid internetipoes, sisaldavad käibemaksu). [8]



Joonis 2. MacBook hinnakiri.

Väikese projekti jaoks see võiks olla liiga suureks investeeringuks.

Xcode 7 IDE-s on kasutatud programmeerimiskeel Swift, kuna see keel on uus, tööturul on raske leida kogenud spetsialiste ja nende töötasu on võrdlemisi suurem kui Eesti keskmine programmeerijate töötasu. Kuna Swift on võrdlemisi uus programmeerimiskeel (ilmus 2014 aastal) on raske arvestada Swift programmeerijate keskmist palka. Xcode vanem programmeerimiskeel on Ojektorijenteeritud C.

Keskmine netokuupalk Eestis Objektorienteeritud C-programmeerijal on 1539 EUR [9]

Keskmine netokuupalk Eestis Java-programmeerijal on 1626 EUR [10]

Native arendamisel potentsiaalne toetatud platvormide arvu suurendamine on raskendatud sellega, et iga platvormi jaoks kasutatakse oma IDE ja API-d, ja iga rakendus peab olema arendatud uuesti kasutades platvormispetsiifilist programmeerimiskeelt. Iga uue platvormi jaoks (nt. Windows Mobile) on vaja uue tiimi vormistada ja värbama uusi arendajaid vajaliku platvormi arendamise kogemusega. Selline lähenemine võttab palju aega ja on rahaliselt kulukas.

3.2 Multiplatvormne arendamine

Natiivse arendamise alternatiiv on multiplatvormne arendamine. Selline lähenemine eeldab ühe koodi versiooni kirjutamist erinevate platvormide jaoks. See tähendab, et arendatakse rakenduse üks versioon mis töötab erinevate platvormidega. Selline lähenemine teeb arendusprotsessi lihtsamaks, kiiremaks ja võimaldab vähendada arendajate arvu tiimis ja sellega vähendada rahalised kulud.

Jargmistes peatükkides on ülevaadatud enim kasutatud ja populaarsed raamistikud multiplatvormseks arendamiseks.

3.2.1 Xamarin

Xamarin on standart ettevõte mobiilarenduseks. See raamistik võimaldab ettevõte äri ulatama kõiki peamiste mobiilplatvormidele: iOS, Android, Windows Mobile. Xamarin rakendused on 100% native rakendused ja kasutavad ühe koodibaasi. Koodibaas on standartiseeritud C# keeles, ja umbes 75% koodist on sama erinevate platvormide jaoks.

Ühine koodibaas võib tõsiselt minimiseerida nõutud töömahu rakenduse arendamiseks.

Xamarin on multiplatvormne mobiilrakenduste arendamise raamistik. Programmeerimiseks on kasutatud C# keel. Kirjutatakse koodi C# keeles kasutades kõik platformi SDK-d ja API-d. Xamarin kompileerib rakenduse erinevaid versioone erinevate platvormide jaoks.

On toetud järgmised platvormid:

- iOS

- Android
- Windows Mobile

Xamarin arhitektuur võimaldab luua natiivseid rakendusi, mida töötavad väga kiiresti ja kasutavad native UI komponente. Teoorias see tähendab, et Xamarin rakendused töötavad nii nagu platvormispetsiifiliselt tehtud (kasutades Xcode iOS jaoks ja Android Studio Android jaoks).

Xamarin on tasuline tarkvara. Litsensi hind varjeerub vahemikul 999 – 1899USD aastas iga arendaja kohta. [9]

3.2.2 Apache Cordova(PhoneGap)

Apache Cordova (varem PhoneGap) on mobiilserakenduste arendamise raamistik, mis oli arendatud Nitobi-ga. Adobe Systems ostas Nitobi 2011 aastal ja pärast avaldas Apache Cordova tasuta versiooni Apache 2.0 litsensiga.

Apache Cordova võimaldab arendada mobiilrakendusi kasutades jargmised veebitehnoloogiaid:

- HTML5
- JavaScript (jQuery, AngularJS j.t)
- CSS3 (Sass)

On toetud järgmised platvormid:

- iOS
- Android
- Windows Phone
- BlackBerry
- Bada
- Symbian

- Ja teised.

Veebitehnoloogiate kasutamine tähendab, et iga veebiarendaja saab arendada mobiilrakendusi kasutades Apache Cordova raamistiku.

Apache Cordova on väga populaarne mobiilrakenduste arendamise raamistik. Tänu populaarsusele on olemas suur Apache Cordova kasutajate kogukond ja suur erinevate foorumite ja juhendajate arv, kus saab õppida rakendusi arendada kasutades Apache Cordova. Algaja jaoks see on mõjutav faktor, sest veebitehnoloogiate baasteadmisega saab alustada multiplatvormseid rakendusi arendada ja vajadusel leida vastusi arendamise küigus tekkinud küsimustele.

Apache Cordova peamised eelised:

- Vaatamata arendaja kogemusele peaaegu igaühel on HTML5 ja JavaScript baasteadmised, platvormispetsiifiline kogemus pole vajalik.
- Suur juhendite arv ja detaalne dokumentatsioon võimaldab kiiresti arendamist alustada ja kogemust suurendada.
- Apache Cordova rakendused installeeritakse ja kasutaja jaoks näevad välja nagu tavalised native rakendused.
- Apache Cordova kasutab plugin arhitektuuri, mis tähendab et native seadmete API-d saab kasutada lisades vajalikke mooduleid.
- Open-source ja tasuta. [10]

Peab arvestama sellega, et hübriidsed, veebitehnoloogiatel baseerivad rakendused töötavad aeglasem kui natiivsed.

3.2.3 Appcelerator

Appcelerator on mobiilrakenduste arendamise ramistik, mis kiirendab „time to market“ kasutades multiplatvormset arendamist ja testimist. Appcelerator võimaldab arendada rakendusi JavaScript keeles. Appcelerator pakkub ühtseid JavaScript API-d erinevate platvormide jaoks, mis toetab natiivse platvormi tehnoloogiaid. „Alloy MVC“ raamistik võimaldab kasutada native UI komponente, mis annab natiivset kiirust rakendusele.

On toetud järgmised platvormid:

- iOS
- Android
- Windows Mobile

Appcelerator peamised eelised:

- JavaScript API
- Native UI komponentid
- Analüüsi, tulemuslikkuse juhtimise ja seire raamistik.
- Privaatne pilv ettevõttetele

Appcelerator võimaldab kasutada umbes 60-90% koodi erinevate platvormide jaoks. Appcelerator raamistikuga loodud rakendused on täiesti natiivsega sarnased, sest kasutavad natiivsed API-d ja UI komponente. Appcelerator on alati uuendatud ja toetab kõiki platvormide uuendusi.

Appcelerator raamistik sisaldab tööristeid rakenduse prototüübi arendamiseks, prototüübid on production versiooniga täiesti integreeriv, mis kiirendab arendustsükle.

JavaScript komponentid on taaskasutavad erinevate platvormide jaoks.

Sisseehitatud testimise tööristad vähendavad rakenduse testimise aja kuni 90%, vähendades sellega arenduskulud kuni 40%.

Appcelerator on tasuline tarkvara, hind on 259USD aastas iga arendaja kohta. [11]

3.2.4 Sencha Touch

Sencha Touch on HTML5 raamistik multiplatvormsete veebirakenduste arendamiseks, mis kasutaja jaoks näevad välja nagu tavalised rakendused. Sencha Touch on Apache Cordova-ga kokkusobiv, Sencha rakenduse kompileerimiseks saab kasutada Cordova pakkijat.

Sencha võimaldab arendada mobiilrakendusi kasutades järgmiseid veebitehnoloogiaid:

- HTML5
- JavaScript
- CSS3

Sencha pakkub arendajatele „Sencha Architect“, mis võimaldab luua rakenduse UI-t visuaalselt ilma koodi kirjutamata. On olemas Sencha Eclipse plugin mis toob palju mugavaid funktsioone Sencha rakenduste arendamiseks.

Sencha pakkub suurt UI komponentide kataloogi ja erinevaid UI teemasid, arendajad saavad kasutada šablooni selleks, et kiiresti arendada natiivse välimusega UI-d.

Sencha kasutab platvormi spetsiifilisi API-d ja SDK-d rakenduse kompileerimiseks erinevate platvormide jaoks, sealhulgas:

- iOS
- Android
- Windows Phone
- BlackBerry
- Symbian

Sencha on tasuline tarkvara, hinnad varjeeruvad vahemikus 6 475 - 9 475USD aastas viie arendajate kohta. [12]

3.2.5 Ionic Framework

Ionic Framework on 2013 aastal ilmunud open-source SDK hübriidsete mobiilrakenduste arendamiseks. Ionic baseerib Apache Cordova ja AngularJS tehnoloogitel. Ionic pakkub töövahendeid multiplatvormse rakenduste arendamiseks.

Ionic rakenduste arendamiseks kasutatakse:

- HTML5
- AngularJS

- CSS (Sass)

Kuna Ionic töötab Apache Cordova-ga koos on toetatud sama platvormid:

- iOS
- Android
- Windows Phone
- BlackBerry
- Bada
- Symbian

Ionic CLI vahend võimaldab kasutada konsoolpäringuid kogu arendusprotsessi etappide jaoks. Sealulgas on päringud rakenduste ehitamiseks, kompileerimiseks, testimiseks (nii emulaatoril kui ka füüsilisel seadmel). Ionic live-reload ja integreeritud logeerimine on väga mugavad funktsioonid agiilseks arendamiseks.

Suureks eeliseks on väga detailsed juhendid, mis kirjeldavad iga sammu reaalse rakenduse arendamise eeskujul, niimoodi on lihtne alustada ilma mobilrakenduste aremdamise kogemuseta.

Ionic pakkub detaalse dokumentatsiooni, kus on kirjeldatud kõik nende UI ja JavaScript komponentid, on olemas näited kuidas neid koodis implementeerida.

On olemas ka arendatud kasutajate kogukond, foorumides saab leida abi ja erinevate probleemide lahendusi. [13]

3.2.6 Multiplatvoormse arenduse analüüs

Multiplatvormne arendus on perspektiivne alternatiiv natiivsele arendusele. Ühe koodibaasi arendamine võimaldab teha multiplatvormseid rakendusi, vormistades selleks ainult ühe tiimi kõikide platvormide jaoks arendamiseks. Sellega on nutud tööhulk on minimiseeritud. Väiksem arendajate arv on suur rahaliste kulude vähenemine.

Arendamise protsessi käigus saab kohe kompileerida platvormispetsiifilisi versioone testimiseks. Paralleelne arendus väheneb kogu projektile kuuluvat aega.

Käesoleval peatükkis kirjeldatud multiplatvormse arendamise raamistikud lahendavad ühe probleemi: Android ja iOS platvormide jaoks saab arendada rakenduse üks versiooni, mis baseerib ühel koodibaasil. Sellega „Kegler“ nõued N1 ja N2 on täidetud.

Arvestades rakenduse kiirust reaalsel seadmel Xamarin ja Appcelerator on kõige parimad variantid, sest nad kasutavad platvormispetsiifilisi UI komponente, mis on oma platvormi jaoks optimiseeritud.

Xamarin ja Appcelerator kasutavad platvormispetsiifilisi API-d otseselt, pärast kompileerimist saab natiivse rakendust, teoorias see on natiivse raamistikude arendatud rakendusega sarnane stabiilsuse ja kiiruse põhimõttest.

Nende miinuseks on see et ainult umbes 75% koodi on sama erinevate platvormide jaoks, aga jäänud 25% peavad olema platvormispetsiifilised, ja selle arendamiseks on vaja värbama iga platvormi jaoks kogenud spetsialiste.

Peab arvestama Xamarin litsentsi suure hinnaga.

Sencha ja Apache Cordova baseeruvad ühel printsiibil. Arendatakse HTML5 veebirakendust, mis töötab veebibrauseris, aga on kompileeritud ja pakitud nagu tavaline rakendus, mida saab seadmele installeerida. Veebitehnoloogiate kasutamine teeb arendamisprotsessi võrdlemisi odavamaks, keskmine netokuupalk Eestis veebimeistril on 1053EUR. [16] Arendajatel ei pea olema kogemust mobiilarenduses, sest kasutatakse HTML5, JavaScript ja CSS nii nagu tavalisel veebirakendusel.

Veebitehnoloogiate kasutamise miinus on rakenduse kiirus. Vanadel seadmetel need rakendused töötavad tuntuvalt aeglasem. Uued seadmed aga on piisavalt võimsad selleks et animatsioon töötaks kiiresti ja ilusti.

Natiivse API-de kasutamine on piiratud, neid saab laiendada plugin-te kasutamisel, aga mitte kõik plugin-id töötavad kõike platvormidega. Mõned plugin-id on mitteametlikud, nende stabiilne töö pole garanteeritud.

Ionic Framework baseerib Apache Cordova raamistikul, üldiselt see on sama veebirakenduse printsiib, aga Ionic lisab oma Ionic CLI funktsionaalsust, šabloone, UI komponente ja CSS stiile. Need komponentid on optimiseeritud ja testitud erinevate platvormide jaoks, nende

komponentide kasutamine rakenduse arendamisel võimaldab vältida suurt osa arendamise protsessist. Tänu sellele arendusprotsess on optimiseeritud.

Ionic ja Cordova on tasuta raamistikud, mis on väikese ettevõte jaoks suur eelis.

3.3 Teoreetilise analüüsi kokkuvõtte

Lähtudes kvaliteedi ja funktsionaalsuse põhimõttest kõige parem variant on kasutada ametlikud IDE-d mis on väljatöötatud konkreetse platvormi jaoks. iOS platvormi jaoks see on Xcode ja Android platvormi jaoks Android Studio. Selline lähenemine tähendab, et rakendus on natiivne, ja saab kasutada kõiki platvormi API-d.

Peab arvestama, et tuleb arendada üks rakenduse versioon iga platvormi jaoks. Selleks on vaja kaks tiimi ja iOS tiimis peavad olema kogenud Swift spetsialistid, kelle töökohad on varustatud MacBook arvutitega.

Natiivsel arendamisel iga uue platvormi jaoks on uus projekt, milles on kasutatud uus IDE, programmeerimiskeel ja tehnoloogiad, mis tähendab uue tiimi vormistamist vajaliku platvormi arendamisel kogenud spetsialistitega.

Multiplatvormsed raamistikud lahendavad need probleeme, arendamine muutub võrdlemisi kiiremaks ja odavamaks. Tööturul on palju veebispetsialisti ja nende keskmine töötasu on võrdlemisi madalam. Uue platvormi jaoks arendamine on lihtsam, eriti kui on kasutatud ainult käesoleva platvormiga sobivaid plugin-id.

Apache Cordova ja Sencha veebitehnoloogiatel baseerivad raamistikud aga on funktsionaalsuses ja platvormispetsiifilise API-de suhtes piiratud, funktsionaalselt keerulitse rakenduste arendamiseks need raamistikud ei sobi. Keeruline veebitehnoloogiatel põhinev UI töötab aeglasem kui natiivse UI komponentidest tehtud.

Veebitehnoloogiatel põhinevad raamistikud sobivad ainult baasfunktsionaalusega rakenduste arendamiseks.

Xamarin ja Appcelerator on variant natiivse rakenduste ja vebitehnoloogiatel baseerivatel rakenduste arendamise keskel. Need raamistikud pakkuvad multiplatvormse arendamise eeliseid ja kasutavad natiivseid UI-d ja API-d, mis tähendab et kompileerimisel saab funktsionaalselt natiivsega sarnast rakendust. Aga need raamistikud on tasulised ja nende hind

on võrdlemisi kõrg. Arendajatel peab olema Java või C# kogemus. Umbes 25% koodist on platvormispetsiifiline, mis nõuab igal platvormil kogenud spetsialiste.

Igal mobiilse arenduse raamistikul on oma eelised ja puudused, valida peab lähtudes konkreetse projekti nõudest.

Lõputöö raames on vaja arendada "Kegler" rakenduse iOS ja Android versioone, lähtudes "Kegler" rakenduse nõuetest, rakendusel on ainult baasfunktsionaalsus ja platvormispetsiifilise API-de kasutamine on minimaalne.

"Kegler" rakenduse arendamise juhul on mõistlik minimiseerida arenduse kulud, kuna see on uus produkt turul ja projekti tasuvus on hetkel küsitav.

Arvestades kõike kriteeriumitega, veebitehnoloogiatel põhinev raamistik on eelistatud.

Ionic Framework pakkub mugava CLI, UI komponente ja valmis šablooni multiplatvormse rakenduse arendamiseks. Ionic Framework baseerib "Kegler" projekti jaoks sobival Apache Cordova raamistikul.

Antud projekti raames kasutame Ionic Framework Apple iOS ja Google Android rakenduste versioonide arendamiseks.

4. Raamistikude kasutamise omadused

Antud peatükkis käsitletakse rakenduse arendamiseks sobivaid raamistikke reaalse rakenduse arendamisel. Raamistikude tutvumiseks arendatakse „Hello World!“ rakendust. Arendamise käigus saab näha raamistikude praktilised omadused ja kõrvutama neid teoreetilise analüüsi kokkuvõttega. Raamistikude hindamiskriteeriumid on sama nagu peatükkis „3. Mobiilarenduse raamistikud“, antud peatükki eesmärk on reaalse rakenduse arendamisel raamistiku detaalsem uuring ja teoreetilise kokkuvõtte kinnitamine.

4.1 Android Studio

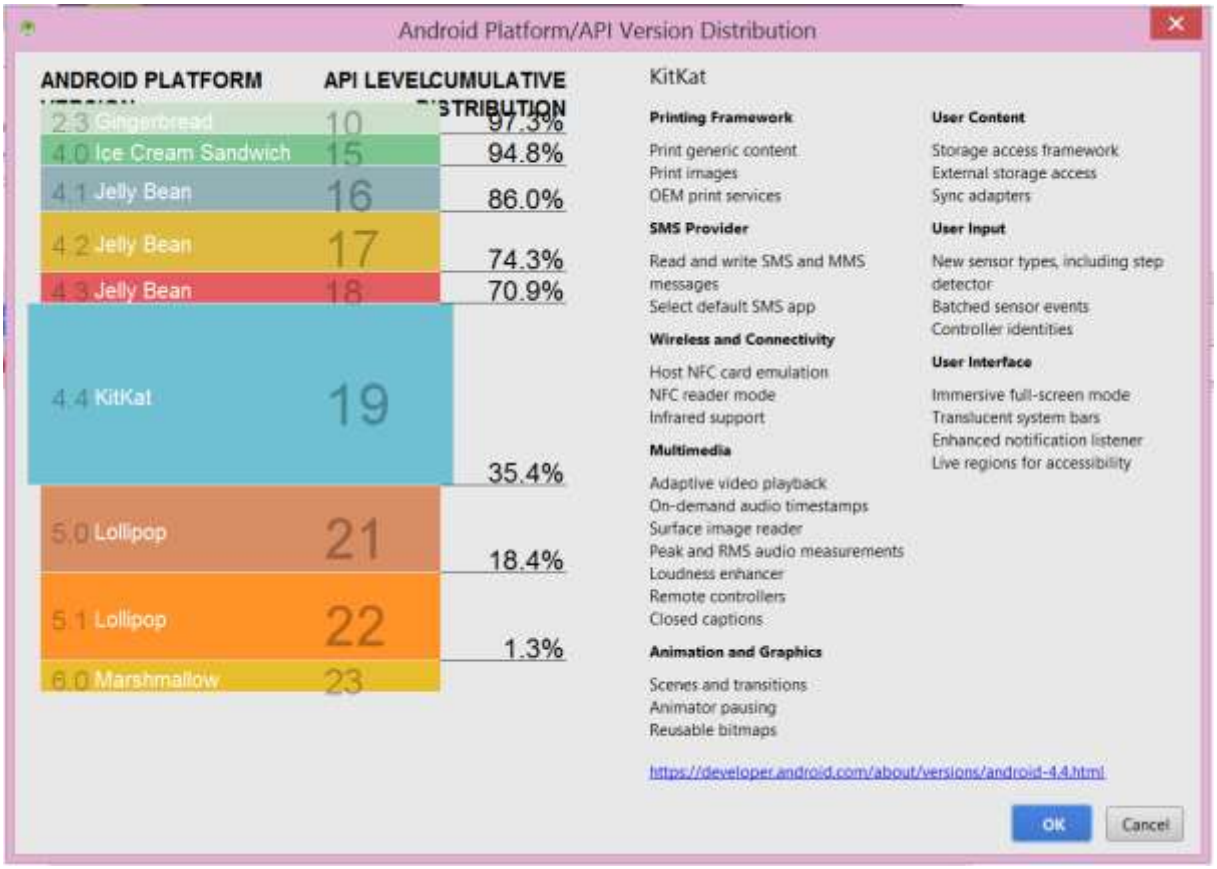
Arendame lihtsa „Hello World!“ rakendust Android platvormi jaoks kasutades Android Studio.

Android Studio saab tasuta allalaadida Android Developer veebilehelt. Lisaks on vaja installeerida Java SE Development Kit (JDK), sobib näiteks Oracle JDK.

Pärast Android Studio installeerimist saab uue projekti alustada.

Uue projekti loomisel on vaja valida rakenduse nimi, firma domeeni, ja kohta, kuhu luuakse uue projekti süsteemis.

Jargmisel leheküljel valitakse „Minimum SDK“, see tähendab millised Android OS versioonid on rakendusega toetatud. Siin saab valida „API 15: Android 4.0“, see tähendab et vana Android versiooniga seadmed on toetatud ja rakendus sobib umbes 97.3% Android seadmetele.



Joonis 3. API versioonidejaotus.

Jargmisena saab valida „activity“, see on rakenduse minimaalne šabloon. Valime “Blank Activity“, seleks et luua tühja rekendust.

Uue projekti struktuur on järgmine:

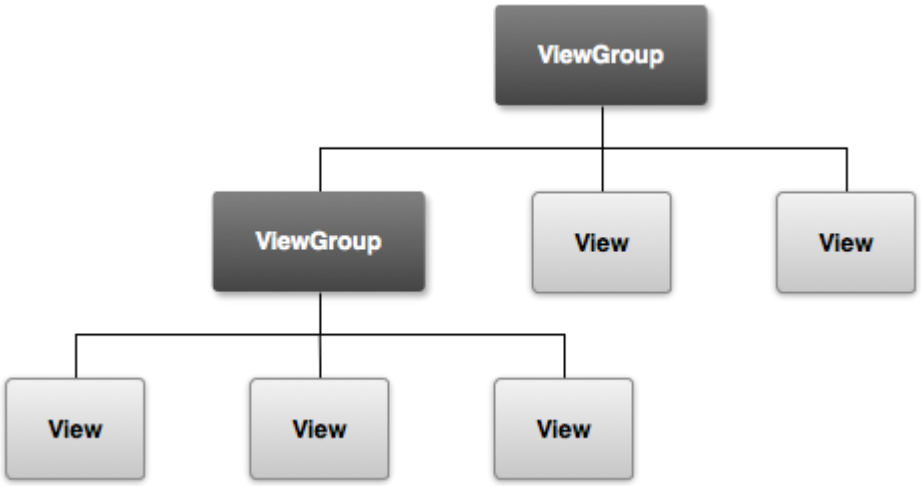
- **.idea:** Android Studio kaust.
- **app:** Rakenduse kaust.
    - **build:** Rakenduse kompileeritut versioon.
    - **libs:** Lisatud libs.
    - **src:**
        - **androidTes:** Rakenduse unit testid.
        - **main:** Peamine kataloog.
            - **java:** Rakenduse kood.
            - **res:** Rakenduse ressursid.
            - **AndroidManifest.xml:** Konfiguratsioon.
    - **.gitignore:** Git ignore konfiguratsioon.
    - **app.iml:** Projekti struktuuri kirjeldus.

        - **build.gradle:** Rakenduse gradle konfiguratsioon.
        - **proguard-rules.pro:** ProGuard.
- **gradle folder:** Gradle.
- **.gitignore:** Peamine git ignore.
- **build.gradle:** Projekti konfiguratsioon.
- **gradle.properties:** Gradle.
- **gradlew:** Teostav fail Unix.
- **gradlew.bat:** Teostav fail (Windows).
- **HelloWorld.iml:** Projekti struktuur.
- **local.properties:** Projekti konfiguratsioon.
- **settings.gradle:** gradle konfiguratsioon.

Jargmisena teeme XML vaadet, millises on „text field“, kus on kirjutatud „Hello World“.

Graafiline UI Android rakenduses on ehitatud kasutades View ja ViewGroup objekte. UI objektid on tavaliselt UI widget-id nagu „button“, „text field“ ja teised.

ViewGroup on nähtamatu konteinerid millised määravad paigutust nende sisse elementidele, näiteks „grid layout“ või „vertical list layout“. Android rakenduses UI paigutust määratatakse kasutades XML.



Joonis 4. ViewGroup skeem.

„Hello World“ kuvamiseks kasutame <LinearLayout>.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/activity_my">
```

Joonis 5. LinearLayout näide.

- „android:orientation=horizontal" - kuvab elemente horisontaalselt.
- „Layout width" ja „layout height" on määratud "match parent", mis tähendab, et vaade on seadme ekraani suurusega.

Jargmisena lisame "text field".

```
<TextField android:id="@+id/message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Joonis 6. Text field.

android:id abil määratakse string-i väärtust.

Lisame „text field"-ga seotud string strings.xml failis.

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="message">Hello Android</string>
</resources>
```

Joonis 7. Text field string.

Selline lähenemine hoiab kõik rakenduse textr ühes kohas, mis võimaldab mugavalt textidega töötada või lokaliseerida rakendust.

Minimaalne "Hello World" Android rakendus on arendatud. Jargmisel etapil tuleb seda ehitama ja käivitama.

Rakenduse ehitamiseks Android Studios on vaja allalaadida Android SDK ehitamise tööristad ja API-d. Selleks käivitame SDK Manager.

Android SDK Manager näitab SDK platvorme, milliseid on vaja uuendada.

Miinimum vajalikud paketid on:

- SDK Tools
- SDK Platform-tools
- SDK Platform

Pärast SDK paketide instaleerimist saab teha rakenduse build, sellega saame rakenduse “apk” faili.

Rakendust saab käivitada android emulaatoril või reaalsel seadmel. Testimiseks on mõlemad variantid vajalikud, sest emulator võimaldav testida rakendust erinevate parameetritega seadmetel, ja reaalsel seadmel testimine näitab kuidas rakendus töötab reaalse kasutaja seadmel, mitte puhtas emulaatori keskonnas.

Alustame Android emulaatorist: selleks seadistame uue AVD (Android Virtual Device). Seal saab valida reaalse Nexus seadme konfiguratsiooni.

Käivitame rakendust seadistatud AVD-s, selleks vajutame Android Studios **Run**  nuppu.

Joonis 8. „HelloWorld“ Android rakendus.

„Hello World“ rakendus töötab korrektselt.

Reaalsel seadmel testimiseks on vaja aktiveerida seadmel „Developer options“ ja aktiveerida Developer menüüs „USB debugging“.

Arvutil peab installeerida OEM USB Driver konkreetse seadme jaoks. Mina kasutan Samsung Galaxy s4. Selle seadme jaoks installeerin Samsung USB Driver.

Kui seade on arvutiga USB kaabliga ühendatud Android Studios saab valida Samsung Galaxy s4 „Choose Device“ menüüs.

Rakendus käivitatakse automaatselt ja Android Studio kuvab kõik seadme log-id.

Reaalsel seadmel “Hello World” rakendus töötab korrektselt. [14]

4.2 Xcode

Arendame „Hello World“ rakendust iOS platvormi jaoks kasutades Xcode.

Xcode on tasuta tarkvara, aga see töötab ainult MacOS Apple arvutiga.

Pärast Xcode installeerimist MacBook arvutil alstame uue „iOS application“ projekti. Tühja šablooni genereerimiseks valime „Single View Application“.

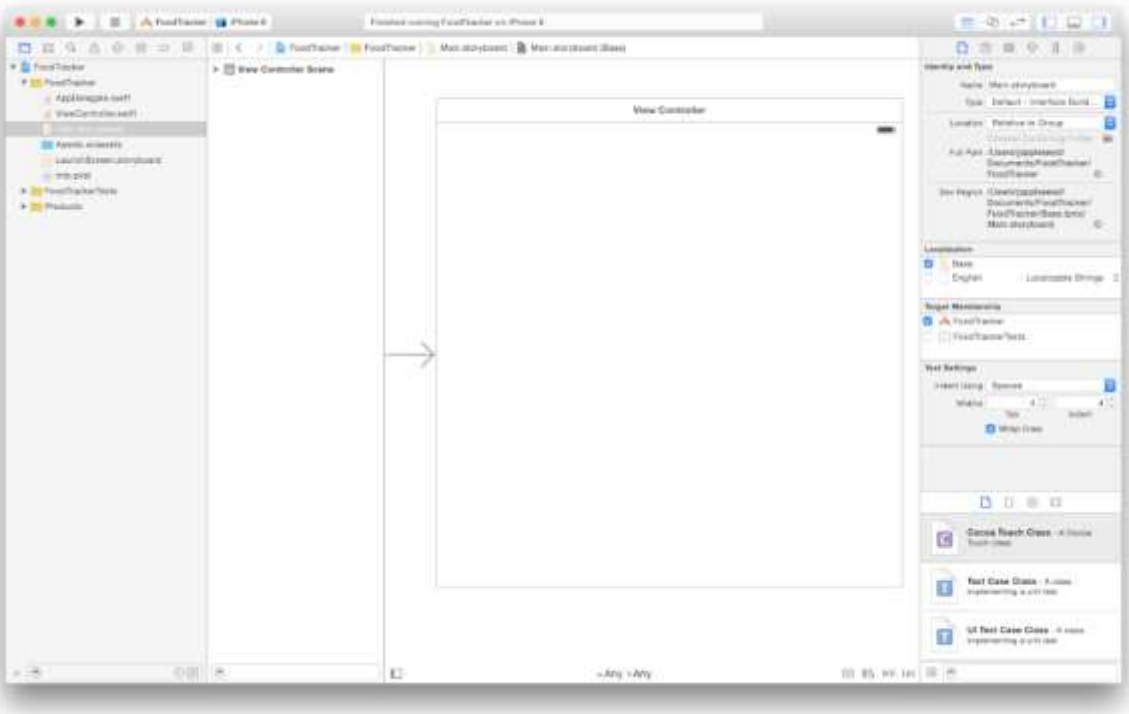
Uue projekti loomisel on vaja valida rakenduse nimi, firma domeeni, ja kohta, kuhu luuakse uue projekti süsteemis.

Xcode genereerib tühja projekti.

„Single View“ genereeritud projektis on järgmine struktuur:

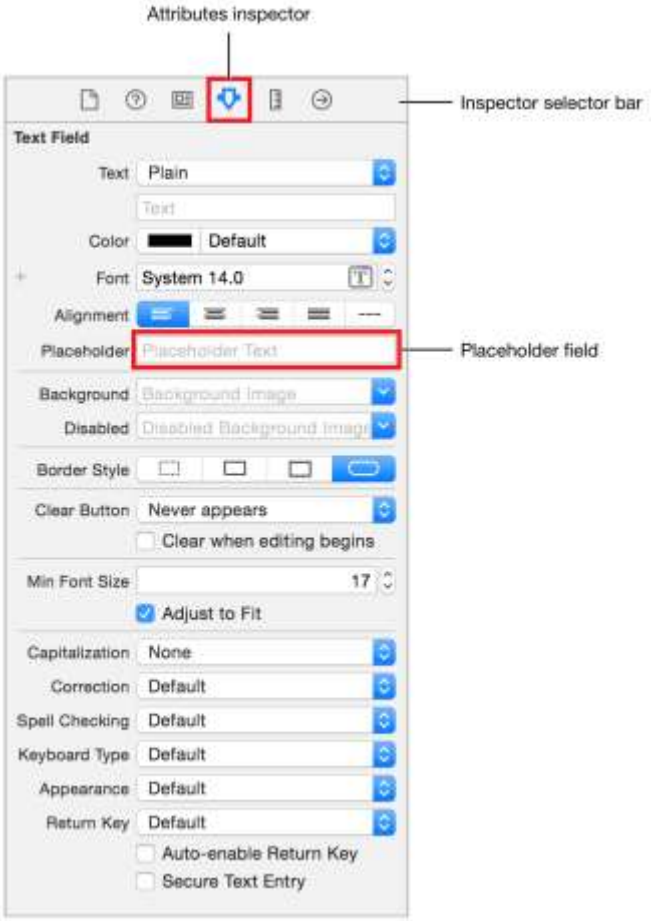
- AppDelegate.swift – peamine rakenduse kood, mida käivitatakse rakenduse avamisel.
- ViewController.swift – konkreetse vaade kood.
- Main.storyboard – rakenduse UI visuaalne representatsioon.

Avame „Main.storyboard“, kus on genereeritud tühi vaade.



Joonis 9. Main storyboard.

Lisame vaatele „text field“ kasutades Xcode Object library, kus on kõik natiivsed UI komponentid. Kasutades „drag-and-drop“ lisame „text field“ peamisele vaatele. Kasutades Xcode Attribute inspector saab seadistada texti.



Joonis 10. Attributes inspector.

ViewController kood jääb muutumata, sest praegu rakendus ei tee midagi vaid kuvab teksti „Hello World“.

```
import UIKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }
}
```

Joonis 11. ViewController.

Nüüd minimaalsed muudatused on tehtud „Hello World“ rakenduse arendamiseks. Käivitame seda iOS Simulaatoris vajutades „Run“ nuppu ja valides „iPhone simulaator“

Joonis 12. iOS "HelloWorld" rakendus.

Reaalsel seadmel testimiseks on vaja registreerida iOS arendajaks tasuliselt. Pärast tuleb registreerida käesoleva iOS seadet testimiseks ja siis saab käivitada rakendust reaalsel seadmel USB kaabli kaudu kasutades Xcode „Run – Run on iOS device“.

„Hello Worls“ rakendus on reaalsel sedmel testitud. [15]

4.3 Native arenduse kokkuvõtte

Tavalise „Hello World“ rakenduse arenamiseks on vaja tutvuda Android Studio ja Xcode raamistikkudega. Igal raamistikul on oma omadused. Natiivse rakenduse arendamise suur probleem on see, et Android Studio ja Xcode on täiesti erinevad raamistikud, ja rakendused on arendatud kasutades erinevad printsiibi ja programeerimiskeeli. Projektide struktuur on täiesti

erinev, vaatamata sellele et „Hello World“ rakendus on väga lihtsa funktsionaalsusega rakendus ja on kasutatud ainult tavaline UI komponent „text field“.

Android-il ja iOS-il on oma raamistikud, millised on kasutatud ainult Android ja iOS rakenduste arendamiseks. Rakenduse Android ja iOS versioonide arendamiseks peab vormistama kaks tiimi, kus töötavad arendajad konkreetse platvormi spetsiifilise kogemusega. „Hello World“ rakenduse praktiline arendamine, kasutades Android Studio Android OS versiooni jaoks ja Xcode iOS versiooni jaoks, kinnitas teoreetilist analüüsi, et arendajatel peab olema platvormispetsiifiline kogemus.

Rakenduse versioonid, tehtud kasutades Android Studio ja Xcode, avanevad seadmel ja töötavad kiiresti, nende UI näeb välja korrektselt.

4.4 Multiplatvormne arendus

Vaatame üle multiplatvormse arenduse võimalust native arenduse alternatiivina. Teoreetilise analüüsi kokkuvõtteks on kontkeetse töö raames kõige sobivaim multiplatvormne raamistik on Ionic Framework. Arendame sama „Hello World“ rakendust Android ja iOS jaoks kasutades Ionic Framework.

4.4.1 Ionic Framework

Arendame „Hello World“ rakendust kasutades Ionic Framework.

Ionic Framework on tasuta raamistik, mida saab installeerida kasutades NPM.

NPM on Node.js osa, seda saab kasutada installeerides Node.js. [16]

Ionic Framework baseerib Apache Cordova CLI komponentidel.

Kasutades Terminal/Command prompt käivitame „npm install“ päringut:

```
$ npm install -g cordova ionic
```

Joonis 13. „npm install“ päring.

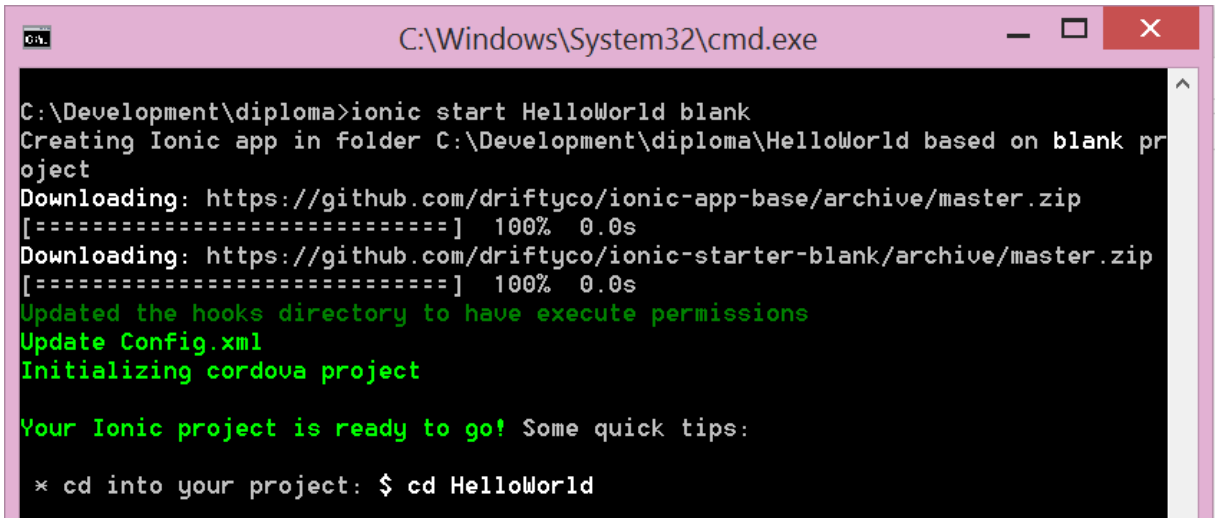
Sellega Ionic Framework on installeeritud ja saab kasutada Ionic CLI multiplatvormse rakenduse arendamiseks.

Alustame uue „Hello World“ projekti:

```
$ ionic start HelloWorld blank
```

Joonis 14. „ionic start“ päring.

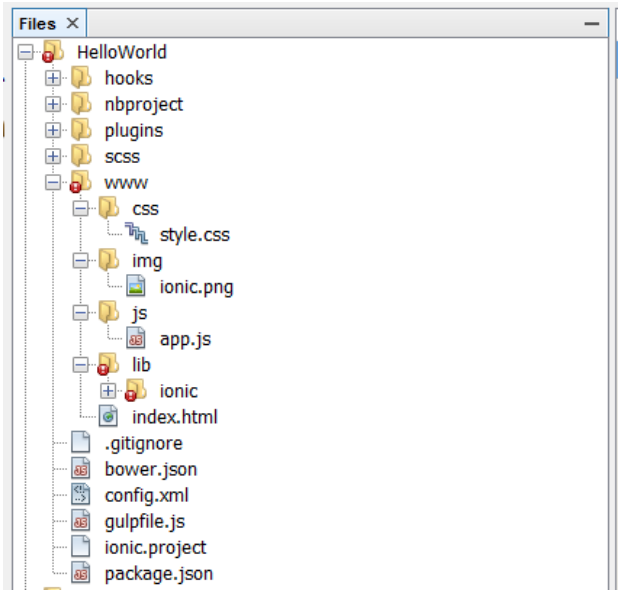
- ionic start – uue projekti alustamine.
- HelloWorld – uue projekti nimi.
- blank – uue projekti tühi šabloon, see võib olla ka „tabs“ või „sidemenu“.



Joonis 15. "blank" projekti loomine.

Projekti saab avada veebiarendamise sobiva IDE-ga, näiteks IntelliJ IDEA, Microsoft Visual Studio, NetBeans.

Uue Ionic Framework projekti struktuur on järgmine:



Joonis 16. Ionic Framewoek prejekt.

```
├── bower.json     // bower dependencies
├── config.xml     // cordova configuration
├── gulpfile.js    // gulp tasks
├── hooks          // custom cordova hooks to execute on specific commands
├── ionic.project  // ionic configuration
├── package.json   // node dependencies
├── platforms      // iOS/Android specific builds will reside here
├── plugins        // where your cordova/ionic plugins will be installed
├── scss           // scss code, which will output to www/css/
└── www            // application - JS code and libs, CSS, images, etc.
```

Joonis 17. Projekti struktuur.

Uurime kausi „www“. Siin asub tavaline veebirakendus:

- css – rakenduse css stiilid.
- img – rakenduse meedia.
- js – rakenduse JavaScript failid. Ionic kasutab AngularJS.
- lib - ionic lib.
- index.html – peamine veebirakenduse HTML5.

Esimesel etapil Ionic arendamine on tavaline veebiarendamine. Nagu tavalisel veebilehel arendamine toimib kasutades HTML5 keelt. Vaatame index.html <body> osa:

```
<body ng-app="starter">
    <ion-pane>
```

```
    <ion-header-bar class="bar-stable">
      <h1 class="title">Ionic Blank Starter</h1>
    </ion-header-bar>
    <ion-content>
    </ion-content>
  </ion-pane>
</body>
```

Joonis 18. „body“ koodiosa.

- ng-app="starter" - on AngularJS binding selleks et seodma JavaScripti HTML5 šablooniga.
- <ion-header-bar> - on Ionic UI component.
- <ion-content> - on Ionic rakenduse vaate osa.

Teeme järgmiseid muudatusi:

```
<h1 class="title">Hello Ionic</h1>
<ion-content>
Hello World!
</ion-content>
```

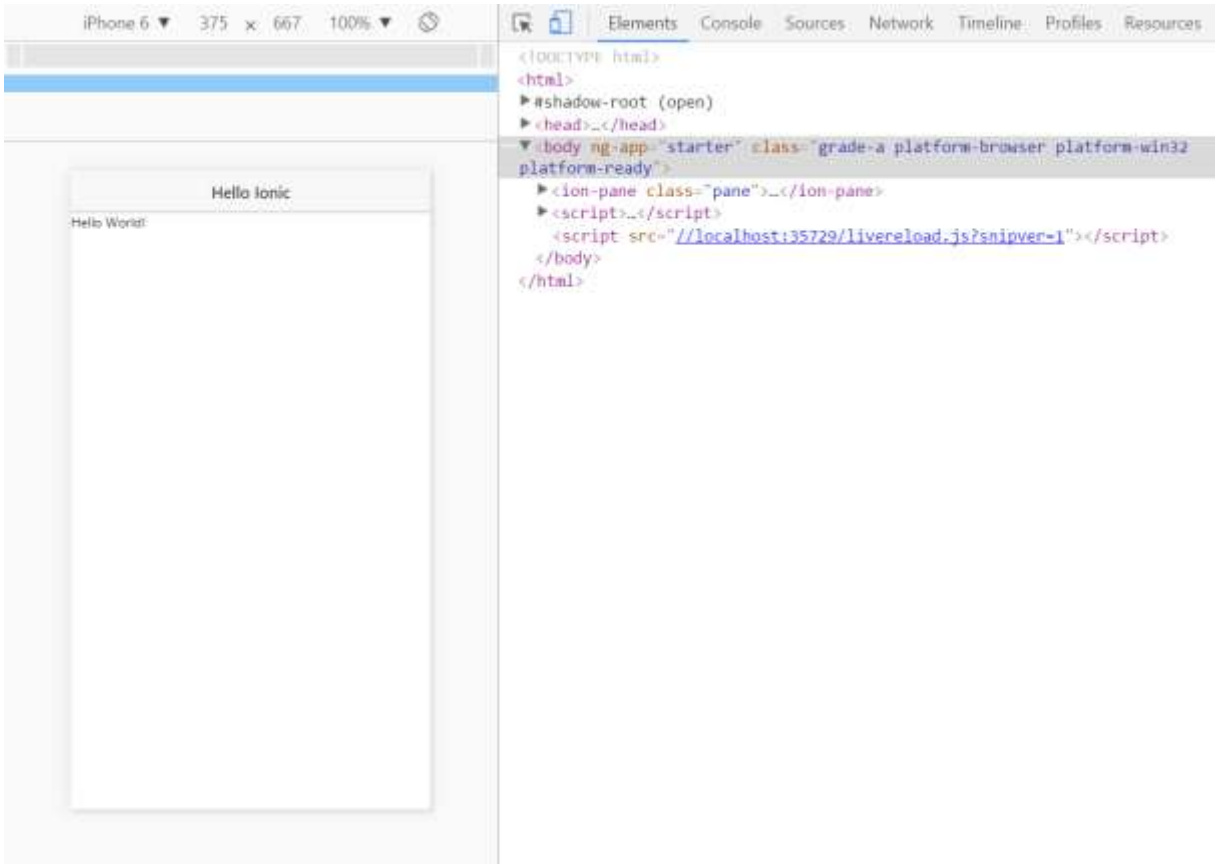
Joonis 19. „Hello World!“ string.

Muudatusi saab testida veebibrauseris kasutades järgmist Ionic CLI päringut:

```
$ ionic serve
```

Joonis 20. „serve“ päring.

Selline päring käivitab live-reload lokaalset serveri. Rakendust saab avada ja testida veebibrauseris. Avame Google Chrome kasutades Developer Options ja Device Mode:



Joonis 21. Live-reload server.

Siin saab valida igasuguseid seadmed erinevate ekraani suurusega testimiseks.

Sellel etapil rakendusel on üks versioon ja ei ole midagi platvormispetsiifilist. Rakenduse arendajaks võib olla ainul veebiarenduse kogemusega arendaja.

Jargmine etap on rakenduse Android ja iOS versioonde arendamine.

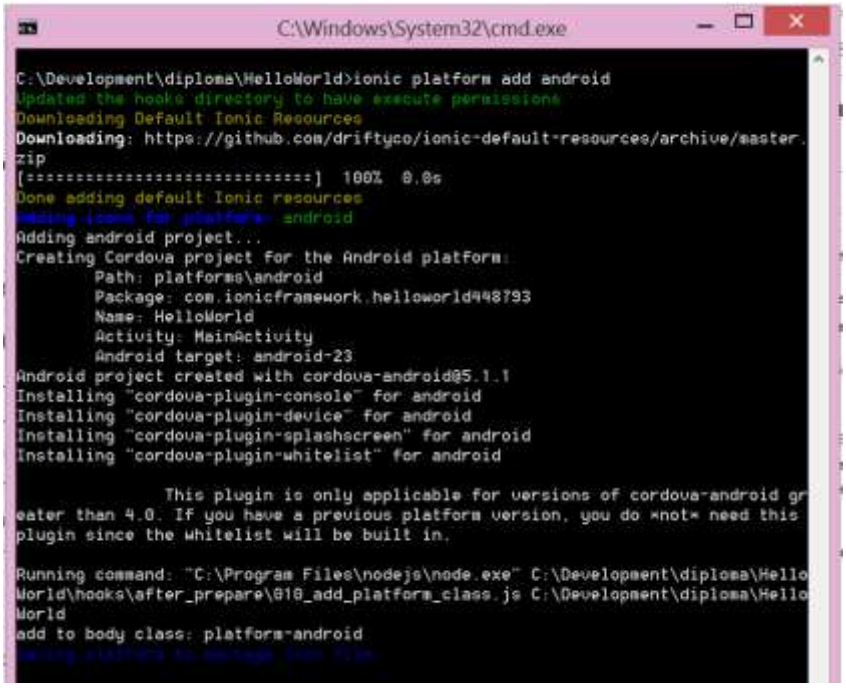
4.4.1.1 Android

Ionic Framework kasutab Android SDK. Rakenduse ehitamiseks ja testimiseks on vaja installeerida Android SDK, uuendada Android platvormi tööriiste ja seadistada AVD emulaatori testimiseks. Need sammud on natiivse Android OS arendamise sarnased ja on kirjeldatud 3.1 „Android Studio“ peatükkis.

Nõudmiste täitmisel saab lisada projektile Android platvormi:

```
$ ionic platform add android
```

Joonis 22. „add android“ päring.



Joonis 23. Ionic CLI add android.

Android platvormi projekti lisades saab käivitada build päringu:

```
$ ionic build android
```

Joonis 24. „build“ päring.

Selline päring ehitab Android rakenduse versiooni kasutades Android SDK, tulemuseks on Android OS seadmel või emulaatoril installeerimiseks valmis apk fail.

Sellega „Hello World“ rakenduse Android OS versioon on valmis.

4.4.1.2 iOS

Ionic Framework kasutab iOS SDK iOS platvormi jaoks rakenduse versiooni ehitamisel. Kuna iOS SDK on kättesaadav ainult Xcode osana, Ionic rakenduse ehitamine iOS jaoks on võimalik ainult Mac OS arvutil.

Kui Xcode on installeeritud saab lisada iOS platvormi Ionic projektile:

```
$ ionic platform add iOS
$ ionic platform prepare iOS
```

Joonis 25. iOS „prepare“ päring.

„prepare iOS“ päring valmistab rakenduse Xcode projekti. Seda projekti peab avama kasutades Xcode. Kasutades Xcode saab kohe rakendust testida iOS seadmel või emulaatoril.

Sellega „Hello World“ rakenduse iOS versioon on valmis. [16]

4.4.2 Multiplatvormse arendamise kokkuvõtte

„Hello World“ rakenduse arendamine Android ja iOS jaoks kasutades Ionic Framework saab jagada kaheks etappideks:

1. **Rakenduse UI ja funktsionaalsuse arendamine** – sellel etapil on kasutatud ainult HTML5, JavaScript ja CSS. Arendajast on nõutud ainult veebitehnoloogia teadmised. Rakenduse arendamine on tavalise veebirakenduse arendamisega sarnane. Platvormispetsiifilised teadmised pole nõutud.

2. **Rakenduse platvormispetsiifiliste versioonide ehitamine** – selline etap on suurel määral valmistatud automaatselt kasutades Ionic CLI käsku. Rakendus 100% baseerib ühel koodibaasil ja platvormi spetsiifilise versiooni ehitamisel pole vaja koodi muuta. Arendajalt on nõutud minimaalne Android SDK ja Xcode arusaamine rakenduse testimiseks seadmel või emulaatoril.

Ionic Framework nõuab Android ja iOS platvormide kogemust minimaalsel määral. On tähis ka see, et rakenduse UI ja funktsionaalsuse arendamine baserib veebitehnoloogiatel. See tähendab et rakendust saavad arendada arendajad, kellel pole mobiilse arendamise kogemust.

5. Analüüsi kokkuvõtte

Arvestades „Kegler“ rakenduse nõuete ja raamistikude kriteeriumitega oli vaja valida kõige optimaalsem raamistik rakenduse Android ja iOS versioonide arendamiseks.

Teoreetilise ja praktilise analüüsi käigus oli põhjalikult uuritud erinevad mobiilarenduse raamistikud.

Apple Xcode ja Google Android Studio raamistikude kasutamine on Google ja Apple soovitatav. See võimaldab kasutada platvormispetsiifilisi tehnoloogiaid, millised on väljatöötatud konkreetse platvormi jaoks ja on selleks optimiseeritud. Natiived rakendused töötavad kiirem kui hübriidsed, ja kasutavad platvormi API-d otseselt, mis võimaldab 100% kasutada platvormi funktsionaalsust ilma piiranguta. Probleem on aga selles, et natiivse arenduse lähenemisel on vaja värbama platvormispetiifilise kogemusega arendajaid. Natiivsel arendamisel iga uue platvormi jaoks on uus projekt, milles on kasutatud oma tehnoloogiad. Xcode kasutamine nõuab Apple arvutiteid. Nende arvutide hind on võrdlemisi kõrg ja „Kegler“ projekti juhul see on suur investeering.

Xamarin ja Appcelerator on mobiilarendamise raamistikude rühm, mis võimaldab arendada ühel koodibaasil põhinevate natiivseid rakendusi. Need raamistikud kasutavad natiivseid UI komponente ja API-d, rakenduse loogika on arendatud kasutades C# ja on multiplatvormne. Nende rakenduste miinuseks on see, et umbes 75% koodist on multiplatvormne, aga jäänud 25% on platvormispetsiifiline. See tähendab et rakenduse arendamiseks tiimis peavad olema nii Android kui ka iOS kogemusega spetsialistid. Need raamistikud on tasulised ja nende hind on võrdlemisi suur. „Kegler“ rakenduse puhul litsentside hind on liiga kallis, sest selle rakenduse tuluvus on arendamise faasis kahenev.

Veebitehnoloogiatel baseerivad raamistikud, nagu Apache Cordova, Ionic Framework, Sencha Touch, sobivad hübriidsete rakenduste arendamiseks. Selline lähenemine on veebitehnoloogiate HTML5, JavaScript, CSS kasutamine multiplatvormsete rakenduste ehitamiseks. Hübriidsete rakenduste peamine eelis on arendamise lihtsus. Suurim osa arendamisprotessist ei nõua mingit platvormispetsifilist teadmisi, mobiilrakenduse arendamine on tavalise veebirakenduse arendamisega sarnane. Veebiarendajate keskmine palk Eestis on tuntuvalt vähem kui Java või C# arendajate oma. Ionic Framework ja Apache Cordova on tasuta raamistikud.

Hübriidsete mobiilrakenduse puudus on nende kiirus ja API-de kasutamise piirangud. „Kegler“ rakenduse puhul nõued on lihtsad, API-de kasutamine on minimaalne. Kiiruse ja piirangute puudused sellel juhul ei mõjuta.

Multiplatvormsed raamistikud lahendavad palju probleeme, arendamine muutub võrdlemisi kiiremaks ja odavamaks. Uue platvormi jaoks arendamine on lihtsam.

Analüüsi tulemuseks on “Kegler” mobiilrakenduse arendamiseks valitud Ionic Framework, lähtudes rakenduse nõuetest ja raamistiku kriteeriumitest, Ionic Framework on kõige sobivaim raamistik. See on tasuta veebitehnoloogiatel põhinev multiplatvormne raamistik. Ionic Framework pakkub CLI arendamise tööriste, lai dokumentatsiooni, oma UI komponente. Analüüsi käigus selgus, et selle raamistiku kasutamine on raha ja ajakulude arvestades optimaalne.

6. Rakenduse arendamine

Antud peatükkis kirjeldatakse „Kegler“ rakenduse arendamine kasutades Ionic Framework-i.

6.1 Ionic projekti loomine

Uue Ionic Framework projekti loomise protsess ja vajalikud programmid on kirjeldatud peatükkis 3.4.1 Ionic Framework. Kegler projekti loomiseks on mõned sammud muudetud.

Alustame Cordova ja Ionic installeerimisest:

Kasutades Terminal/Command prompt käivitame „npm install“ päringu:

```
$ npm install -g cordova ionic
```

Joonis 26. „cordova ionic“ installeerimine.

Sellega Ionic Framework on installeeritud ja saab kasutada Ionic CLI multiplatvormse rakenduse arendamiseks.

Alustame uue „Kegler“ projekti:

```
$ ionic start Kegler sidemenu
```

Joonis 27. Projekti loomine.

- ionic start – uue projekti alustamine.
- Kegler – uue projekti nimi.
- sidemenu – uue projekti šabloon vasakpoolse menüüga.

Loodud projekti saab kohe brauseris käivitada, et näha Ionic šablooni vasakpoolse menüüga.

Šabloonile on lisatud testandmed. Testandmete abil saab testida kuidas menüü töötab ja uurida selle koodi. Vasakpoolne menüü funktsionaalselt vastab Kegler rakenduse nõuetele N3 – vasakpoolse menüüga UI.

6.2 User Interface

Kegler projekt on seadistatud Sass kasutamiseks. Ionic Framework võimaldab teha standartsete CSS stiilide „override“.

Selleks avame styles/main.css faili ja lisame Kegler rakenduse stiile:

```
//Ionic var overrides
$positive:                          $kglr-olympink !default;
$dark:                              $kglr-olympink-dark !default;
$base-color:                        $kglr-text-color !default;
$button-dark-border:                darken($dark, 10%) !default;
$button-dark-active-bg:             $dark;
$button-dark-active-border:         $button-dark-border;
$item-dark-active-bg:               $positive;
$menu-bg:                           $dark;
```

Joonis 28. CSS stiilid.

Sellega on standaartsed Ionic värvid on muudetud ja on kasutatud „Kegler“ rakenduse igal leheküljel automaatselt.

Sellega on „N4: Omapärane „Kegler“ UI stiil“ nõue täidetud.

Muudame vasakpoolse menüü punkte. Menüü HTML5 kood asub templites/menu.html failis.

Teeme järgmised muudatused ühe menüü punkti näitel:

```
<ion-item class="item-dark item-icon-left" nav-clear menu-close
href="#/app/kegelbahn">
<i class="icon kglr-kegelpic"></i>
Kegelbahn
</ion-item>
```

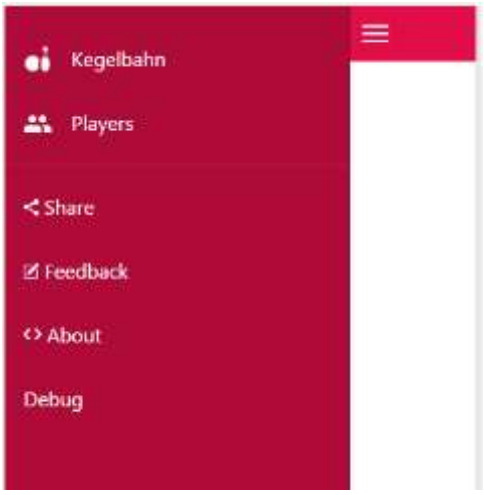
Joonis 29. Menüü punkti näide.

See koodiosa vastab menüü punktile Kegelbahn.

- On muudetud selle „href“ link, mis vastab kegelbahn vaatele: "#/app/kegelbahn".
- On lisatud ikoon.
- Menüü punkti nimi on Kegelbahn.

Kõik teised menüü punktid on samamoodi tehtud.

Rakenduse CSS ja menüü HTML5 koodi muudatuste tulemus on järgmine:



Joonis 30. "Kegler" rakenduse menüü.

Kegler rakenduse pealehel peavad olema erinevad kegel mängude tüübid.

Selleks loome Kegelbahn lehe. Lisame uue HTML5 templates/kegelbahn.html faili.

Iga vaade on seadistatud peamises AngularJS scripts/app.js failis.

Kegelbahn „state“ scripts/app.js failis on järgmine:

```
$stateProvider
                    .state('app.kegelbahn', {
                        url: "/kegelbahn",
                        views: {
                            'menuContent': {
                                templateUrl: "templates/kegelbahn.html",
                                controller: 'KegelbahnCtrl'
                            }
                        }
                    })
```

Joonis 31. stateProvider näide.

- stateProvider – AngularJS „state“ funktsioon.
- „/kegelbahn“ – link navigeerimiseks.
- templateUrl – HTML5 faili asukoht.
- controller – vaate script.

Kegelbahn vaatel peab olema kaks kategooriat:

- Sport

- Fun

Kategooriate kuvamiseks kasutame tabe. Ionic Framework pakkub CSS komponentid UI ehitamiseks.

Kasutame „Tabs“ componenti:

```
<div class="tabs">
  <a class="tab-item">
    SPORT
  </a>
  <a class="tab-item">
    FUN
  </a>
</div>
```

Joonis 32. „Tabs“ koodiosa.

Erinevate mängutüüpide kuvamiseks kasutame „Card Showcase“ UI komponenti:

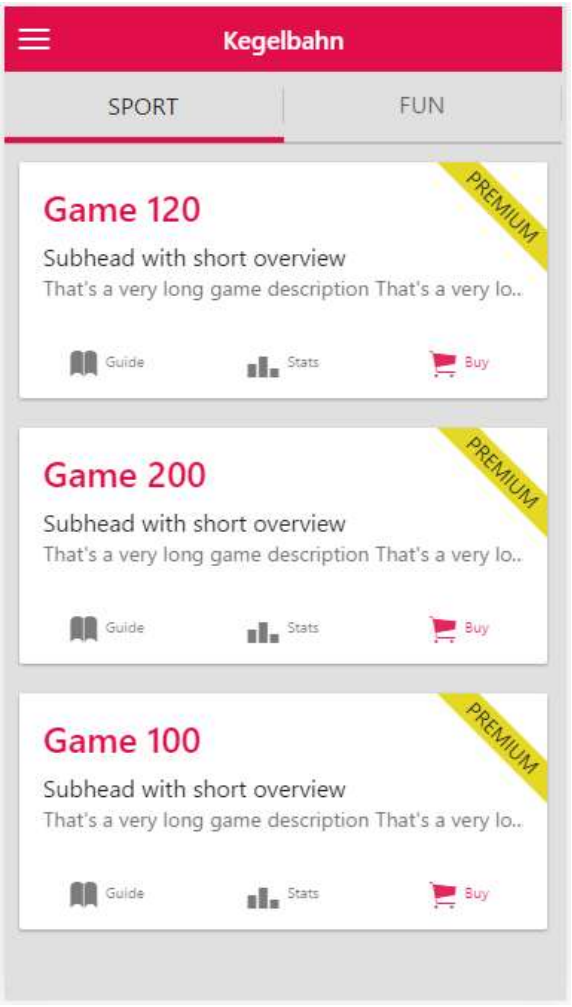
```
<div class="card">
  <div class="item item-divider">
    I'm a Header in a Card!
  </div>
  <div class="item item-text-wrap">
    This is a basic Card with some text.
  </div>
  <div class="item item-divider">
    I'm a Footer in a Card!
  </div>
</div>
```

Joonis 33. „Card Showcase“ koodiosa.

See ei sobi täelikult rakenduse disainile, aga iga komponent on tavaline HTML5 + CSS veebikomponent. HTML5 ja CSS teadmisega saab Ionic komponente muuta ja kohe live-reload serveri kasutamisel brauseris testida.

Mängutüübid on ehitatud sama printsiibiga nagu menüü: igal punktil on oma link, mis võimaldab liiguda järgmisele vaatele.

Tulemuseks on järgmine vaade:



Joonis 34. "Kegler" rakenduse pealeht.

Sama printsiibiga lisame lehed iga menüü punkti jaoks.

Sellega on „Kegler“ rakenduse „N5: Mängude nimekiri pealehel“ nõue on täidetud.

Antud peatükkis on kõik UI-ga seotud nõued on täidetud. [18]

6.3 Funktsionaalsus

Antud peatükkis realiseeritakse „N7: Mängu tulemuste sisestamine“.

Ionic Framework kasutab AngularJS. Rakenduse loogika ja funktsionaalsuse realiseerimiseks kirjutame AngularJS scripte. Igal rakenduse vaatel on oma „Controller“, mis on määratud „stateProvider“-is.

Realiseerime tavalist „Summa“ mängu, sellek loome uue Controller-i scripts/summa.js

```
angular.module('kegler.summa', [])
        .controller('SummaCtrl', function ($scope) {
             $scope.summa = 0;
             $scope.setScore = function ($scope.roundData.currentScore) {
                    $scope.summa = $scope.summa + $scope.roundData.currentScore;
                };
});
```

Joonis 35. Summa controller.

Controller-ile vastav templites/summa.html on järgmine:

```
<ion-view view-title="Round">
    <ion-content class="has-header">
      <div>{{summa}}</div>
      <kglr-scorepad data-ng-model="roundData.currentValue" data-ng-
change="setScore()"></kglr-scorepad>
    </ion-content>
</ion-view>
```

Joonis 36. Summa template.

AngularJS lahendab HTML5 ja Scriptide suhteseose probleemi. „$scope“ kasutamine võimaldab omavahel seotada andmed šabloonis ja koodis. Summa väärtud on peegeldatud šabloonis ja väärtuse muudatused on kohe ekraanil kuvatud.

summa.html šablooni „data-ng-model“ on mängu jooksva väärtuse sisend, „data-ng-change“ on AngularJS funktsioon, mis märkab et väärtus on sisestatud ja kutsub Controller-i setScore()funktsiooni, mis arvutab mängu tulemuse summat.

Sellega on „N7: Mängu tulemuste sisestamine“ nõue täidetud. [19]

6.4 SQLite andmebaas

Kegler rakenduse peamine funktsioon on tulemuste salvestamine ja analüüs (statistika arvestamine). Selleks on vaja salvestada andmeid lokaalselt seadmel. Selline nõue vastab punktile „N6: Lokaalne andmebaas“.

Cordova dokumentatsioon pakkub erinevaid võimalusi andmete salvestamiseks:

- LocalStorage
- WebSQL
- IndexedDB

- Plugin-Based Options

Statistika arvestamiseks on mõistlikum kasutada SQL tehnoloogiat või selle alternatiive.

Kasutame projektis „Cordova-sqlite-storage“ plugin-i, mis implimenteerib SQLite tehnoloogiat rakenduses. [20]

Plugin installeerimiseks kasutatakse Ionic CLI päringu:

```
$ ionic plugin add cordova-plugin-sqlite
```

Joonis 37. sqlite plugin-i lisamise päring.

Pärast installeerimist saab kasutada plugin-it:

```
document.addEventListener('deviceready', onDeviceReady, false);
function onDeviceReady() {
  var db = window.sqlitePlugin.openDatabase({name: 'my.db', location: 'default'});
}
```

Joonis 38. sqlite initsialiseerimine projektis.

„deviceready“ sündmus on kutsutud kui rakendus on avatud seadmel ja ionic-cordova on valmis.

„openDatabase“ funktsioon avab uue SQLite andmebaasi, pärast seda saab kasutada SQL pärunguid.

Loome uue tabeli, kui seda ei ole veel baasis (rakenduse esimesel avamisel).

```
db.transaction(function (tx) {
          tx.executeSql('CREATE TABLE IF NOT EXISTS Result (' +
                  'result_id integer primary key,' +
                  'game_id integer,' +
                  'player_id integer,' +
                  'total integer,' +
                  );
      });
```

Joonis 39. Tabeli loomise kood.

SQLite päringud on täidetud asünhroonses transaktsioonis, mis järgib HTML5/Web SQL API ja kasutab BEGIN/COMMIT/ROLLBACK printsiibi, mis teeb iga SQLite päringu vigadest ohutu.

SQLite võimaldab kasutada palju funktsioone, seal hulgas many-to-many seosed, mis on vajalikud „Kegler“ rakenduses, sest ühel mängul saavad osaleda mõned mängijad ja üks

mängija saab osaleda erinevatel mängudel. Need SQLite funktsioonid on väga kasulikud pikaajalise statistika analüüsimisel.

Andmete salvestamise SQLite päringu näide:

```
db.transaction(function(tx) {
    tx.executeSql("INSERT INTO Games (type, status, settings, finishDate, data)
VALUES (?,?,?,?,?)");
}, function(e) {
      console.log("ERROR: " + e.message);
    });
```

Joonis 40. SQLite salvestamise päring.

Andmete küsitluse SQLite päringu näide:

```
db.transaction(function(tx) {
    tx.executeSql("SELECT p.player_id, p.firstname, p.lastname FROM Games_Players
gp INNER JOIN Players p ON p.player_id = gp.player_id WHERE gp.game_id = ?");
}, function(e) {
      console.log("ERROR: " + e.message);
    });
```

Joonis 41. Sqlite päringu näide.

Kui SQL päringu „executeSql“ täitmisel juhtub viga, siis vastavalt ROLLBACK printsiibile kõik vigase transaktsiooni muudatused on taastatud ja on kutsutud viga ligeerimine.

SQLite plugin-i kasutamine on funktsionaalne ja ohutu tehnoloogia andmete salvestamiseks ja töötlemiseks, mis töötab nii Android kui ka iOS platvormides.

Sellega on „N6: Lokaalne andmebaas“ nõue täidetud.

6.5 Statistika kuvamine

Antud peatükkis on realiseeritud „N8: Mängija statistika kuvamine“ nõue.

„Kegler“ rakenduse peamine funktsioonid on ajalugu ja statistika salvestamine ja kuvamine. Kasutajale on informatiivne näha statistikat graafiku kujul.

Ionic Framework ei paku UI komponente graafiku kuvamiseks.

Selleks saab kasutada erinevaid veebikomponente, Ionic Framework-il on funktsionaalsus erinevate lib komponente lisamiseks. Komponente saab käsitsi lisada „index.html“ failis ja

kasutada JavaScript koodis. Ionic Framework kasutab ka „bower“-i, mis võimaldab lisada komponente automaatselt ja mugavalt neid uuendada.

On olemas erinevad veebitehnoloogiatel HTML5/JavaScript tehnoloogiatel baseerivaid komponentid, mis realiseerivad graafikud.

„Chart.js“ on tasuta OpenSource ja funktsionaalselt „Kegler“ projektile sobiv komponent, millist saab installida kasutades „Bower“. [21]

```
bower install Chart.js –save
```

Joonis 42. „Chart.js“ komponenti instaleerimise päring.

Päringu tulemusena on „Chart.js“ komponent lisatud „bower_components“ kataloogi ja on automaatselt lisatud index.html faili:

Uue komponenti saab kasutada. Chart.js dokumentatsioonis on kõik etapid kirjeldatud ja saab kasutada näidist.

Lisame graafiku „canvas“ templites/history.html faili:

```
<canvas id="myChart" width="400" height="400"></canvas>
```

Joonis 43. „Chart.js“ canvas.

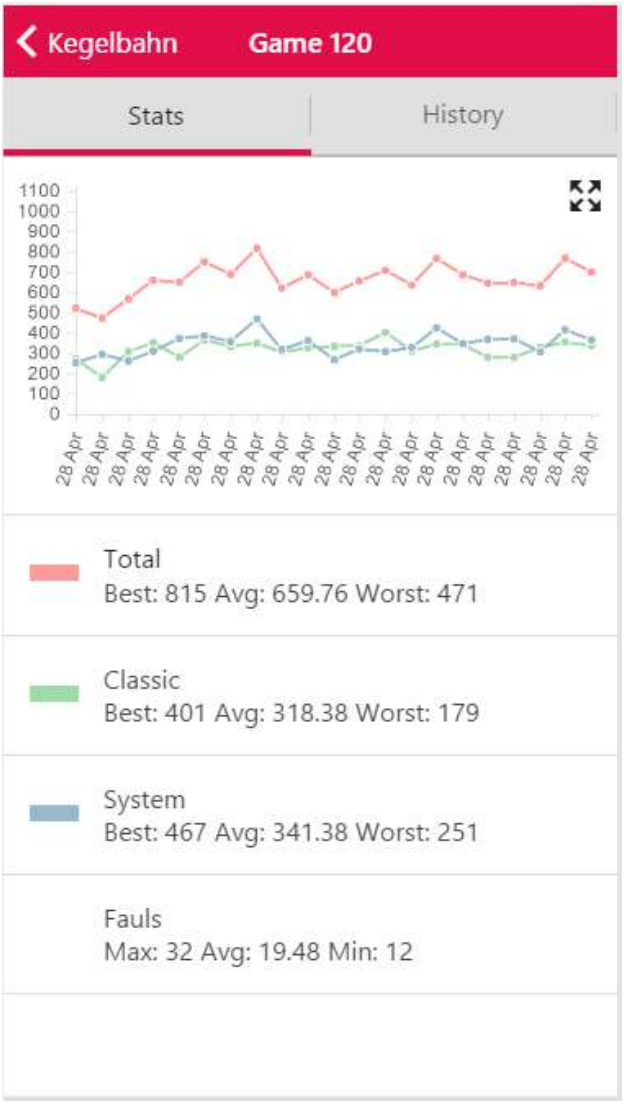
Seotud kontrolleris on SQLite päringuga töödeldut andmete massiiv, mis on lisatud “myChart” objektile. Selle näidis on järgmine:

```
var ctx = document.getElementById("myChart");
var myChart = new Chart(ctx, {
    type: 'bar',
    data: {
        labels: ["Red", "Blue", "Yellow"],
        datasets: [{
            label: '# of Votes',
            data: [12, 19, 3]
        }]
    },
    options: {
        scales: {
            yAxes: [{
                ticks: {
                    beginAtZero:true
                }
            }]
        }
    }
});
```

Joonis 44. „Chart.js“ skript.

“myChart” objekt on muudetud reaalse andmete kuvamiseks.

Tulemuseks on ilus ja funktsionaalne graafik.



Joonis 45. "Kegler" rakenduse statistika.

Sellega on „N8: Mängija statistika kuvamine“ nõue täidetud.

6.6 Android/iOS versioonide loomine

Antud peatükkis on loodud „Kegler“ rakenduse Android ja iOS versioonid, neid saab reaalsetel seadmetel installeerida ja kasutada. Antud peatükk vastab nüuetele „N1: „Kegler“ rakenduse Android versioon“ ja „N2: „Kegler“ rakenduse iOS versioon“.

Ionic rakenduse Android ja iOS versioonide loomine on kirjeldatud peatükkides:

- 3.4.1.1 Android
- 3.4.1.2 iOS

Android ja iOS versioonide loomiseks põle vaja teha platvormispetsiifilisi muudatusi koodis.

Rakendus on testidud järgmistel seadmetel:

- iPhone 5s (iOS 8.2)
- iPhone 6s (iOS 9.3)
- Galaxy S4 (Android 5.1)
- Galaxy S6 (Android 6.0)

Kõik funktsionaalsus töötab. UI on kuvatud õigesti. Lisatud „Chart.js“ komponent on kuvatud igesti erinevate ekraani suurusega seadmetel. Rakendus avaneb ja töötab kiiresti. Testimisel probleemi ei esine.

6.7 „Kegler“ rakenduse arendamise kokkuvõte

On arendatud „Kegler“ rakendus kasutades Ionic Framework. Kõik nõued on täidetud:

- N1: „Kegler“ rakenduse Android versioon.
- N2: „Kegler“ rakenduse iOS versioon.
- N3: Vasakpoolse menüüga UI.
- N4: Omapärane „Kegler“ UI stiil.
- N5: Mängude nimekiri pealehel.
- N6: Lokaalne andmebaas.
- N7: Mängu tulemuste sisestamine.
- N8: Mängija statistika kuvamine.

Arendamise protsess oli natiivse arendamise võrreldes lihtne, nõutud platvormispetsifiline kogemus oli minimaalne. Ionic Framework kasutamisel kõik tarkvara oli tasuta.

Ionic Framework sobis täismääral “Kegler” rakenduse arendamiseks.

7. Kokkuvõte

Antud töö põhieesmärk on „Kegler“ mobiilrakenduse arendamine Google Android ja Apple iOS platvormide jaoks, arvestades mõistlikkude ajalise ja rahalistekuludega.

Arvestades „Kegler“ rakenduse nõuetega oli vaja valida kõige optimaalsem raamistik ja arendada rakendust.

Teoreetilise ja praktilise analüüsi käigus oli põhjalikult uuritud erinevad mobiilarenduse raamistikud. Need raamistikud lahendavad probleemi erinevate lähenemisega, ja ilma projekti nõuete arvestamata ei saa teha järeldust milline raamistik on kõige parim.

Apple Xcode ja Google Android Studio raamistikude kasutamine on Google ja Apple soovitatav. See võimaldab kasutada platvormispetsiifilisi tehnoloogiaid, millised on väljatöötatud konkreetse platvormi jaoks ja on selleks optimiseeritud. Natiived rakendused töötavad kiirem kui hübriidsed, ja kasutavad platvormi API-d otseselt, mis võimaldab 100% kasutada platvormi funktsionaalsust ilma piiranguta. Probleem on aga selles, et natiivse arenduse lähenemisel on vaja värbama platvormispetiifilise kogemusega arendajaid. Natiivsel arendamisel iga uue platvormi jaoks on uus projekt, milles on kasutatud oma tehnoloogiad.

Xamarin ja Appcelerator on mobiilarendamise raamistikude rühm, mis võimaldab arendada ühel koodibaasil põhinevaid natiivseid rakendusi. Need raamistikud kasutavad natiivsed UI komponente ja API-d, rakenduse loogika on arendatud kasutades C# ja on multiplatvormne. Nende rakenduste miinuseks on see, et umbes 75% koodist on multiplatvormne, aga jäänud 25% on platvormispetsiifiline. See tähendab et rakenduse arendamiseks tiimis peavad olema nii Android kui ka iOS kogemusega spetsialistid. Need raamistikud on tasulised ja nende hind on võrdlemisi suur. „Kegler“ rakenduse puhul litsentside hind on liiga kallis, sest selle rakenduse tuluvus on arendamise faasis kahenev.

Veebitehnoloogiatel baseerivad raamistikud, nagu Apache Cordova, Ionic Framework, Sencha Touch, sobivad hübriidsete rakenduste arendamiseks. Selline lähenemine on veebitehnoloogiate HTML5, JavaScript, CSS kasutamine multiplatvormsete rakenduste ehitamiseks. Hübriidsete rakenduste peamine eelis on arendamise lihtsus. Suurim osa arendamisprotessist ei nõua mingit platvormispetsifilist teadmisi, mobiilrakenduse arendamine on tavalise veebirakenduse arendamisega sarnane. Veebiarendajate keskmine palk Eestis on

tuntuvalt vähem kui Java või C# arendajate oma. Ionic Framework ja Apache Cordova on tasuta raamistikud.

Hübriidsete mobiilrakenduse puudus on nende kiirus ja API-de kasutamise piirangud. „Kegler“ rakenduse puhul nõued on lihtsad, API-de kasutamine on minimaalne. Kiiruse ja piirangute puudused sellel juhul ei mõjuta.

Multiplatvormsed raamistikud lahendavad palju probleeme, arendamine muutub võrdlemisi kiiremaks ja odavamaks. Uue platvormi jaoks arendamine on lihtsam.

Analüüsi tulemuseks on “Kegler” mobiilrakenduse arendamiseks valitud Ionic Framework, analüüsi käigus selgus, et selle raamistiku kasutamine on raha ja ajakulude arvestades optimaalne.

“Kegler” rakenduse arendamisprotsess tõendas, et Ionic Framework sobib “Kegler” rakenduse nõuete täitmiseks. Arendatud “Kegler” rakendus töötab Android jai OS seadmetel, nõued on täidetud mõlema platvormide jaoks. Arendamise protsess oli natiivse arendamise võrreldes lihtne, nõutud platvormispetsifiline kogemus oli minimaalne. Ionic Framework kasutamisel kõik tarkvara oli tasuta, rahakulusi litsenside ostmiseks ei olnud.

Ionic Framework sobis täismääral “Kegler” rakenduse arendamiseks, raha ja ajakulude arvestades Ionic Frameworki kasutamine on optimaalne variant.

Sellega võib jareldust teha, et lõputöö eesmärgid on saavutatud täismääral.

“Kegler” projekti võimalik edasiarendus on Windows Platvormi omandamine. Ionic Framework võimaldab luua rakenduse Windows Mobile versiooni sama koodibaasil. Natiivse arendamise puhul oleks vaja arendada “Kegler” rakendust uuesti kasutades Windows Visual Studo.

Edasiarenduseks Ionic Framework on sobiv ning optimaalne raamistik.

Summary

The aim of this work is to develop „Kegler“ mobile application for both Google Android and Apple iOS mobile platforms in a limited periood of time and resources invested.

There were comprehensively examined mobile development frameworks in the process of theoretical and practice analysis. These frameworks solve problems with the different approaches and without taking project demands into consideration it is impossible to decide which one is better one.

Development using native framework is recommended by the vendor, whereas Google suggests the Google Android Studio and Apple suggests Apple Xcode. It makes it possible to use special technologies, which were developed for particular framework. Native apps work faster than hybrid and use platform API directly, what makes it possible to them without any limitations. However, the problem is that in approach to native development there is a need to employ workers with development experience for specific platform.

Xamarin and Appcelerator are the group of mobile development frameworks, which allows to develop the native apps, that are drawn on the same code base. These frameworks use the native UI components and API-s. The logic of the app is developed using C# and it is cross-platform. The disadvantage of the app is that only about 75% are cross-platform and the rest 25% are platform-specific. It means that specialists with an experience in Android, as well as specialists with an experience in iOS are needed. These frameworks are priced, in the limited financial conditions („Kegler“ project case) license price could be critical investment amount.

The frameworks, that are based on web-technologies, such as Apache Cordova, Ionic Framework, Sencha Touch, are suitable for mobile application development. Those applications are developed using web-programming languages: HTML5, JavaScript, CSS. The advantage of this approach is it`s simplicity. The biggest part of a development process can be completed without any platform-specific skills. Hybrid application development is equal to simple web-application development. Web-developer`s average salary in Estonia is notably lower than Java or C# developer`s one. Ionic Framework and Apache Cordova are free for commercial use.

The main disadvantages of hybrid applications are lower speed and API usage limitations. In case of “Kegler” project, all requirements can be completed using Ionic Framework, those limitations doesn’t affect the final decision whish framework to use.

Cross-platform frameworks solve a lot of problems and optimize a development process, this reduses project price and time needed.

The result of analysis the decision to use Ionic Framework to develop “Kegler” mobile application for Android and iOS platforms. This framework is optimal taking project requirements into consideration.

The “Kegler” application development using Ionic Framework approved theoretical analysis decision. Compared to native, web-technology based development is notably easier, platform specific skills needs is minimal and all used software for this project is free.

The aims of this work are achieved, the most suitable mobile development framework had been selected and the “Kegler” application was developed.

Kasutatud kirjandus

[1] „Gartner Says Worldwide Smartphone Sales Grew 9.7 Percent in Fourth Quarter of 2015,“ [Võrgumaterjal]. Available: http://www.gartner.com/newsroom/id/3215217.

[2] „NORTH AMERICAN BOWLING: The Game of Kegel,“ [Võrgumaterjal]. Available: http://www.northamericanbowling.com/Articles/8-KEGEL1.HTML.

[3] „kegel - Android Apps on Google Play,“ [Võrgumaterjal]. Available: https://play.google.com/store/search?q=kegel&c=apps.

[4] „Xcode - What's New - Apple Developer,“ [Võrgumaterjal]. Available: https://developer.apple.com/xcode/.

[5] „Swift - Overview - Apple Developer,“ [Võrgumaterjal]. Available: https://developer.apple.com/swift/.

[6] „Xcode on the Mac App Store,“ [Võrgumaterjal]. Available: https://itunes.apple.com/us/app/xcode/id497799835?mt=12.

[7] „Android Studio Overview | Android Developers,“ [Võrgumaterjal]. Available: http://developer.android.com/tools/studio/index.html.

[8] „E-pood » MacBook Pro,“ [Võrgumaterjal]. Available: http://imarvutid.ee/et/shop/kat/macbookpro.

[9] „Palk - Objektorienteeritud C-programmeerija - Palgad.ee,“ [Võrgumaterjal]. Available: http://www.palgad.ee/salaryinfo/infotehnoloogia-it/objektorienteeritud-c-programmeerija.

[10] „Palk - Java-programmeerija - Palgad.ee,“ [Võrgumaterjal]. Available: http://www.palgad.ee/salaryinfo/infotehnoloogia-it/java-programmeerija.

[11] „Mobile Application Development to Build Apps in C# - Xamarin,“ [Võrgumaterjal]. Available: https://www.xamarin.com/platform.

[12] „Apache Cordova,“ [Võrgumaterjal]. Available: https://cordova.apache.org/#getstarted.

[13] „Mobile App Development & MBaaS Products | Appcelerator,“ [Võrgumaterjal]. Available: http://www.appcelerator.com/mobile-app-development-products/.

[14] „Cross-platform Mobile Web App Development Framework for HTML5 and JS | Sencha,“ [Võrgumaterjal]. Available: https://www.sencha.com/products/touch/#overview.

[15] „Ionic: Advanced HTML5 Hybrid Mobile App Framework,“ [Võrgumaterjal]. Available: http://ionicframework.com/.

[16] „Palk - Veebimeister / Webmaster - Palgad.ee,“ [Võrgumaterjal]. Available: http://www.palgad.ee/salaryinfo/infotehnoloogia-it/veebimeister-webmaster.

[17] „Building Your First App | Android Developers,“ [Võrgumaterjal]. Available: http://developer.android.com/training/basics/firstapp/index.html.

[18] „Start Developing iOS Apps (Swift): Jump Right In,“ [Võrgumaterjal]. Available: https://developer.apple.com/library/ios/referencelibrary/GettingStarted/DevelopiOSAppsSwift/index.html?utm_source=statuscode&utm_medium=email.

[19] „About | Node.js,“ [Võrgumaterjal]. Available: https://nodejs.org/en/about/.

[20] „iOS Platform Guide - Apache Cordova,“ [Võrgumaterjal]. Available: https://cordova.apache.org/docs/en/latest/guide/platforms/ios/index.html.

[21] „What's Included - Ionic Components,“ [Võrgumaterjal]. Available: http://ionicframework.com/docs/overview/#whats-included.

[22] „AngularJS: Tutorial: Tutorial,“ [Võrgumaterjal]. Available: https://docs.angularjs.org/tutorial.

[23] „GitHub - litehelpers/Cordova-sqlite-storage: A Cordova/PhoneGap plugin to open and use sqlite databases on Android & iOS with HTML5/Web SQL API (Windows version available),“ [Võrgumaterjal]. Available: https://github.com/litehelpers/Cordova-sqlite-storage.

[24] „Chart.js | Open source HTML5 Charts for your website,“ [Võrgumaterjal]. Available: http://www.chartjs.org/.