LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

MOBILAS LIETOTNES IZSTRĀDES TEORIJA UN PRAKSE, IZMANTOJOT ANDROID UN IOS IZSTRĀDES VIDES

BAKALAURA DARBS

Autors: Ilmārs Svilsts
Studenta apliecības Nr.: is13135
Darba vadītājs asoc.prof., Dr.dat. Uldis Straujums

RĪGA 2017

Anotācija

Bakalaura darba autors ir izvēlējies tēmu “Mobilas lietotnes izstrādes teorija un prakse, izmantojot Android un iOS izstrādes vides”.

Atsaucoties uz to, ka jauni izstrādātāji neapzinās *Android* un *iOS* izstrādi, tādējādi izvēloties izstrādāt lietotni kādā no operētājsistēmām, ko nepārzin. Izstrādātājam veicot nepareizu izvēli, tiek ietekmēts izstrādes laiks, tā līdzekļi un gala produkta peļņa.

Bakalaura darba mērķis ir iepazīties ar pašreizējām *Android* un *iOS* izstrādes vidēm, programmēšanas valodām un tirgu, tās apskatīt un salīdzināt.

Šī darba ietvaros darba autors sniedz ieskatu abu operētājsistēmu izstrādes vidēs, kā arī valodās un tirgū. Visi aspekti tiek salīdzināti savā starpā un gala rezultātā tiek izdarīti secinājumi.
Atslēgvārdi: Android, iOS, izstrādes vide.

Abstract

Bachhelor work author has chosen the theme of „Mobile application development, using Android and iOS systems – theory and praxis”.

Referring to the fact that new developers are not aware of Android and iOS development, thus choosing to develop an app in one of the operating systems, that they are not familiar with.

By making the wrong choice , development time, resources and total profit is affected.

The objective of bachelor’s work is to get acquainted with the current Android and iOS development environments, programming languages and the market, take insight and compare them.

In this work the author provides an insight into both operating system development environments, as well as languages and market. All aspects are compared and conclusions are made.

Keywords: Android, iOS, development environment

Saturs

Apzīmējumi

Android Inc.- uzņēmums, kurš sākotnēji izstrādāja Android operētājsistēmu.

Android OS – mobilo ierīču operētājsistēma, kas uz šo brīdi pieder „Google" kompānijai.

API(Application programming interface) – lietojumprogrammas saskarne, kas ir iepriekš definētu klašu, procedūru, funkciju, struktūru un konstanšu kopums, kas tiek pasniegts kā pielikums, kuru iespējams izmantot ārējiem programmatūras produktiem.

Apk(Android Package Kit) – faila formāts, kuru izmanto *Android* operētājsistēma lietotņu izplatīšanai un instalācijai.

Apple Inc. – ASV korporācija, kas izstrādā un pārdod elektroniku un programmatūru, kā arī tā ir *iOS* operētājsistēmas izstrādātāja.

Css(Cascading Style Sheets) – stila lapas valoda, kas ir paredzēta, lai aprakstītu izskatu iezīmēšanas valodā veidotiem dokumentiem.

Emulators – datortehnika vai programmatūra, kura ļauj vienai sistēmai izlikties par kādu citu sistēmu.

Google – amerikāņu multinacionāla kompānija, kas specializējas uz interneta saistītiem servisiem un produktiem.

Google Play – elektronisks lietotņu veikals, kas ir paredzēts lietotņu lejuplādei *Android* operētājsistēmas ierīcēm.

Html(HyperText Markup Language) – iezīmēšanas valoda, kas ir paredzēta pārlūkprogrammā attēlojamas informācijas glabāšanai.

iOS OS – mobilo lietotņu operētājsistēma, kuru izstrādāja „Apple".

Javascript – skripta veida valoda, kura ir paredzēta tīmekļa lapu vai lietotņu funkcionalitātes veidošanai.

Linux – bezmaksas operētājsistēma, kura modificēta versija tiek izmantota uz *Android* ierīcēm.

Maven – automatizācijas rīks, kas ir paredzēts primāri „Java" projektiem.

MVC(Model-view-controller) – programmatūras arhitektūras modelis, kurš sadala lietotni trīs daļās, kur modelis atdala lietotnes datus no biznesa loģikas un no vizuālā attēlojuma.

OS (Operētājsistēma) – programmu komplekss, kas vada datu organizēšanu un programmu izpildi datorā, nodrošina aparatūras un programmatūras kopdarbību, resursu racionālu izmantošanu, kā arī sadarbību ar lietotāju.

ProGuard – komandrindas rīks, kas atļauj samazināt un optimizēt “Java” izstrādātu programmas kodu.

QEMU(Quick Emulator) – resursu datorprogramma, kas izveido un palaiž virtuālu ierīci.

Samsung – starptautisks elektronikas uzņēmums, kurš ir populārākais *Android* mobilo ierīču izplatītājs pasaulē.

SDK(Software development kit) – izstrādātāju lietotņu rīku kopa, kas ļauj izstrādāt lietotnes priekš noteiktas operētājsistēmas. Parasti satur visu nepieciešamo lietotņu izstrādei un to atvieglošanai.

Simulators – ierīce vai programmatūra, kura mākslīgi imitē kādu citu ierīci.

Unity – spēļu dzinējs, kurš primāri ir izstrādāts spēļu veidošanai.

Ievads

Latvijā un pasaulē aiz vien populārākas kļūst mobilās ierīces, kas aizvieto daudzas citas ierīces un atvieglo cilvēku darbu un ikdienu. Gandrīz katram cilvēkam uz ielas ir iespējams rokās redzēt mobilo ierīci, pat atpūšoties vai, piemēram, vasarā sauļojoties, cilvēki izmanto mobilās ierīces un to iespējas. Aizvien vairāk sabiedrībā var novērot to, ka tā paliek atkarīga no mobilajām ierīcēm.

Attīstoties tehnoloģijām, pēdējos 10 gados ļoti populāras ir kļuvušas mobilās ierīces tieši ar *Android* un *iOS* operētājsistēmām, kas ir pārņēmušas gandrīz visu mobilo ierīču tirgu. Cilvēki bieži vien nevar izlemt par to, kādu ierīci viņi vēlas, kas izstrādātājiem ir vēl sarežģītāk, jo bieži vien projekti saņem ierobežotus līdzekļus natīvo lietotņu izstrādei un viņiem ir jāizlemj, uz kuras operētājsistēmas tiks veidota attiecīgā lietotne. Pēdējo gadu laikā izplatīta ir kļuvusi tīmekļa lietotņu veidošana, kur iespējams izveidot lietotni uzreiz uz visām operētājsistēmām, bet šī pieeja joprojām nespēj aizstāt natīvo izstrādi.

Bakalaura darba mērķis ir salīdzināt *Android* un *iOS* natīvo mobilo lietotņu izstrādi, veikt ieskatu svarīgākajos tās aspektos, lai uzzinātu, kura operētājsistēma ir labāka tās lietotņu izstrādē.

Šajā darbā tiek apskatītas divas populārākās mobilo ierīču operētājsistēmas, to natīvās izstrādes vides, izstrādes valodas, kā arī tiek veikts ieskats mobilajā tirgū. Šie faktori tiek salīdzināti savā starpā un veikti kopsavilkumi un secinājumi.

1. Vēsture

Lai saprastu, par ko tiks rakstīts šajā darbā, ir svarīgi veikt ieskatu mobilo ierīču operētājsistēmu vēsturē. Liela daļa izstrādātāju izvēlas operētājsistēmu, kas ir tikusi izveidota tuvāk mūsdienām. Šī iemesla dēļ tiek veikts ieskats šajā aspektā.

1.1 Android

Android ir mobilo ierīču operētājsistēma, kas tika izstrādāta kompānijā "Android Inc.", kuru 2005. gadā pārpirka "Google" [25]. 2007. gadā *Android* operētājsistēma tika atklāta plašākai sabiedrībai un pirmā komerciālā *Android* ierīce tika izlaista 2008. gada septembrī, kas bija "HTC" mobilā ierīce ar nosaukumu "HTC Dream", pazīstama arī kā "T-Mobile G1" vai "Era G1".

Šī operētājsistēma ir bāzēta uz "Linux" kodola operētājsistēmas, un, tā pat, kā "Linux" operētājsistēma ir atvērtā pirmkoda projekts, kas nozīmē, ka operētājsistēmu var iegūt un modificēt jebkurš, kuram ir tāda vēlme. Tāpēc šobrīd uz vairākām mobilām ierīcēm viena un tā pati operētājsistēmas versija, iespējams, izskatīsies savādāk.

Android lietotnes tiek ielādētas no "Google Play" veikala, kas ir viens no lielākajiem mobilo lietotņu izplatītājiem pasaulē. 2017. gadā veikalā ir pieejamas apmēram 2,7 miljoni lietotnes, kur tikai 13% sastāda zemas kvalitātes produkti [8]. Šo operētājsistēmu izmanto vairāk nekā 1.4 miljardi lietotāju [9].

Android lielākais pluss ir to mobilo ierīču daudzveidība. Pircēju klāsts ir milzīgs. *Android* ierīces tiek pārdotas caur vairākām kompānijām, kur pazīstamākās ir "Samsung" un "HTC" kompānijas. 2016. gada ceturksnī kompānijām "Samsung" un "Apple" bija vienāds pārdotais ierīču skaits pasaulē, kas bija apmēram 77000 vienības [16], pēc tā var secināt, ka *Android* ierīces ir daudz populārākas tirgū nekā *iOS*, jo *Android* ierīces tiek pārdotas caur vairākām kompānijām.

1.2 iOS

Kompānija "Apple Inc." izstrādāja "iPhone OS", kura plašākā sabiedrībā ir pazīstama kā *iOS* operētājsistēma. Atšķirībā no *Android* operētājsistēmas šī operētājsistēma tika veidota ekskluzīvi "Apple" kompānijas produktiem. *iOS* operētājsistēma ir otrā populārākā

operētājsistēma pasaulē, kur *Android* operētājsistēma ieņem pirmo vietu. Pirmo reizi *iOS* ar sabiedrību tika iepazīstināts 2007. gada 9. janvārī. Tas nozīmē, ka *iOS* tirgū iekļuva neilgu laiku pirms *Android*, kas deva savus plusus un mīnusus. Par plusu var uzskatīt to, ka "Apple" kompānijai bija vairāk laika izvērsties tirgū un līdz ar to iegūt lietotāju uzticēšanos. Taču *Android* varēja novērtēt ierīču cenas un to mīnusus, līdz ar to nepieļaujot ko tādu savā produkcijā. Tādējādi apgalvojot, ka tieši *Android* produkts ir lētāks, labāks un izdevīgāks.

iOS lietotnes ir iespējams nolādēt no "Apple App Store" kopš 2008. gada, kad pirmo reizi ar 500 lietotnēm tas tika atvērts. Tajā uz šo brīdi ir jau pieejamas vairāk kā 2,2 miljoni lietotnes [17], kas ir par 0,5 miljoniem mazāk nekā *Android* lietotņu veikalā. Publicējot lietotni *iOS* veikalā, lietotne tiek pārbaudīta un testēta. Šīs pārbaudes rezultātā lietotne var tikt arī noraidīta.

2. Izstrādes vides

Izstrādes vide ir viena no svarīgākajām izstrādātāja darba sastāvdaļām, jo tieši tur notiek lietotnes izstrāde un testu veidošana, tādēļ to izvēle ir jāveic ļoti rūpīgi un jāņem vērā katrs vides aspekts. Izstrādes vides ir ar visdažādākajām iespējām un iebūvētiem palīglīdzekļiem izstrādātājam. Izstrādātājiem ir jāspēj vidi izmantot savā labā, lai tā atvieglotu viņu darbu.

Šajā nodaļā tiek apskatītas abu operētājsistēmu vides.

2.1 Android

Lai izstrādātu lietotni uz *Android* platformas ir nepieciešams "Android SDK", kas nav piesaistīts pie konkrētas vides vai operētājsistēmas. Sākumā, kad izstrādātāji tika iepazīstināti ar "Android SDK", to nācās pašam piesaistīt pie vidēm, bet, rūpējoties par izstrādātājiem un viņu dārgo laiku, lielākajā daļā vides "Android SDK" ir integrēts klāt lejuplādes laikā.

"Android Studio" ir oficiālā izstrādes vide priekš *Android* platformas. Vide ar sabiedrību pirmo reizi tika iepazīstināta 2013. gada 16.maijā Google konferencē. Darba autors ar vidi pats iepazinās 2015. gada februārī, kad "Android Studio" pirmā stabilā versija tika izlaista tikai mēnesi agrāk.

"Android Studio" nav piesaistīta nevienai no operētājsistēmām, tai skaitā "Windows", "macOS" un "Linux". "Android Studio" aizstāja "Eclipse" *Android* izstrādē, kā primārā natīvo *Android* lietotņu izstrādes vide.

Bez oficiāli pazīstamās izstrādes vides ir iespējams izvēlēties kādu alternatīvu, piemēram, "Microsoft Visual Studio" [20]. Tas ir ieguvums, ja kādu iemeslu dēļ uzņēmumā, kurā strādā, ir nepieciešams izmantot konkrētu vidi, lai, piemēram, būtu iespējams strādāt ar šīs vides iebūvētajiem rīkiem. Strādājot lielās komandās, nebūtu ieteicams lietot dažādas vides izstrādē, jo būtu ļoti sarežģīti nodrošināt koda kvalitāti, ja viens cilvēks izmanto vienu vidi un otrs citu vidi. Pirms projekta vadītājs vai izstrādātāji nolemj, kurā vidē tiks izstrādāta lietotne, tomēr ir ilgi jāapspriež tas vai tiešām ir jāizvēlas kāda no alternatīvajām vidēm.

Tā kā *Android* versijas mainās bieži un kods noveco, tad pastāv liela iespējamība, ka jāveic koda labošanu, jo uz jaunākām ierīcēm, ar jaunāku operētājsistēmu lietotne vairs nefunkcionē tā kā tas bija paredzēts sākumā.

2.1. tabula

Android SDK sastāvs un to paskaidrojums

SDK sastāvdaļas	**Apraksts**
SDK platforma	Ietver sevī *Android* platformu versijas, kura tiks palaista kompilācijas laikā.
SDK rīki	Ietver sevī būtiskākos rīkus, tādus kā "ProGuard", kas samazina kodu un pēc tā ietekmes koda dekonstrukcija ir daudz sarežģītāka, tādējādi padarot lietotni drošāku.
SDK kompilācijas rīki	Ietver sevī rīkus, lai kompilētu *Android* lietotnes.
SDK platformas rīki	Ietver vairākus rīkus, kuri ir nepieciešami *Android* platformai, to starpā ir arī "ADB" rīks(komandrindas rīks), kurš ļauj sazināties ar *Android* ierīcēm.
Android emulators	"QEMU" bāzēts rīks, paredzēts ierīču emulācijai, lai testētu un atkļūdotu savu lietotni.
Balsta repozitorija	Ietver lokālu "Maven" repozitoriju atbalstītām bibliotēkām, kas nodrošina papildinātu "API" kopu, kas ir saderīga ar lielāko daļu *Android* versiju.
Intel vai ARM sistēmas bildes	Sistēmas bildes, kas ir nepieciešamas, lai būtu iespējams ielādēt *Android* emulatoru.

Apskatot tabulu (skat. 2.1. tabulu) var redzēt, ka izstrādātājs pats var izlemt, kas viņam ir nepieciešams no "SDK". Plānojot jaunu projektu, ir jāņem vērā tas, ka bieži tiek publicētas jaunas *Android* versijas, un tāpēc projektā ir jāierēķina papildus laiks jauno versiju izpētei. To atvieglo internetā atrodamie daudzie piemēru piedāvājumi, bet kā darba autors ir sapratis, praksē ne vienmēr tas ir tik vienkārši, un piemēri nestrādās uzreiz bez jebkādas piepūles. Piemēram, ja projektā ir noteikts, ka *Android* jaunās versijas netiks atbalstītas pēc, piemēram, 4.1.x versijas, tad

tas nav jādara, bet neviens konkurētspējīgs uzņēmums to nepieļautu, jo tas ierobežotu pieejamo tirgu.

Kā jau tika minēts, tad *Android* lielākais pluss ir tas, ka izstrādei nav jāizmanto obligāti "Android Studio", kas ir oficiālā vide, bet iespējams izmantot kādu no alternatīvām. Vispopulārākā joprojām pastāv "Eclipse" vide, jo tā bija *Android* oficiālā izstrādes vide pirms "Android Studio". Taču šeit rodas jautājums, kādu lietotni izstrādātājs vēlas veidot, vai tā būs tīmekļa bāzēta lietotne vai natīvā, vai hibrīd lietotne, kas ir pa vidu abiem šiem veidiem. Šī darba ietvaros tālāk tiks apskatīts vairāk natīvais virziens. Uz šo brīdi populāras kļūst vides, kā "Microsoft Visual studio", "Apache Cordova" un "IntelliJ IDEA", bet šīs vides vairāk bāzējas uz tīmekļa lietotnēm, kas ietaupa laiku kompānijām, zaudējot ātrdarbību un iegūstot kļūdas, jo tīmekļa lietotnes ir iespējams vienlaicīgi kompilēt uz visām operētājsistēmām, riskējot ar to, ka kaut kas iespējams strādās uz vienas operētājsistēmas un nestrādās uz citas. Tīmekļa lietotnes nav tik atsaucīgas un atbildes laiks ir lēnāks nekā uz natīvām lietotnēm. Tomēr uz tīmekļa lietotnēm tiek izveidotas vizuāli pievilcīgas lietotnes, jo "Html" ar, piemēram, "Boostrap" ietvaru, padara lietotnes skatus ātri un vizuāli pievilcīgi veidojamus, tāpēc dažkārt projektos tiek izvēlēts veidot hibrīd lietotnes. Tas nozīmē, ka skati tiek veidoti ar „Html" un "CSS" palīdzību, kas tiek palaists uz "WebView" un loģika tiek rakstīta ar "Java" palīdzību, un viss kopā savienots ar "JavaScript". Darba autors pats ir veidojis hibrīd lietotni, tīmekļa lietotni, kā arī vairākas natīvas lietotnes un no pieredzes darba autors var teikt, ka veidojot nativu vai tīmekļa lietotni jaunam izstrādātājam var būt neinteresanti, bet veidojot hibrīd lietotni, tiek izmantotas vairākas valodas, tehnoloģijas un izstrādātājam nav jādara viens un tas pats vairāku mēnešu ietvarā.

Vēlētos piebilst, ka pieminētās vides nav vienīgās un to izvēle ir milzīga. Darba autors ir redzējis kā *Android* lietotne tiek veidota uz "Unity" vides, kas izskatās ļoti interesanti, kur ir iespējams veidot modeļus ar daudz lielākām grafiskām iespējām nekā uz kādas no citām vidēm. Šeit visa loģika tiek veidota ar "JavaScript" palīdzību. Tas nozīmē, ka ja izstrādātājs vēlas, tad lietotne var būt tīmekļa vai hibrīd lietotne, ja "Unity" modelis tiek pēc tam piesaistīts "Android Studio" un loģika rakstīta ar "Java" palīdzību.

2.2 iOS

Lai izstrādātu lietotni uz *iOS* platformas, tā pat kā uz *Android* platformas ir nepieciešams "iOS SDK". "iOS SDK" ir jau integrēts izstrādes vidē un nav nepieciešama tās papildus instalācija. Pirmo reizi izstrādātājiem pieejams šis "SDK" kļuva 2008. gada februārī.

"Xcode" ir *iOS*, "watchOS", "tvOS" un "macOS" oficiālā izstrādes vide kopš 2003. gada. Darba autors ar vidi pats iepazinās 2016. gada decembrī.

2.2. tabula

iOS SDK sastāvs un to paskaidrojums

SDK sastāvdaļas	**Apraksts**
Cocoa Touch	Ietver sevī vairākas saskarnes komponentes, piemēram, kameras atbalstu, paziņojumus lietotājam, daudzskārienu kontroles.
Media	Ietver sevī video un audio komponentes, piemēram, audio ierakstīšanu vai video atskaņošanu.
Core Services	Ietver sevī datu un tīkla komponentes, piemēram, adrešu grāmatu vai failu piekļuves iespējas.
Core OS	Ietver sevī komponentes, kuras nav pieejamas parastam lietotājam, piemēram, sertifikāti vai barošanas pārvaldība.
iOS emulators	Programmatūra ierīcēs emulācijai, lai testētu un atkļūdotu savu lietotni.

Apskatot tabulu (skat. 2.2. tabulu), var redzēt no kā sastāv "iOS SDK". Atšķirībā no "Android SDK" lietotājs nevar izvēlēties tā sastāvu, jo viss jau ietilpst "SDK" [21].

Tā pat kā uz *Android* platformas "iOS SDK" versijas mainās ļoti bieži un kods noveco, tāpēc pastāv liela iespējamība, ka būs jāveic koda labošana, jo uz jaunākām ierīcēm ar jaunāku operētājsistēmu lietotne vairs nestrādā tā kā tas bija paredzēts sākumā.

Atšķirībā no *Android, iOS* bez oficiāli zināmās vides natīvo lietotņu izstrādei nav citas alternatīvas vides. Protams, ir iespējams veidot tīmekļa lietotnes, tādējādi ir iespējamas vairākas alternatīvas, kas tika jau apskatītas iepriekšējā nodaļā par *Android* vidi.

2.3 Kopsavilkums

Izstrādes vides izvēle ir svarīgs brīdis projekta gaitā un šīs zināšanas samazina iespējamās problēmas projekta laikā. *iOS* vienīgā izstrādes vide ir "Xcode", bet uz *Android* platformas ir iespējams veikt izvēli, atkāpjoties no oficiālās vides, kas nebūtu ieteicams, jo oficiālās vides ir optimizētas tieši šīm platformām. "Xcode" vide ir pieejama tikai un vienīgi uz "macOS", bet "Android Studio" ir pieejama uz visām operētājsistēmām. No tā var secināt, ka *iOS* izstrāde var prasīt lielākus projekta resursus.

Apskatot platformu "SDK", var secināt, ka izstrādātājam nav iespējas izvēlēties nepieciešamo no "iOS SDK", tādējādi zaudējot daudz laika gaidot, kamēr uz sistēmas tiek uzstādītas nevajadzīgās komponentes. Abi "SDK" ir "MVC" orientēti, tas nozīmē, ka ja tiek izstrādāts kods *Android*, tad ir iespējams viegli izstrādāt paralēli kodu objektiem *iOS,* jo, piemēram, uz *Android* skata koda klase atradīsies starp „Activities" un uz *iOS* atradīsies starp „ViewControllers" u.t.t..

3. Izstrādes vides valodas

Šajā nodaļā tiek aprakstītas *Android* un *iOS* operētājsistēmas izstrādes vides valodas.

3.1 Android

Android natīvās lietotnes tiek izstrādātas izmantojot "Java" programmēšanas valodu, kas ir "Sun Microsystems" izstrādāta objektorientēta programmēšanas valoda. "Java" valoda tika izstrādāta 1991. gadā un prasīja 18 mēnešu darbu un pirmo reizi ar sabiedrību tika iepazīstināta 1994. gadā. Šīs programmēšanas valodas sākotnējais vārds bija "Oak". Tas radās tāpēc, ka viena izstrādātāja ofisa logā bija vienmēr redzams ozols, vēlāk valoda tika pārsaukta par "Green" un galu galā par "Java", ko visi labi pazīst.

Veidojot "Java" valodu, tika noformulēti pieci galvenie mērķi:

1)valodai ir jābūt vienkāršai, objektorientētai, pazīstamai;

2)valodai ir jābūt drošai;

3)valodai ir jābūt neatkarīgai no datora arhitektūras;

4)valodai ir jātiek izpildītai ar augstu veiktspēju;

5)valodai ir jābūt interpretējamai un dinamiskai.

Valodas sintakse tika veidota līdzīga "C" valodai un "C++" valodai, kā valodas aizstājēja. Darba autors var atsaukties uz to, ka, ja programmētājs pārzin "C" valodu vai "C++", tad viņam ir viegli saprast "Java" valodu, kā arī ja programmētājs pārzin "Java" valodu, tad viņam ir viegli saprast "C++" vai "C". Tā kā darba autors, sākot mācīties "Java" valodu labi pārzināja "C++" valodu, tad viņam tas nesagādāja lielas grūtības, taču problēmas radīja saprašana kā "Java" valoda strādā uz "Android Studio" vides.

"Java" valoda kļūst populārāka ik gadu un tādēļ, domājot par izstrādātājiem, valoda tiek uzlabota ik pēc pāris gadiem. Tabulā ir iespējams apskatīt, kā gadu gaitā ir mainījušās "Java" versijas (skat. 3.1. tabulu).

3.1. tabula

Java programmēšanas valodas versijas

Versija	**Izlaišanas datums**
JDK 1.0	1996. gada 23. janvāris
JDK 1.1	1997. gada 19. februāris

Versija	Izlaišanas datums
J2SE 1.2	1998. gada 8. decembris
J2SE 1.3	2000. gada 8. maijs
J2SE 1.4	2002. gada 6. februāris
J2SE 5.0	2004. gada 30. septembris
Java SE 6	2006. gada 11. decembris
Java SE 7	2011. gada 28. jūlijs
Java SE 8	2014. gada 18. marts

3.2 iOS

iOS natīvās lietotnes atšķirībā no *Android* ir iespējams izstrādāt izmantojot "Swift" un "Objective-C" programmēšanas valodām. Projekta ietvarā ir iespējams izmantot gan tikai vienu valodu pēc izvēles, gan arī abas valodas. "Swift" kods tiek rakstīts klasēs, kuras paplašinājums ir ".swift", bet "Objective-C" tiek rakstīts vienai klasei piederot diviem failiem ar paplašinājumiem ".m" un ".h". Ja projekta ietvarā ir nepieciešams izmantot abas valodas, piemēram, ir nepieciešams izmantot kādu bibliotēku, kas ir vecāka nekā "Swift" valoda, tad "Objcetive-C" klasēm tiek izveidots speciāls galvas(header) fails, kurš palīdz "Swift" kodam atpazīt "Objective-C" klases.

Pirms "Swift" valodas iepazīstināšanas, vienīgā *iOS* izstrādes valoda bija "Objective-C". "Objective-C" valoda ir objektorientēta un tika izstrādāta pateicoties "Stepstone" kompānijai 1980s gados.

"Swift" tā pat kā "Objective-C" ir objektorientēta programmēšanas valoda, kas tika izstrādāta pateicoties "Apple Inc.". "Swift" valoda ir drošāka par "Objctive-C", jo, piemēram, deklarējot mainīgos, ir jāparedz vai mainīgais būs tukšs tā izmantošanas laikā.

Ar "Swift" pirmo reizi tika iepazīstināts 2014. gadā WWDC konferencē [6]. "Swift" valoda 2015. gadā WWDC konferencē saņēma milzīgu jauninājumu un saņēma jaunu nosaukumu „Swift 2", vēlāk „Swift 2.2" tika pasludināts kā atvērtais koda projekts. 2016. gada 13.semptembrī tika izlaista pašreizējā "Swift" versija „Swift 3.0".

Pēc trīs gadiem, tas ir 2017. gada martā, "Swift" valoda iekļuva 10 mēnešu TIOBE populārāko programmēšanas valodu topā [7]. "Objective-C" ieņem tikai 16. vietu un pārliecinoši pirmajā vietā, kā populārākā programmēšanas valoda martā pasaulē ir "Java".

4.Emulēšana

Šajā nodaļā tiek aprakstītas *Android* un *iOS* operētājsistēmas izstrādes vides emulēšanas iespējas.

4.1. Android

Emulēšana ir izstrādes viena no procesa sastāvdaļām, kas dod izstrādātājam izveidoto lietotni testēt uz savas darba stacijas, vides ietvaros un gūt priekšstatu par to kā lietotne darbojas uz kādas no ierīcēm. Praksē pirms koda nodošanas lietotne ir jātestē uz fiziskas ierīces, jo uz emulatoriem pastāv liela iespēja, ka kaut kas varētu nestrādāt kā uz ierīces, piemēram, darba autors pats ir piedzīvojis praksē, ka "Google" autentifikācija un funkcionalitāte saistībā ar karti mēdz strādāt uz emulatora, bet testējot to uz fiziskas ierīces, sāk veidoties kļūdas un lietotne nefunkcionē, kā tā ir paredzēta. Ja lietotnē ir karte, kura tiek darbināta, izmantojot "JavaScript", tad tas palielina iespējamību, ka tas varētu nestrādāt uz fiziskas ierīces. Tāpēc emulēšana nekādā veidā nespēj pilnvērtīgi aizstāt lietotnes testēšanu uz fiziskas ierīces. Taču emulēšana ļauj testēt lietotni uz vairākām ierīcēm ar savādākām izšķirtspēju ekrāniem, ko darot uz fiziskām ierīcēm, no finansiālā aspekta un laika būtu ļoti neizdevīgi.

Par emulatoriem ir atbildīgs *Android* virtuālo ierīču rīks jeb "AVD". Emulatoram ir iespējams mainīt un piešķirt vairākas sastāvdaļas, piemēram, priekšējā kamera(skat. 4.1 tabulu).

4.1. tabula

Android Studio emulatoru sastāvs

Sastāvdaļa	**Apraksts**
Ekrāna izšķirtspēja	Sākumā izstrādātājam ir jāizvēlas ierīce un tās ekrāna izšķirtspēja.
Android versija	Pēc izšķirtspējas izvēles lietotājam ir jāizvēlas operētājsistēmas versija, kas tiek iedalīts tā "API" līmenī, jeb "API" 22 ir *Android* 5.1 versija, "API" 23=*Android* 6.0.
Virtuālās ierīces vārds	Virtuālās ierīces nosaukums, ko ir izvēlējies izstrādātājs

Sastāvdaļa	**Apraksts**
Kamera(priekšējā aizmugurējā)	Jāveic izvēle, vai kamera ir emulēta vai tiek izmantota tīmekļa kamera, vai tādas vispār nav.
Internets	Interneta ātrums un aizture.
Procesors	Procesoru kodolu skaits
Atmiņa	"RAM" un iekšējā atmiņa

Viena no emulatoru problēma ir tāda, ka tiem ir nepieciešami lieli sistēmas resursi. Neviens emulators nekad nekonkurēs ar reālu ierīci, pat ja to izmanto uz jaudīga datora. Uz emulatora lietotnes ātrdarbību nav iespējams notestēt, tāpēc uz tiem testē saskarnes un virspusēji funkcionalitāti. Lietotnes saskarnes ir jānodrošina tā, lai tās izskatītos lietotājam vienādi uz jebkuras ierīces. Tā kā lielākai daļai izstrādātāju nav pieejas pie nepieciešamajiem resursiem(fiziskām ierīcēm), tad pamatā tas tiek testēts uz vairākiem emulatoriem ar dažādām izšķirtspējām. Darba autors šādi arī ir izstrādājis vairākas lietotnes, kas ir pieejamas "Google Play", piemēram, "This Way" [1]. Rezumējot iepriekš minēto, darba autors secina, ka emulatori prasa lielus resursus, un to palaišana prasa ilgu laiku, un saskarnes testēšana tādējādi aizņem lielu daļu no izstrādes.

4.2. iOS

Izstrādājot *iOS* lietotnes, ierīcēm un projektam ir jābūt reģistrētiem "Apple developers", jo atšķirībā no *Android*, *iOS* lietotnes ir iespējams izmantot tikai un vienīgi tās ierīces, kurām tas ir atļauts. Sekojot tam, ka projektu vadītāji nosaka nepieciešamo laiku projektiem, viņiem ir jāierēķina arī laiks šo ierīču reģistrēšanai, ja tādas ir pieejamas. Tomēr ir iespējams izmantot ari "Xcode" emulatorus, kas strādā tā pat kā *Android* emulatori, bez iespējas rediģēt pašu emulatoru.

5.Tirgus

Šajā nodaļā tiek aprakstīts *Android* un *iOS* operētājsistēmas tirgus un tā analīze.

5.1.Android

Android tirgus ir sadalīts vairākos "API" līmeņos jeb *Android* operētājsistēmas versijās, piemēram, "API" 18 ir *Android* versija 4.3. *Android* platforma attīstās lielā ātrumā, bieži vien kods noveco un tiek izlaistas jaunas funkcijas un iespējas, ko izstrādātājs var paveikt ar lietotni. Taču šīs funkcijas ir ierobežotas ar noteiktu "API" līmeni, piemēram, klase "android.media.tv" [2], kas ir pieejama tikai no "API" 19. Darba autors pats ir pieredzējis to, ka, ir funkcija, kas nomaina noteiktas pogas fona krāsu, bet šī funkcija ir pieejama tikai no 19 "API" un projektā ir noteikts, ka lietotne ir pieejama visiem sākoties no "API" 16. Lai to apietu nākas pašam izdomāt kā to paveikt, izmantojot kādu citu pieeju, jo lielākajā daļā pamācību un cilvēku piedāvājumu ir domāta jaunākām versijām. Kā iespējams redzēt tabulā (skat. 5.1. tabulu) [5], tad lielākā daļa lietotāju ir "API" 19, 22 un 23 līmeņos, tāpēc būtu ļoti ieteicams lietotnes veidot tā, lai tās būtu paredzētas, šim operētājsistēmas versijām. Ievērojamas izmaiņas tika veiktas pēc 15 "API" līmeņa, tāpēc pēc darba autora pieredzes un zināšanām, būtu ieteicams lietotnes izstrādāt no 16 "API" līmeņa.

5.1. tabula

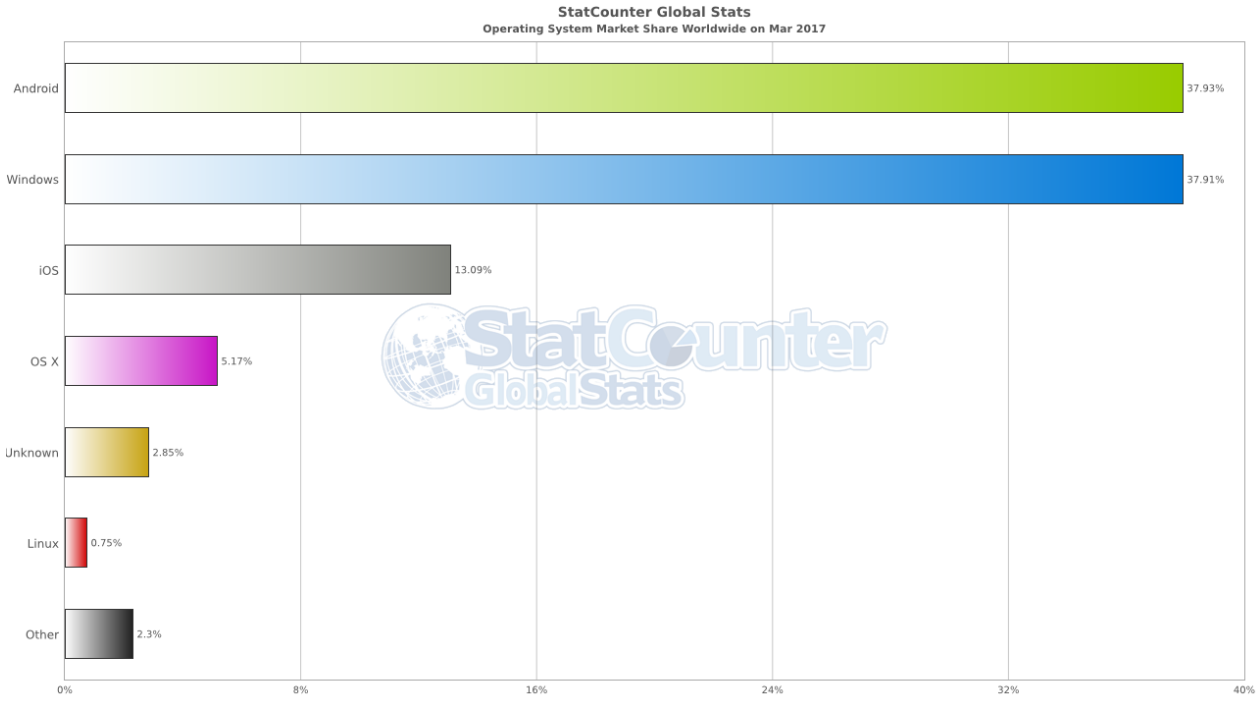
Android operētājsistēmas versiju izplatība

Versija	**Versijas nosaukums**	**API**	**Izplatība%**
2.3.3 – 2.3.7	Gingerbread	10	1.0%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	0.8%
4.1.x	Jelly Bean	16	3.2%
4.2.x		17	4.6%
4.3		18	1.3%
4.4	KitKat	19	18.8%
5.0	Lollipop	21	8.7%
5.1		22	23.3%
6.0	Marshmallow	23	31.2%

Versija	Versijas nosaukums	API	Izplatība%
7.0	Nougat	24	6.6%
7.1		25	0.5%

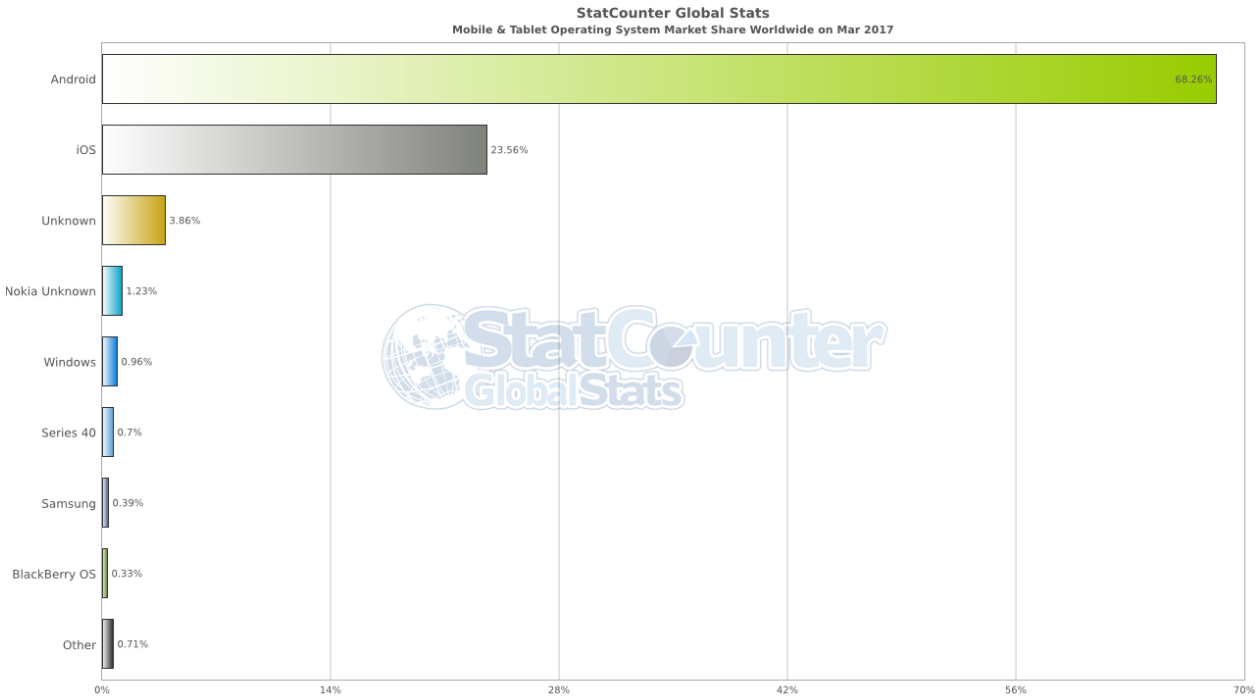
Android lietotnes tiek lejuplādētas no "Google Play" veikala, kas ir viens no lielākajiem mobilo lietotņu izplatītājiem pasaulē, 2017. gadā veikalā ir pieejamas apmēram 2,7 miljoni lietotnes, kur 13% sastāda zemas kvalitātes produkti [8]. Šo operētājsistēmu izmanto vairāk nekā 1.4 miljardi lietotāju jau 2015. gadā [9].

Izlasot jaunāko pētījumu no tīmekļa analītikas kompānijas "StatCounter", tika uzzināts, ka pirmo reizi vēsturē 2017. gada martā *Android* operētājsistēma kļuva par pasaules populārāko OS interneta izmantotāju (skat. 5.1 attēlu) [10].



5.1. att. **Operētājsistēmas tirgus interneta izmantošanā [11]**

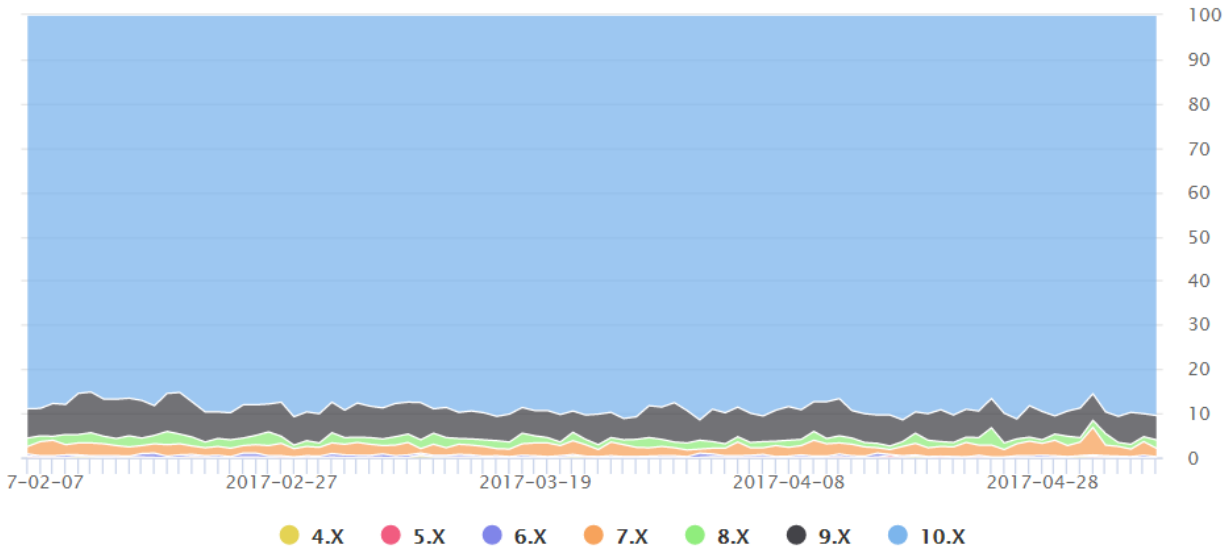
Šie dati ir ieskaitot arī konsoles un personīgos datorus, ja mēs apskatām datus, kur ir tikai mobilās ierīces un planšetdatori, tad šeit ar gandrīz 45% pārsvaru uzvar *Android* (skat. 5.2 attēlu).



5.2. att. **Operētājsistēmas tirgus interneta izmantošanā [11]**

5.2.IOS

Ne tikai *Android* tirgus, bet arī *iOS* tirgus ir sadalīts versijās, ar vispopulārāko versiju 10.x, kas ir 90.6% no *iOS* OS tirgus, pēc kā seko 9.x versija ar 5.5%, tālāk 8.x un 7.x versijas, katra ar 1.9% un pēdējā ir 6.x versija tikai ar 0.2% (skat. 5.3 attēlu) [19]. Ir jāpiemin, tas, ka atšķirībā no *Android*, kur mobilā iekārta ietekmē to, kāda ir maksimālā versija, tad uz *iOS*, dzelži neietekmē to, kāda var būt operētājsistēmas versija.



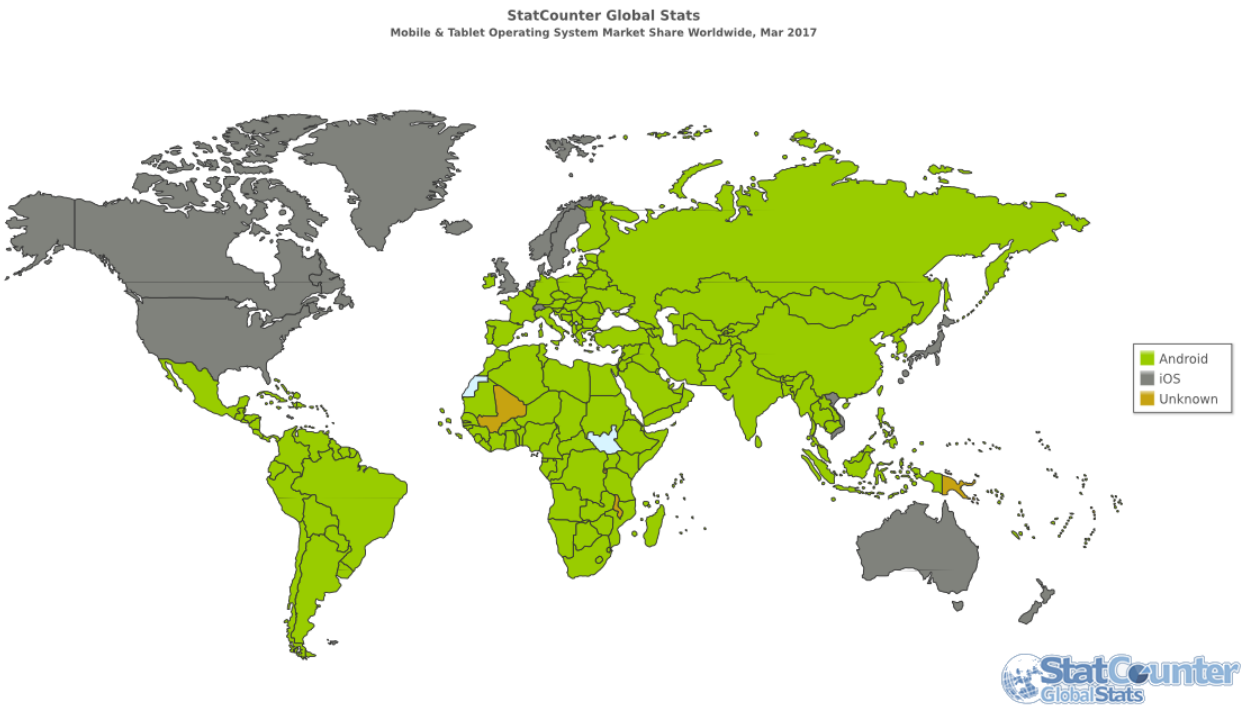
5.3. att. **IOS operētājsistēmas versiju izplatība[19]**

5.3.Kopsavilkums

Atsaucoties uz 2013. gadā veiktu pētījumu par *Android*, *iOS* un citu operētājsistēmu mobilam iekārtām, kas ir izsūtīts attiecīgajā gadā [3], tiek secināts, ka *Android* ir populārākā operētājsistēma tirgū ar apmēram 80% un *iOS* ar apmēram 20%, pēc kā seko pārējās operētājsistēmas ar atlikušajiem 10%. Bet šie cipari var mainīties katru gadu, katru ceturksni. Darba autoram pētot citus jaunākus avotus [4] tika secināts, ka *iOS* svārstās tirgū starp 10% un 20%, taču *Android* starp atlikušajiem procentiem pieder apmēram 80% tirgus, kur pārējās operētājsistēmām sadala tirgus atlikušos 10% līdz 20 %.

Darba autors uzskata, ka *iOS* pieeja operētājsistēmas versijai ir daudz labāka nekā *Android*, jo uz savas iekārtas var uzstādīt pašu jaunāko versiju, pat ja mobilā ierīce ir ļoti veca un nespēj izturēt noslodzi, tai ir jābūt lietotāja izvēlei, nevis noteiktai pēc izstrādātāja.

Izstrādājot lietotnes ir noteikti zināms tas, kāda ir lietotnes mērķauditorija, tai skaitā valstis, kurās tā tiks izplatīta. Tāpēc ir svarīgi apzināties, kura operētājsistēma ir populārāka šajos reģionos (skat. 5.4 attēlu).



5.4. att. **Operētājsistēmas popularitāte atsevišķos reģionos [11]**

iOS lietotnes statistiski iegūst 45% vairāk ienākumu no viena lietotāja nekā *Android*, kā arī pastāv 10% lielāka iespēja *iOS* lietotājam, ka viņš veiks pirkumu lietotnē [22]. Piemērs ir 2014. gada pirmais ceturksnis, "Google Play" veikala lejuplādes pārspēja "iOS App" veikala lejuplādes par 45%, toties *iOS* ienākumi bija 85% lielāki nekā tos ieguva *Android*. *Android* lietotnes visvairāk nopelnīja uz piemaksām, ko ieguva par papildu funkciju piemaksām, ko izmanto lietotāji.

Android lietotāji pavada 17% vairāk laika lietotnēs nekā *iOS* lietotāji mēnesī [22].

"Apple" lietotāji ir daudz uzticamāki par *Android* lietotājiem, kur 62% *iOS* lietotāji ir apmierināti ar savu ierīci un 80% no viņiem iepriekš jau ir piederējusi *iOS* ierīce, turpretim tikai 48% *Android* lietotāji ir apmierināti ar viņu ierīcēm. Protams šo rezultātu iespaido lielā kārtā ierīču brends, jo lielākai daļai *Android* ierīču nav klāt "Google" vai "Samsung" brends, kā arī "Apple" pieder lieliskas marketinga kampaņas [23].

Viens no viss svarīgākajiem aspektiem, kas ir jāapskata, ir lietotnes publicēšanas iespējamība. Izveidojot *Android* lietotni, izstrādātājam ir jāautinficējas "Google Play" veikala, jāsamaksā vienreizējā iemaksa, kas ir 25 ASV dolāri un jāaugšuplādē lietotnes fails, kuru jau pēc pāris stundām varēs lietotāji lejuplādēt savā mobilajā ierīcē.

Taču uz *iOS* tas viss ir sarežģītāk, vispirms izstrādātājiem ir jāveic 99 ASV dolāru iemaksa katru gadu, pēc kā seko lietotnes augšuplāde, tālāk šo lietotni testēs sertificēta persona no "Apple". Pastāv iespēja, ka lietotne tiks noraidīta un tā būs jālabo, kā arī šī testēšana aizņem vairākas dienas. Ja lietotne ir garlaicīga, aizskaroša vai neorģināla, tad tā ar lielu varbūtību tiks noraidīta. Tātad, izstrādātāji var pavadīt vairākus mēnešus veidojot lietotni un izstrādes beigās tā var tikt noraidīta. Šādi veikals uztur augstus standartus un lietotnes ir interesantas un daudzveidīgas.

6.Lietotnes uzbūve

Labam *Android* vai *iOS* lietotnes izstrādātājam ir jāpārzina ne tikai vide un valoda, bet arī lietotnes dzīves cikls un lietotnes uzbūve. To zināt ir ļoti svarīgi, jo bieži, kad tiek izsaukta noteikta funkcija vai projektos, strādājot vairākiem cilvēkiem pie lietotnes, būtu grūti tajā orientēties.

6.1.Android

Šajā nodaļā tiek aprakstītas *Android* lietotnes uzbūve.

6.1.1. Lietotnes sastāvdaļas

Kā jau darba autors minēja iepriekšējās nodaļās, tad *Android* lietotnes tiek rakstītas "Java" valodā un pateicoties "Android SDK" tiek izveidots ".apk" fails, kas ir lietotnes instalācija. Šis fails sastāv no visiem resursiem, bibliotēkām un izstrādātāja sarakstītā koda, lai lietotni būtu iespējams instalēt uz ierīces, taču papildus fails satur unikālu atslēgu, kura norāda uz to, kuram izstrādātājam pieder lietotne.

"Apk" fails ir viegli izplatāms starp jebkura veida *Android* ierīcēm, jo pret to izplatīšanu nekādas aizsardzības nav, tāpēc par to ir jāparūpējas izstrādātājiem, lai neuzmanības gadījumā kāds "apk" fails netiktu atstāts kādā publiskā vietā.

Katra lietotne tiek darbināta savā virtuālā mašīnā, izolācijā no citām lietotnēm. Kā jau tika minēts šajā darbā, tad *Android* ir daudzlietotāju "Linux" sistēma, kur katra lietotne ir kā cits lietotājs. Pēc noklusējuma sistēma katrai lietotnei piešķir savu unikālu lietotāja identifikatoru. Šo identifikatoru izmanto tikai sistēma un lietotnes šo identifikatoru nezina, tādējādi sistēma piešķir atļaujas visiem failiem lietotnē, tā lai tikai lietotne ar šo lietotāja identifikatoru var piekļūt šiem failiem. Pēc noklusējuma katra lietotne tiek darbināta savā "Linux" procesā, *Android* sistēma šo procesu darbina, kad jāiedarbina kāda no lietotnes komponentēm un izslēdz procesu, ja tas vairs nav nepieciešams vai ir nepieciešams atbrīvot atmiņu citām lietotnēm.

Android sistēma darbojas pēc mazākās privilēģijas(least privilege) principa, kas nozīmē, ka katra lietotne, pēc noklusējuma var piekļūt tikai komponentēm, kuras nepieciešamas, lai lietotne spētu izpildīt savu darbu. Šādi lietotne nevar piekļūt sistēmas daļām, kurām tai nav dota

piekļuves atļauja. Tomēr ir iespējams, ka lietotne var dalīties ar informāciju ar citām lietotnēm un piekļūt sistēmas servisiem. Lietotne var piekļūt sistēmas servisiem, ja "Manifest" failā tiek pieprasīts pēc kāda servisa piekļuves, piemēram, kameras, kad jau lietotnes lietotājs instalēs lietotni, viņš tiks informēts par to, ka lietotnei ir jāpiekļūst kamerai. Apskatot otru variantu, kad ir jāveic datu apmaiņa starp divām lietotnēm, tad tas arī ir paveicams, lietotnēm ir jāpiešķir vienāds lietotāja identifikators un šīm lietotnēm ir jāpieder vienāds sertifikāts.

Lietotnes komponentes ir kā lietotnes pamats, bez kā nav iespējams izveidot lietotni (skat. 6.1 tabulu).

6.1. tabula

Lietotnes komponentes

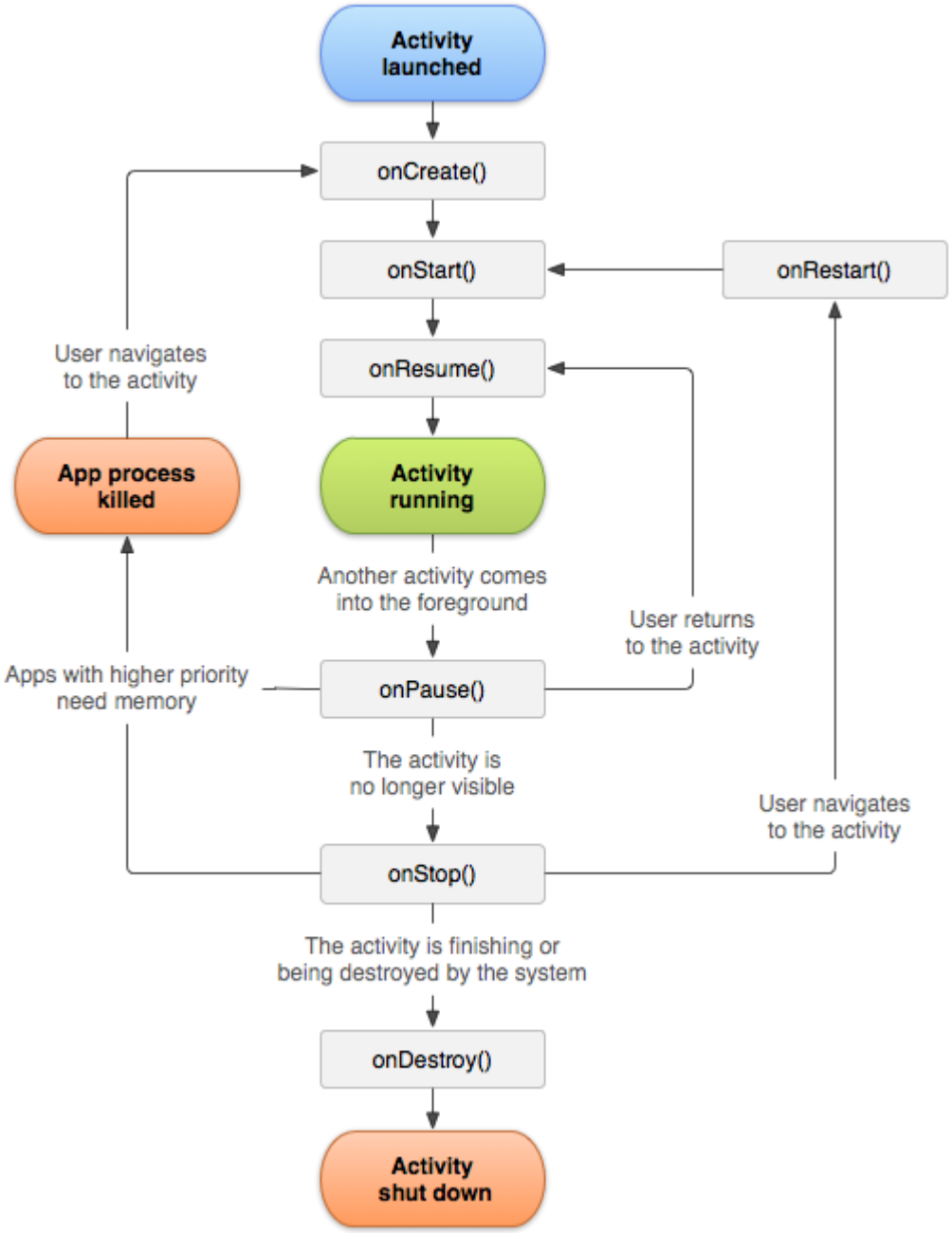
Komponente	**Apraksts**
Activities	Aktivitātes darbojas kā ieejas punkts lietotājiem, tās reprezentē ekrānu ar vienu konkrētu saskarni, piemēram, video apskatīšanas lietotne, kur viena aktivitāte attēlo sarakstu ar visiem video, otra aktivitāte attēlo pašu video skatu un trešā lietotāja skatu, kurā ir viņa informācija vai iestatījumi. Neskatoties uz to ka aktivitātes strādā kopā, lai izveidotu lietotni, katra aktivitāte ir neatkarīga no citām.
Services	Servisi ir atbildīgi par to ka lietotne strādās fonā, jebkādu iemeslu dēļ. Tā ir komponente, kas darbojas fonā, lai izpildītu ilgtermiņa operācijas, kuru pārvalda aktivitātes, piemēram, kamēr lietotājs darbojās ar lietotni, fonā tiek nolādēts kāds videoklips vai dziesma.
Broadcast receivers	Apraides uztvērējs ir komponente, kas atļauj sistēmai piegādāt notikumus lietotnei ārpus regulāras lietotāja plūsmas, atļaujot lietotnei atbildēt sistēmas paziņojumiem. Piemērs varētu būt modinātājs, kamēr nav nepieciešams modinātājs, lietotne nestrādā, kā pienāk

Komponente	Apraksts
	atzīmētais laiks, tiek palaists modinātājs.
Content providers	Satura nodrošinātājs ir komponente, kas pārvalda lietotnes koplietojamu datu kopu, kurus glabā failu sistēmā, datubāzē vai jebkurā citā datu glabātuvē, kam var piekļūt lietotne. Protams šī komponente var lasīt un rakstīt datus, kas ir privāti tikai un vienīgi šai lietotnei.

Unikāls *Android* sistēmas aspekts ir tāds, ka viena lietotne var uzsākt citas lietotnes komponenti, piemēram, ja mēs vēlētos ierakstīt video ar ierīces kameru, tad lietotne varētu izmantot citu lietotni, kas ir paredzēta šim procesam nevis tā vietā rakstot aktivitāti, kas to darītu. Kad video ir ierakstīts, tas tiek pārsūtīts uz lietotni un lietotājs pat nav pamanījis, ka ir tikta izsaukta cita lietotne. Tas ir paveicams ar nodomu(intent), izstrādātājam ir jānorāda, kādu komponenti viņš grib izsaukt, un sistēma pēc tam to aktivizē lietotāja vietā. Tas tiek sasniegts pateicoties tam, ka atšķirībā no citām sistēmām *Android* nav noteikta ieejas punkta, kas lielākai daļai ir main() funkcija.

6.1.2.Dzīves cikls

Šajā darbā tika apskatītas vairākas lietas, tomēr viena no svarīgākajām lietām, kas ir jāzina jebkuram *Android* izstrādātājam, ir lietotnes dzīves cikls, jo izstrādātājam ir jāparedz viss, sākot ar to, kas var notikt ar lietotni. Viņam ir jāzina īstais brīdis kad izsaukt, kādu funkciju un jāzina īstais brīdis, kad apturēt kādu no procesiem (skat. 6.1 attēlu).



6.1. att. **Dzīves cikls [13]**

Katra aktivitāte sastāv no dzīves cikla metodēm, kuras tiek izsauktas aktivitātes izsaukšanas sākumā, apturēšanas brīdī un iznīcināšanas brīdī (skat. 6.2 tabulu).

6.2. tabula

Dzīves cikla metodes

Metode	**Apraksts**
onCreate()	Šī metode tiek izsaukta, kad sistēma pirmo reizi izveido aktivitāti. Parasti šeit inicializē saskarni un tās elementus, kā arī inicializē fonā darbojošos klausītājus.
onStart()	Metode tiek izsaukta, tieši pirms skats tiek parādīts lietotājam.
onResume()	Šī metode tiek izsaukta, kad skats pāriet no fona uz priekšplānu un lietotājs var sākt lietot lietotni. Kamēr lietotājs darbojas skata ietvarā, lietotne atrodas šajā skatā.
onPause()	Metode tiek izsaukta, kad lietotājs pamet skatu, vai tiek, piemēram, atskaņota mūzika.
onStop()	Tiek izsaukta, kad aktivitāte vairs nav redzama lietotājam vai aktivitāte ir beigusi darboties un tūlīt tiks iznīcināta.
onDestroy()	Tiek izsaukta tieši pirms aktivitāte tiks iznīcināta. Šeit bieži vien ir ieteicams atbrīvot atmiņu vai izdzēst kādu no nevajadzīgajiem failiem vai informācijas.
onRestart()	Ja no onStop() stāvokļa aktivitāte atgriežas, lai lietotājs varētu turpināt darbu ar viņu tiek izsaukta onRestart() metode.

Sistēma nekad neiznīcina aktivitāti tieši, turpretim tā iznīcina procesu, kurā aktivitāte darbojās. Sistēma iznīcina procesus parasti, kad tai ir nepieciešams atbrīvot “RAM”. Programmējot lietotnes, vienmēr, ir jāņem vērā, ka lietotne pēkšņi, jebkurā brīdī var tikt iznīcināta un ja tādos gadījumos kaut kas nav ticis apstrādāts, palaižot lietotni nākamajā reizē, ir iespējams, ka lietotne vairs nestrādās.

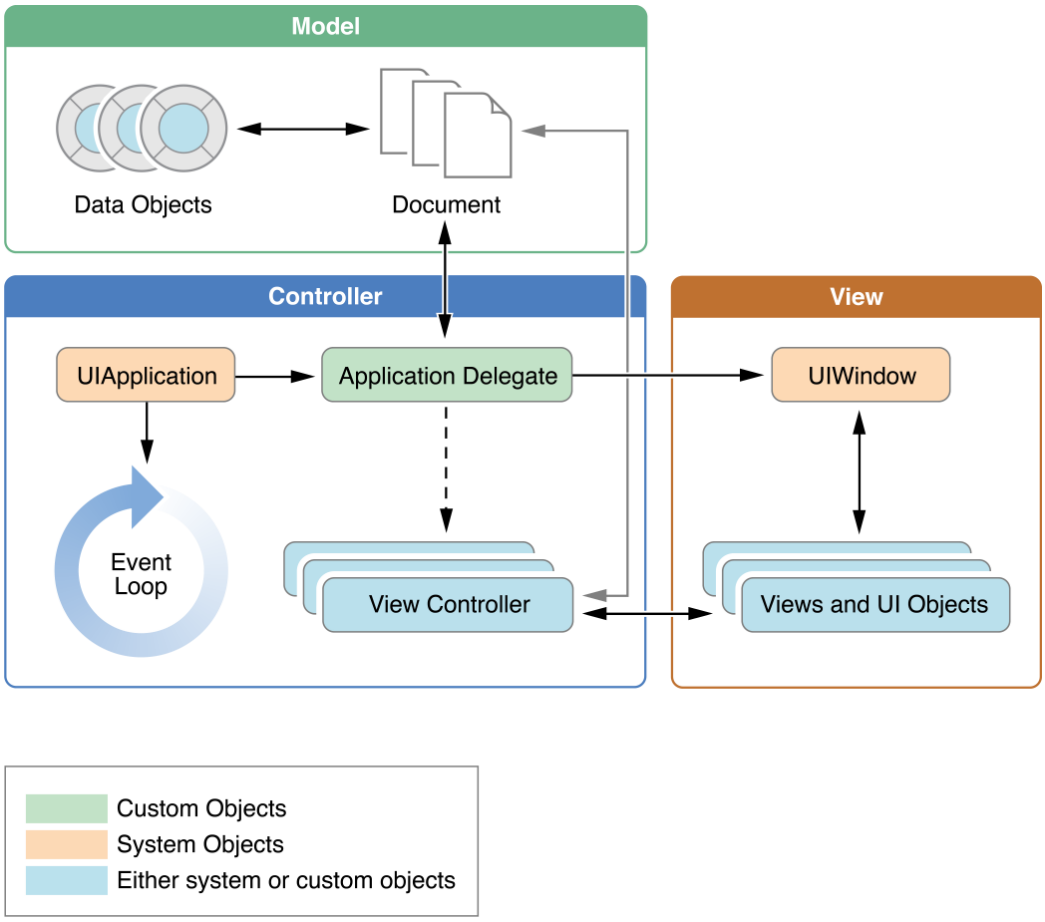
6.2.iOS

Šajā nodaļā tiek aprakstītas *Android* lietotnes uzbūve.

6.2.1.Lietotnes sastāvdaļas

Ieejas punkts jebkurai C-bāzētai valodas lietotnei ir main() funkcija un *iOS* lietotnes nav izņēmums, taču atšķirība ir tāda, ka tas nav jādara pašam izstrādātājam un šī funkcija tiek izveidota, kā pamats projektam, pateicoties "Xcode" videi.

iOS lietotnes izmanto "MVC" arhitektūras pieeju (skat. 6.2 attēlu), šī pieeja, atdala lietotnes datus no biznesa loģikas un no vizuālā attēlojuma. Šāda pieeja ir izvēlēta, jo lietotnes var tikt laistas uz dažādām ierīcēm un dažādām ekrāna izšķirtspējām.



6.2. att. **MVC modelis [14]**

Līdzīgi, kā uz *Android* operētājsistēmas, arī uz *iOS* ir pats pamats bez kā lietotnes nevar iztikt (skat. 6.3 tabulu) [14].

6.3. tabula

iOS lietotnes objekti

Objekts	**Apraksts**
UIApplication object	Šis objekts, katrai lietotnei ir tikai viens un eksistē, kā ieejas punkts lietotnē. Objekts pārvalda notikumu ciklu un citas augsta līmeņa lietotnes uzvedības.
App delagate object	Objekts ir visa centrs izstrādātāja kodam. Šis objekts strādā kopā ar "UIApplication" objektu, lai apstrādātu lietotnes inicializāciju, transakcijas un augsta līmeņa lietotnes notikumus.
Documents and data model objects	Objekts glabā lietotnes saturu un ir specifisks tikai lietotnei. Lietotne var izmantot arī dokumenta objektu, lai pārvaldītu datu modeļu objektus, tas ir labs variants, lai grupētu datus.
View controller objects	Šis objekts pārvalda lietotnes prezentēšanu uz ierīces ekrāna. Objekts attēlo viņa skatus, instalējot tos uz lietotnes ekrānā.
UIWindow object	"UIWindow" objekts koordinē skatu prezentēšanu uz ierīces ekrāna.
View objects, control objects, andlayer object	Skati un kontroles nodrošina vizuālo reprezentāciju lietotnes saturam. Skats ir objekts, kas uzzīmē saturu noteiktajā zonā un reaģē uz notikumiem šajā zonā. Kontroles ir specializēts skata tips, kas ir atbildīgs par tādu objektu implementēšanu, kā pogām vai teksta laukiem.

6.2.2.Dzīves cikls

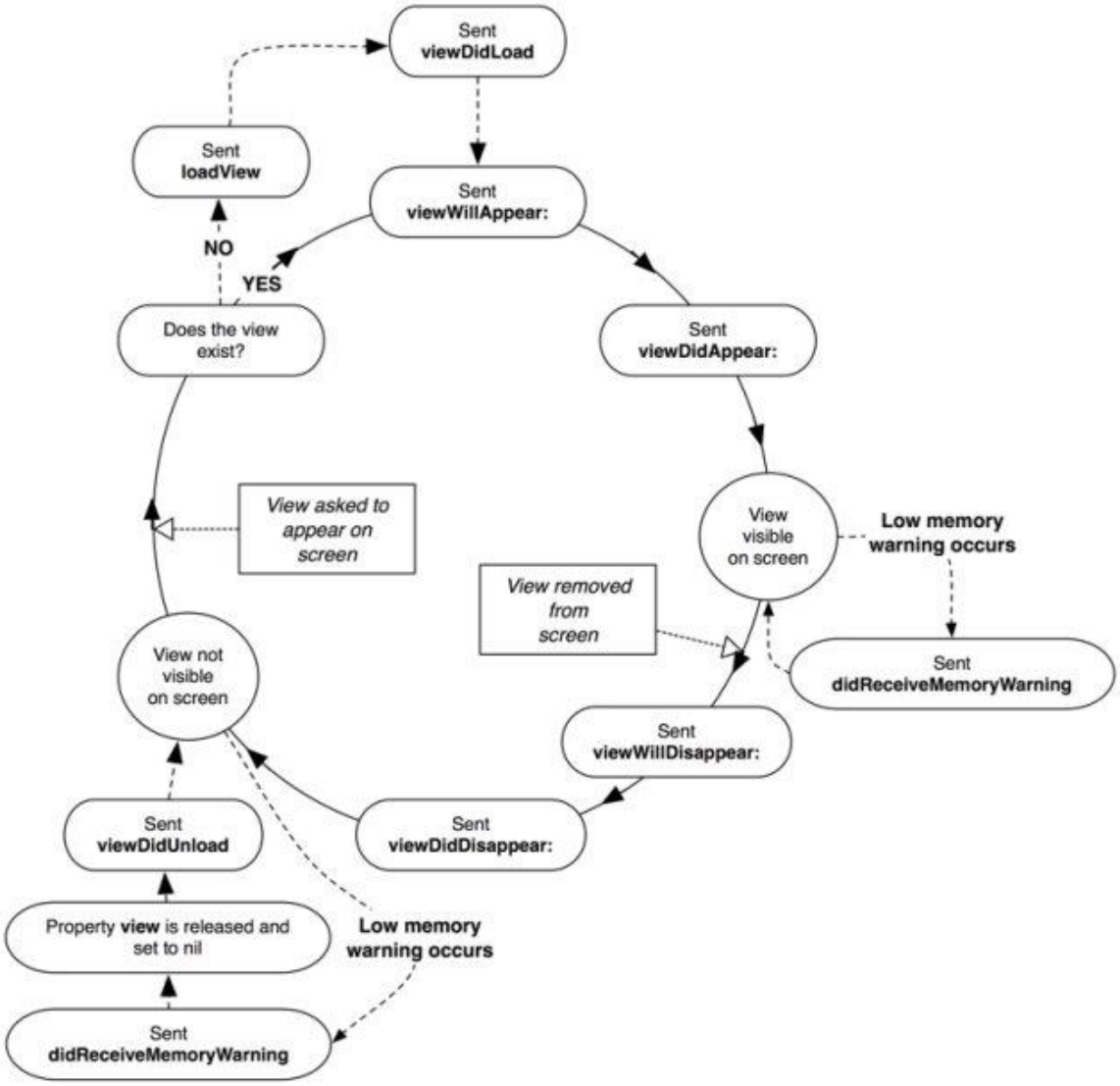
iOS sistēma pārvieto lietotāja lietotni no viena stāvokļa uz citu, attiecībā pret lietotāja veiktajām darbībām sistēmā. Tieši tā pat kā *Android*, pie šīm stāvokļa maiņām tiek izsauktas attiecīgas metodes (skat. 6.4 tabulu) [14].

6.4. tabula

iOS Dzīves cikla metodes

Metode	**Apraksts**
application:willFinishLaunchingWithOptions:	Paziņo delegātam, ka palaišanas process ir sācies, bet stāvokļa atjaunošana vēl nav notikusi. Šī ir pirmā iespēja, kad izstrādātājs var izpildīt savu kodu.
application:didFinishLaunchingWithOptions:	Šajā metodē izstrādātājam atļauj pēdējo reizi pirms lietotne ir attēlota lietotājam izpildīt savu kodu. Metode paziņo delegātam, ka palaišanas process ir gandrīz pabeigts un lietotne ir gandrīz gatava izmantošanai.
applicationDidBecomeActive:	Paziņo lietotnei, ka tā tūlīt nonāks priekšplānā.
applicationWillResignActive:	Paziņo izstrādātājam, par to, ka lietotne tūlīt pametīs priekšplānu.
applicationDidEnterBackground:	Paziņo par to, ka lietotne strādā fonā un tās darbība var tikt apturēta jebkurā mirklī.
applicationWillEnterForeground:	Lietotne tiek no fona pārvietota atpakaļ priekšplānā, bet nav vēl aktīva.
applicationWillTerminate:	Metode paziņo par to, ka lietotne tiek iznīcināta. Šī metode netiek izsaukta, ja lietotne ir apturēta.

Līdzīgi, kā uz *Android, iOS* arī ir skatu kontrolieru dzīves cikls (skat. 6.3 attēlu).



6.3. att. **Skatu kontrolieru dzīves cikls [15]**

6.3. Kopsavilkums

Android operētājsistēma ļauj viegli izplatīt lietotnes failu, kas ir ".apk" fails, bez nekādas aizsardzības, atgriezenisko lietotnes atkodēšanu aizsargā tikai atslēga, kurā ir noteikts kompānijas vārds un paroles. Toties *iOS* lietotni var izmantot tikai ierīces, kurām ir dota tāda atļauja, tas var ierobežot lietotnes neatļautu izplatību, bet tas var apgrūtināt lietotnes izstrādi, jo ir jātērē laiks ierīču reģistrēšanai.

Android liels plus ir tāds, ka lietotne var uzsākt citas lietotnes komponenti, un tādējādi varam izstrādē ietaupīt laiku.

Dzīves cikls gan *Android* operētājsistēmai, gan *iOS* operētājsistēmai ir līdzīgs, lietotņu kontrolieri/aktivitātes dzīves strādā apmēram vienādi, taču *Android* tas tiek izmantots daudz vairāk un daudz biežāk, jo biežāk lietotnei ir jāreaģē uz skata izmaiņām tieši caur dzīves cikla metodēm, *iOS* lietotnēs, turpretim, vairāk reaģē ar izstrādātāja veidotām delegāta metodēm. Dzīves cikla metodes ir ļoti līdzīgas un abu operētājsistēmu lietotnēm spēj vajadzīgā brīdī izsaukt izstrādātāja kodu.

7. Projekta izstrāde

Izstrādājot lietotnes gan *iOS*, gan *Android* sistēmai, svarīgākais ir sākumā novērtēt projekta ilgumu un apzināties, cik ilgi varētu prasīt projekta izpilde. Neskatoties uz to, ir jāapzinās, kura operētājsistēma prasīs vairāk laika lietotnes izstrādei.

Lai uzzinātu, kuras operētājsistēmas lietotne ir ātrāk izstrādājama, tika izmantots pētījums par to. Pētījumā tika izvēlēti 6 projekti, kuri tika veikti 1,5 gada laika intervālā [18]. Katram projektam tika izveidota gan *Android*, gan *iOS* natīvā lietotne. Lietotnēm tika mērīts gan koda daudzums līnijās, gan stundu daudzums pavadīts lietotņu izstrādē.

Apskatot pētījumu var secināt, ka izstrādātāji vidēji raksta apmēram 40% vairāk koda *Android* lietotnēm, nekā *iOS* lietotnēm (skat. 7.1 tabulu). Liels daudzums šī koda ir automātiski ģenerēts, toties tas ir jāpārlasa, jāatkļūdo un jāuztur, kas prasa izstrādes laiku.

7.1. tabula

Projektu koda līnijas

	iOS	**Android**	**Atšķirība %**
Projekts A	6 829	15 323	124%
Projekts B	48 671	50 756	4%
Projekts C	15 807	28 449	80%
Projekts D	5 148	14 893	189%
Projekts E	21 698	25 501	18%
Projekts F	6 956	10 347	49%
Kopējais skaits	105 109	145 269	38%

Pētījumā tika arī noskaidrots patērētais laiks lietotņu izveidei un procentuālā atšķirība. No tā var secināt, ka vidēji, apmēram, 30% vairāk laiks tiek patērēts *Android* lietotņu izstrādē nekā *iOS* lietotnēm (skat. 7.2 tabulu). Tomēr var redzēt arī, ka patērētā laika atšķirība var būt daudz lielāka, kā arī mazāka par 30%.

7.2. tabula

Projektu patērētais stundu skaits

	iOS	**Android**	**Atšķirība %**
Projekts A	241	440	83%
Projekts B	1 586	1 613	2%

	iOS	Android	Atšķirība %
Projekts C	822	1 157	41%
Projekts D	295	755	156%
Projekts E	602	647	7%
Projekts F	244	257	5%
Kopējais skaits	3 790	4 869	28%

Apskatot iegūtos datus, var secināt, ka izstrādājot lietotni *Android* operētājsistēmai ir nepieciešams 30% vairāk laika nekā *iOS* lietotņu izstrādei. Šāds iznākums ir viegli izskaidrojams. Pirmais no iemesliem ir tāds, ka lietotnes *Android* vidē tiek rakstītas izmantojot "Java" programmēšanas valodai, kura vienkārši sakot ir vairāk izvērstāka valoda par "Objective-C" vai "Swift" valodu. Rakstot vairāk koda, nozīmē to, ka ir lielāka iespēja pielaist vairāk kļūdas, tādējādi jāpatērē vairāk laiks koda labošanai.

Vēl viens no iemesliem ir tāds, ka *Android* emulatori ir nedaudz lēnāki par *iOS* simulatoriem, tas ir faktors, kas paildzina lietotnes izstrādi. Neskatoties uz to *Android* lietotnes ir jātestē uz daudz vairāk ierīcēm, kas palielina arī izplatītāja saistītu kļūdu iespējamību. Darba autors ir pieredzējis to, ka funkcionalitāte saistībā ar karti strādāja uz "Google Nexus" mobilās ierīces, bet testējot to pašu uz "Prestigio Grace" ierīces funkcionalitāte nestrādāja, kā tas bija paredzēts.

Pēdējais iemesls ir tāds, ka *Android* lietotnes skati tiek biežāk veidoti manuāli, izmantojot "XML" valodu, kur uz *iOS* vides tas nav iespējams.

Protams šajā pētījumā ņemtās metrikas nevar nosegt visus iespējamos scenārijus un problēmas lietotnes izstrādes laikā, taču vispārīgi var secināt, ka *Android* lietotnes izstrāde aizņem vairāk laika, nekā *iOS* lietotnes izstrāde.

Darba autoram apskatot citu projektu [24] (skat. 7.3 tabulu), var ieraudzīt ko līdzīgu iepriekš apskatītam pētījumam. Koda līnijas ir vairāk *Android* lietotnei. Taču *iOS* projekta laiks ir lielāks nekā *Android* projektam. To var viegli izskaidrot: šo lietotni izstrādāja cilvēks, kurš gandrīz nav programmējis ar "Java" vai "Objective-C" valodu, kā stāsta izstrādātājs, tad "Java" valodu viņam bija daudz vieglāk apgūt nekā "Objective-C", no kā var secināt, ka izstrādātājs varēja nepilnīgi apgūt "Objective-C" valodu, jo ar to iesācējam ir grūtāk strādāt.

7.3. tabula

GQueues lietotnes izstrāde

	Android lietotne	**iOS lietotne**
Projekta sākums	21.09.2012	02.03.2013
Beta versija testiem	22.12.2012	10.06.2013
Lietotne publicēta	31.01.2013	18.07.2013
Kopējais projekta laiks	4.25 mēneši	4.5 mēneši
Izstrādes laiks	870h	960h
Beta testēšana un labošana	34 dienas	38 dienas
Beta testētāju daudzums	92 cilvēki	48 cilvēki
Koda līnijas	26 981	23 872
Lietotnes izmērs	1.1MB	3.5MB

Izstrādājot lietotni, nevar paļauties uz to, ka *iOS* lietotne aizņems mazāk izstrādes laika. Kā tika noskaidrots, tad projekti var atšķirties un izstrādes laiks var atšķirties lielā amplitūdā.

Rezultāti

Bakalaura darba mērķis bija iepazīties ar pašreizējām *Android* un *iOS* izstrādes vidēm, programmēšanas valodām un tirgu, tās apskatīt un salīdzināt, kas darba gaitā arī tika paveikts.

Darba autors veica ieskatu *Android* un *iOS* plašajā vēsturē, neskatoties uz to, ka operētājsistēmas ir tikai 10 gadus vecas. Tika uzzināts, ka pirmā tika izlaista *iOS* operētājsistēma, un pēc neilga laika tika arī izlaista *Android* operētājsistēma.

Apskatot izstrādes vides un to emulatorus tika secināts, ka *iOS* OS nav alternatīvu iespēju, turpretim *Android* ir. Kā arī uz oficiālās *Android* vides ir iespējams pašam izvēlēties, ko no "SDK" izstrādātājam ir nepieciešams, tādējādi ieekonomējot dārgo izstrādes laiku. Darba procesā ir jāņem vērā arī tas, ka *iOS* simulatori strādā ātrāk nekā *Android*.

Aplūkojot izstrādes valodas tika uzzināts, ka "Java" valoda ir populārākā pasaulē un to apgūt būtu ļoti ieteicams konkurētspējīgam izstrādātājam, taču "Swift" valoda sevi pēdējā laikā aiz vien vairāk pierāda konkurences tirgū.

Darbā tālāk tika apskatīts OS tirgus, kur izstrādātājiem ir jāņem vērā daudz faktoru. Galvenais ir saprast, kādai mērķauditorijai tiks izstrādāta lietotne, kurā reģionā, kura OS ir populārāka. Tālāk izstrādātājiem jāsaprot, kurām versijām tiks veidota lietotne. *iOS* lietotāji ir lojālāki operētājsistēmai.

Darba autoram apskatot lietotnes uzbūvi tika secināts, ka dzīves cikli abām OS ir līdzīgi, taču operētājsistēmām atšķiras lietotnes gala fails, kur *Android* failu var izmantot jebkurš, kam ir pieeja failam, bet *iOS* failu var izmantot tikai reģistrētas ierīces/lietotāji.

Darba beigās tika apskatīta projekta izstrāde, kur tika secināts, ka uz *Android* kods būs apmēram 40% vairāk par *iOS*, kā arī tiks patērēts vidēji 30% vairāk laika projekta īstenošanai. Pēc šīs informācijas nevar virzīties uz priekšu, jo projekti mēdz būt dažādi un arī izstrādātāju sagatavotība un valodu zināšanas abās veida OS ir dažādas, tāpēc šie procentuālie skaitļi var atšķirties projektiem.

Secinājumi

Darba autoram veicot ieskatu OS vēsturē, likās interesanti uzzināt, ka abas OS ir tikušas iepazīstinātas ar sabiedrību viena gada robežās.

No iegūtajiem rezultātiem par izstrādes vidi un to emulatoriem, darba autors dod nelielu priekšroku *Android* oficiālai videi. Šī nodaļa nesagādāja darba autoram lielas grūtības, jo aprakstītās vides tiek izmantotas darbā.

Pēc darba autora secinājumiem daudz vieglāk un ātrāk ir iespējams apgūt "Java" valodu un ja izstrādātājam ir domas par "Objective-C" valodas apgūšanu vai "Swift" valodas apgūšanas priekšrocībām, tad būtu jāizvēlas "Swift" valodu, jo valoda ir vieglāk apgūstama, kā arī tā ir daudz populārāka un laika gaitā pārņems tirgu un pēc darba autora domām "Objective-C" valoda atkritīs.

Darbā tika apskatīts arī OS tirgus, kura darba autoram likās visinteresantākā no tēmām. Lielākās grūtības, bija atrast pēc iespējas jaunākus pētījumus un datus. Darba autors secina, ka ja lietotni vēlas veidot tikko sācis izstrādātājs ar savām idejām, tad darba autors var droši ieteikt to darīt uz *Android* vides, jo tirgus ir atvērts cilvēkiem, kuri eksperimentē un kuriem ir dārgs laiks, jo uz "Apple" veikala pastāv risks par to, ka lietotni var neapstiprināt bez iemesla un izstrādātājs tādējādi būs zaudējis savu ieguldīto laiku.

Darba autors, apskatot lietotnes uzbūvi, secināja, ka uzņēmumiem ir nedaudz labāks *iOS* stils, taču pašnodarbinātam izstrādātājam labāk tomēr izvēlēties *Android*, jo viss beigās slēpjas testēšanā un faila izplatībā. Uzņēmumiem ir svarīga datu drošība, un lietotni var testēt tikai uz *iOS* reģistrētas ierīces. Taču pašnodarbinātam izstrādātājam nav jāuztraucas par to, ka kāds viņam „nozags" ".apk" failu, jo tas ir izdarāms tikai viņa paša nolaidības gadījumā.

Visā bakalaura darba gaitā vislielākās grūtības darba autoram sagādāja projektu piemēru atrašana, ko pēc tam varētu izmantot savā darbā, jo reti kad viens izstrādātājs piedalās gan *Android*, gan *iOS* vienas lietotnes izstrādē. Darba autors bija pārsteigts par to, ka Android kods ir garāks par *iOS*.

Rezultātā darba autors secina, ka apgūt zināšanas izstrādei vieglāk būtu *Android* lietotnei. Taču izstrādes laiks šai lietotnei būs ilgāks par *iOS* lietotnes izstrādi. Tirgū realizēt gala produktu vieglāk varētu uz *Android* operētājsistēmas, taču peļņa būtu mazāka nekā, ja tas būtu ticis paveikts uz *iOS* operētājsistēmas.

Darba autors darba izpētes gaitā ieguva daudz jaunu zināšanu, kuras viņš centīsies izmantot savā turpmākajā darbā, kā arī nostiprināja jau esošās. Visinteresantākais darba autoram šajā darbā likās OS tirgus izpēte un analizēšana.

Izmantotā literatūra un avoti

1. This Way [tiešsaiste]-[atsauce 03.02.2017.]. Pieejams: https://play.google.com/store/apps/details?id=eu.opentransportnet.thisway

2. Media tv [tiešsaiste]-[atsauce 03.02.2017.]. Pieejams: https://developer.android.com/reference/android/media/tv/package-summary.html

3. Mobilo ierīču izsūtīšana [tiešsaiste]-[atsauce 20.03.2017.]. Pieejams: https://www.canalys.com/newsroom/android-80-smart-phones-shipped-2013

4. Mobilo ierīču izsūtīšana [tiešsaiste]-[atsauce 20.03.2017.]. Pieejams: http://www.idc.com/promo/smartphone-market-share/vendor

5. Android operētājsistēmas izplatība [tiešsaiste]-[atsauce 23.03.2017.]. Pieejams: https://developer.android.com/about/dashboards/index.html

6. Swift iepazīstināšana [tiešsaiste]-[atsauce 26.03.2017.]. Pieejams: https://thenextweb.com/apple/2014/06/02/apple-announces-swift-new-programming-language-ios/#.tnw_nexsWdKC

7. Programmēšanas valodas tops[tiešsaiste]-[atsauce 02.04.2017.]. Pieejams: http://www.cultofmac.com/471301/swift-is-already-of-the-worlds-most-popular-programming-languages/

8. Lietotņu skaits Google Play[tiešsaiste]-[atsauce 06.04.2017.]. Pieejams: https://web.archive.org/web/20170210051327/https://www.appbrain.com/stats/number-of-android-apps

9. Android lietotāju skaits[tiešsaiste]-[atsauce 06.04.2017.]. Pieejams: https://www.theverge.com/2015/9/29/9409071/google-android-stats-users-downloads-sales

10. Interneta lietošanas statistika OS[tiešsaiste]-[atsauce 13.04.2017.]. Pieejams: http://gs.statcounter.com/press/android-overtakes-windows-for-first-time

11. OS tirgus daļa pasaulē[tiešsaiste]-[atsauce 15.04.2017.]. Pieejams: http://gs.statcounter.com/os-market-share#monthly-201703-201703-map

12. Android lietotnes sastāvs[tiešsaiste]-[atsauce 18.04.2017.]. Pieejams: https://developer.android.com/guide/components/fundamentals.html#ActivatingComponents

13. Android dzīves cikls[tiešsaiste]-[atsauce 15.04.2017.]. Pieejams: https://developer.android.com/guide/components/activities/activity-lifecycle.html

14. iOS lietotnes sastāvs[tiešsaiste]-[atsauce 18.04.2017.]. Pieejams:

https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html
15. iOS skatu kontrolieru dzīves cikls[tiešsaiste]-[atsauce 18.04.2017.]. Pieejams: https://rdkw.wordpress.com/2013/02/24/ios-uiviewcontroller-lifecycle/
16. Pārdotais ierīču skaits 2016 gada ceturksnī[tiešsaiste]-[atsauce 25.04.2017.]. Pieejams: http://www.gartner.com/newsroom/id/3609817
17. iOS lietotņu skaits[tiešsaiste]-[atsauce 26.04.2017.]. Pieejams: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/
18. Operētājsistēmu metrikas[tiešsaiste]-[atsauce 08.05.2017.]. Pieejams: https://infinum.co/the-capsized-eight/android-development-is-30-percent-more-expensive-than-ios
19. iOS versijas[tiešsaiste]-[atsauce 12.05.2017.]. Pieejams: https://david-smith.org/iosversionstats/
20. Android Studio alternatīvās iespējas[tiešsaiste]-[atsauce 18.05.2017.]. Pieejams: http://alternativeto.net/software/android-studio/
21. iOS sdk saturs[tiešsaiste]-[atsauce 21.05.2017.]. Pieejams: https://www.macrumors.com/2008/03/06/apple-releases-iphone-sdk-demos-spore-instant-messaging/
22. Ienākumi no lietotnēm [tiešsaiste]-[atsauce 22.05.2017.]. Pieejams: https://dazeinfo.com/2014/05/27/ios-users-32-likely-make-app-purchases-android-users-engaged/
23. Lietotāju lojalitāte uz OS [tiešsaiste]-[atsauce 22.05.2017.]. Pieejams: http://www.comscore.com/lat/Insights/Blog/Android-vs-iOS-User-Differences-Every-Developer-Should-Know#imageview/0/
24. Gqueues lietotņu salīdzināšana [tiešsaiste]-[atsauce 24.05.2017.]. Pieejams: http://blog.gqueues.com/2013/07/android-vs-ios-comparing-development.html
25. Gqueues lietotņu salīdzināšana [tiešsaiste]-[atsauce 28.05.2017.]. Pieejams: https://www.androidpit.com/the-sweet-history-of-android

Bakalaura darbs “Mobilas lietotnes izstrādes teorija un prakse, izmantojot Android un iOS izstrādes vides” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Ilmārs Svilsts

Rekomendēju/nerekomendēju darbu aizstāvēšanai

Vadītājs: asoc.prof., Dr.dat. Uldis Straujums

Recenzents:

Darbs iesniegts Datorikas fakultātē

Dekāna pilnvarotā persona:

Darbs aizstāvēts bakalaura darba komisijas sēdē

Komisijas sekretāre: