LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

ŠĶĒRSPLATFORMU IZSTRĀDES RĪKU SALĪDZINĀJUMS MOBILĀS LIETOJUMPROGRAMMAS RAKSTĪŠANAI

BAKALAURA DARBS

Autors: **Jevgēnija Gluškova**
Studenta apliecības Nr.: jg16036
Darba vadītājs: Dr.dat. Uldis Straujums

RĪGA 2020

ANOTĀCIJA

Mobilo lietojumprogrammu tirgus strauji aug, tādēļ rādas liels pieprasījums pēc ātras augstas kvalitātes lietojumprogrammu izstrādes. Papildus vietējām izstrādes pieejām, kad lietojumprogrammu var izveidot tikai vienai konkrētai mobilajai platformai, savu popularitāti iegūst arī šķērsplatformu mobilās attīstības pieejas iegūst arvien lielāku popularitāti. Bakalaura darba mērķis ir izpētīt un sniegt vispārīgu pārskatu par dažiem pieejamiem šķērsplatformu izstrādes rīkiem, kā arī detalizēti analizēt divus izstrādes rīkus, lai sniegtu rīku salīdzināšanas rezultātus.

Šī darba rezultātā, izmantojot vairākus ietvarus, ir izstrādātas četras vienas lapas lietotnes. Izvēlēto šķērsplatformu rīku analīzē kā atskaites punkti tika izmantoti lietojumi, kas bija izstrādāti, izmantojot vietējās pieejas (Android un iOS), un, ar šķērsplatformu rīku palidzību, tika izstrādāti vēl divas lietojumprogrammas: Flutter un ReactNative.

Šķērsplatformu salīdzinājums atklāja atšķirības izstrādes procesā, kā arī lietojumprogrammu vispārējā veiktspējā, kas tika izveidotas, izmantojot izvēlētos starpplatformu izstrādes rīkus.

Atslēgas vārdi: Cross-platform, Native, IOS, Android, Flutter, Xamarin, NativeScript, Titanium, ReactNative, mobilo lietotņu izstrāde, vienota koda bāze

ABSTRACT

Comparison of cross-platform development tools for development of mobile application

The mobile application market is growing rapidly, and there is great demand for rapid development of high-quality applications. In addition to native development approaches, where an application can only be created for only one specific mobile platform, cross-platform mobile development approach alongside the available development tools are gaining their popularity. The aim of Bachelor Paper is to explore and provide overall overview on some of the available cross-platform development tools, as well as in detail analyze two cross-platform development tool, to provide tool comparison results.

As a result of this work four instances of simple one page application have been developed using multiple frameworks,. Applicaitons that were developed using native approach (Android and iOS), were used as a point of reference during the analysis of chosen cross-platform tools, and two more applicaitons were developed with help of cross-platfrom tool: Flutter and ReactNative.

The comparison revealed the differences in development process and overall performance of applications, that were created using chosen cross-platform development tools.

Keywords: Cross-platform, Native, IOS, Android, Flutter, Xamarin, NativeScript, Titanium, ReactNative, mobile app development, single code base

SATURS

APZĪMĒJUMU SARAKSTS

SDK (programmatūras izstrādes komplekts) - izstrādes rīku komplekts, kas programmatūras izstrādātājiem ļauj izveidot lietojumprogrammas konkrētai programmatūras pakotnei, operētājsistēmām un dažādām platformām.

IDE (integrēta izstrādes vide) - programmatūras rīku komplekts, ko programmētāji izmanto programmatūras izstrādei. Izstrādes vidē ietilpst: teksta redaktors, tulks, atkļūdotājs.

UX (lietotāja pieredze) - ir lietotāja uztvere un reakcija, kas rodas no produkta, sistēmas vai pakalpojuma lietošanas un / vai turpmākas izmantošanas.

UI (lietotāja saskarne) - saskarne, kas nodrošina informācijas pārsūtīšanu starp lietotāju un datorsistēmas aparatūras un programmatūras komponentiem

API (lietojumprogrammu saskarne) - apraksts par veidiem, kā viena datorprogramma var mijiedarboties ar citu programmu.

JS (JavaScript) - ir vairāku paradigmu programmēšanas valoda.

JIT (Just-In-Time) - ir veids, kā izpildīt programmas kodu, kas ietver kompilāciju programmas izpildes laikā - nevis pirms izpildes.

OS (Operētājsistēma) - ir saskarne starp ierīces lietotāju un ierīces aparatūru.

AOT (Ahead-of-Time) – kompilators, kas konvertē kodu izpildlaikā un glabā tās koplietotajā klases kešatmiņā.

IEVADS

Tehnoloģiskais progress ir sasniedzis lielus augstumus, un šis progress saskaņā ar sniegtiem Statista datiem, ļauj 3,5 miljardiem cilvēku visā pasaulē izmantot viedtālruņus [1]. Statista ir tirgus un patērētāju datu sniedzējs. Dažādiem viedtālruņiem var būt dažādas operētājsistēmas, plaši pazīstamās ir IOS, Android.

Saskaņā ar StatCounter, kas ir tīmekļa trafika analīzes platforma, Android un IOS kopā aizņem nedaudz vairāk par 99% viedtālruņus no tirgus daļas [23]. Tādēļ, lai nodrošinātu lietotnes popularitāti, ir svarīgi nodrošināt lietojumprogrammas versiju katrai no šīm platformām.

Lietotnes izstrāde vienā no mobilās operētājsistēmām prasa pietiekami daudz laika un izdevumu, bet tās ieviešanai otrajā vai pat trešajā operētājsistēmā būs vajadzīgs tik daudz pūļu, laika un naudas, jo būtībā katru reizi, tiek izveidota pilnīgi jauna programma.

Iespējamais veids, kā var atrisināt šīs problēmas, ir izmantot šķērsplatformas izstrādes pieeju. Šķērsplatformas izstrādes pieeja ļauj izstrādātājiem izstrādāt vienu lietojumprogrammu, kas uzreiz ir paredzēta vairākām mobilajam platformām. Tomēr vairāku platformu lietotņu izstrāde var izraisīt vairākas ar veiktspēju un ar lietotāja dizainu saistītas problēmas. Tādēļ ir svarīgi atrast pareizo šķērsplatformu attīstības rīku, kas būs piemērots jūsu vajadzībām un izvirzītiem mērķiem.

Bakalaura darba ietvaros tiks detalizēti apskatīti vairāki šķērsplatformu izstrādes rīki. Savukārt darba mērķis ir sniegt vispārēju izpratni par šķērsplatformas pieeju un tai pieejamajiem rīkiem. Kā arī sniegt divu izvēlēto izstrādes rīku analīze ar mērķi noteikt to stiprās un vājās puses. Balstoties uz analīzi, rīki tiks salīdzināti savā starpā. Šī plašā salīdzinājuma mērķis ir sniegt pamatzināšanu par abām tehnoloģijām un to atšķirībām.

Darbs sastāv no 5 nodaļām. Primaja nodaļā ir sniegts vispārīgs mobilo lietotņu izstrādes pieeju ieskāts, tiek sniegta gan vietējās, gan šķērsplatformu pieejas definīcija, turklāt ir dots minēto pieeju salīdzinājums. Otrajā nodaļā tiek sniegts šķērsplatformu attīstības rīku apraksts. Savukārt trešajā nodaļā tiek aprakstīti izstrādāto šķērsplatformu lietojumprogrammu salīdzināšanas kritēriji, kā arī norāda izvēlētos rīkus lietotnes ieviešanai. Ceturtajā nodaļa sīki aprakstīta paveikto praktisko darbu, kā tika izstrādātas un testētas lietotnes. Piektajā nodaļā ir sniegti rīku analīzes rezultāti.

1 MOBILO LIETOTŅU IZSTRĀDES PIEEJAS

Nodaļā tiek detalizēti aprakstīti mobilo lietotņu izstrādes pieejas, tiks ņemtas vērā katras pieejas stiprās un vājās pusēs. Turklāt visas pieejas tiks salīdzinātas viena ar otru.

1.1 Dzimtā pieeja

Dzimta, jeb vietēja pieeja nozīmē lietojumprogrammu izstrādi konkrētām mobilajām operētājsistēmām, izmantojot mobilās programmatūras izstrādes komplektus, jeb SDK, rīkus un valodas, kas ir dzimtās konkrētai mobilajai operētājsistēmai. Piemēram, Android lietojumprogrammu izstrāde parasti tiek veikta, izmantojot Android SDK, Java vai Kotlin programmēšanas valodu, kā arī tādu integrētas izstrādes vides, jeb IDE, rīku kā Android Studio. Turpretī iOS lietojumprogrammas tiek izstrādātas, izmantojot iOS SDK, X-Code IDE un Objective C vai Swift programmēšanas valodas. Blackberry lietojumprogrammas var izveidot, izmantojot Blackberry SDK, Momenatics IDE un Java ME valodu [9].

Pēc izstrādāšanas lietojumprogrammu var augšupielādēt lietojumprogrammu veikalā (Android Market, Apple Store vai Blackberry lietojumprogrammu pasaule), lai to plaši izmantotu.

Īsumā ar "dzimto" tas tiek domāts par oriģināliem, katrai platformai paredzētiem instrumentu komplektiem.[2]

Vietējā pieeja ir pārāka par citām pieejām šādos punktos:

- **Bagātīga lietotāju pieredze, jeb UX**

Izmantojot šo mobilās izstrādes pieeju, lietotņu dizaineri un izstrādātāji var koncentrēties uz lietotnes funkcionalitātes pielāgošanu konkrētai platformai ar tās valodu, konkrētiem elementiem un iestatītiem žestiem. Visi šie aspekti izstrādātājiem ļauj sasniegt vienotības sajūtu. Turklāt dzimtas lietojumprogrammas var plaši izmantot arī bezsaistes režīma iespējas.

Sākot no galalietotāja perspektīvas, UX / UI attiecīgajā platformā ir vienoti, tāpēc lietotāji to jau labi zina. UX modeļi atkārtojas citās dzimtas lietotnēs, tāpēc cilvēki intuitīvi zinās, kā pārvietoties pa lietotnēm.

- **Augsta veiktspēja**

Ar dzimtas mobilās lietotnes izstrādi lietotne tiek izveidota un optimizēta konkrētai platformai. Rezultātā lietotne demonstrē ārkārtīgi augstu veiktspējas līmeni. Dzimtas lietotnes ir ļoti ātras un atsaucīgas, jo tās ir izveidotas konkrētai platformai un tiek kompilētas, izmantojot platformu galveno programmēšanas valodu un API. Rezultātā lietotne ir daudz efektīvāka. Ierīcē tiek saglabāta lietotne, kas programmatūrai ļauj maksimāli izmantot ierīces apstrādes ātrumu. Kad lietotāji pārvietojas pa dzimtām mobilajām lietotnēm, saturs un vizuālie elementi jau tiek saglabāti ierīcē, kas nozīmē, ka ielādēs laiks ir raits.

- **Labāka drošība**

Dzimtās lietotnes var pilnībā izmantot ierīces platformas un operētājsistēmas (OS) iebūvētās drošības funkcijas.

- **Pilna funkcionalitāte**

Dzimtam lietotnēm ir pilnīga piekļuve datu bāzēm un ierīces aparatūras funkcijām. Turklāt spraudņi, citas trešo personu programmas vai rīki neapdraudēs to funkcionalitāti.

Vietējām lietotnēm atšķirībā no citām lietotnēm, kas izveidotas, izmantojot citas pieejas, ir piekļuve visām platformas funkcijām.

- **Var palaist bezsaistē**

Dzimtam lietojumprogrammām nav nepieciešams interneta savienojums, lai darbotos. Tā kā tie ir uzstādīti ierīcē, tie var darboties jebkur, ja vien ierīce darbojas pareizi.

Visizplatītākie šīs pieejas trūkumi ir:

- **Augstākas attīstības izmaksas**

Dzimtas lietotnes izstrādes process ir sarežģīts un prasa daudzu izstrādes komandas dalībnieku piedalīšanos.

Lai izstrādātu lietojumprogrammu divām platformām, būs nepieciešamas divas izstrādes komandas.

- **Ilgāks izstrādes laiks**

Dzimtam lietotnēm parasti ir nepieciešams vairāk laika, lai tās pabeigtu un palaistu, galvenokārt, ja mērķis ir vienlaikus palaist dažādu platformu lietotnes. Tā kā dzimta lietotne ir paredzēta noteiktai operētājsistēmai, katras platformas izstrādei ir nepieciešams vairāk laika, jo izstrādātājiem jāizmanto īpaša izstrādes pieeja priekš iOS, Android uc. Tā rezultātā, lai nodrošinātu kvalitatīvu produktu, kopējais izstrādes laiks ir ilgāks.

1.2 Šķērsplatformu pieejas

Šķērsplatformas izstrādes pamatā ir izstrāde ar vienu vidi un izvietošana daudzās platformās [34]. Šī lietojumprogrammu izstrādes pieeja norāda kodu bāzes izveidošanas procesu vienreiz un pēc tam tās apkopošanu dažādām mobilajām operētājsistēmām [16][34]. Izmantojot šo pieeju, vissvarīgākā priekšrocība ir tā, ka izstrādātāji kodu raksta tikai vienu reizi un nav jāatkārto vai jāveicina kods izpildei citās operētājsistēmās, kas savukārt var dot izdevīgu rezultātu laikā un samazināt izmaksas, jo tas ļauj izstrādātājiem rakstīt tikai ar vienu no programmēšanas valodām un izmantot vienotu ietvaru. Šķērsplatformas izstrādi var iedalīt četras pieejās:

- Tīmekļa pieeja;
- Hibrīda izstrādes pieeja;
- Šķērskompilēta pieeja;
- Interpretēta pieeja [16].

1.2.1 Tīmekļa pieeja

Tīmekļa pieeja mobilo lietotņu izstrādē nozīmē tādas lietojumprogrammas izveidi, kas darbotos pārlūkprogrammā, piemēram, Safari, Mozilla Firefox vai Google Chrome, kas nozīmē, ka atšķirībā no "dzimtam" lietotnēm lietojumprogrammai var piekļūt tikai no tīmekļa pārlūka.

Šīs pieejas galvenā priekšrocība ir spēja nodrošināt optimālu lietotāja pieredzi, jeb UX, lietotājiem, kuri izmanto dažādas mobilās operētājsistēmas, kā arī galddatoru vietņu lietotājiem, jo katra vietne tika izstrādāta speciāli attiecīgajam ierīces tipam. Lejupielādes nav vajadzīgas, jo lietotnes tiek veidotas, izmantojot CSS, HTML vai JavaScript valodu.

Tādējādi lietotne nav specifiska iPhone vai Android vai jebkurai citai mobilai operētājsistēmai.

Citiem vārdiem sakot, tas ir būvēts vienreiz un darbojas visur. Jāatzīmē, ka mobilās tīmekļa lietotnes ir gluži kā parasta mobilā vietne. Tīmekļa lietotnes lietotāja interfeiss izskatās kā dzimtas lietotnes interfeiss, taču tiek izmantotas tīmekļa tehnoloģijas.

Tīmekļa pieejai ir sekojošas priekšrocības:

- **Saderība**.

Vietne uzlabo lietotāju pieredzi dažādu veidu mobilajās ierīcēs. Turpretī dzimtajai mobilajai lietojumprogrammai ir jāizstrādā atsevišķa versija katram operētājsistēmas un ierīces tipam. Lietotāji, kuriem pieder dažāda veida ierīces, var īpaši novērtēt saderības priekšrocības, kuras nodrošina adaptīvās vietnes, sk. 1.2.1.1. Adaptīva tīmekļa dizains.

Tīmekļa lietotnes var izstrādāt visām platformām, ja vien tās var darboties atbilstošā tīmekļa pārlūkā.

- **Viegli atbalstīt un uzturēt.**

Tīmekļa lietotnes ir salīdzinoši viegli uzturēt, jo tām tiek izmantota kopēja kodu bāze vairākās mobilajās platformās.

Salīdzinot ar mobilajām lietotnēm, kurām nepieciešams lejupielādēt katru atjauninājumu, tīmekļa lietotnes ļauj mainīt saturu vai projektējumu, tikai rediģējot tos vienu reizi, un ļauj to izdarīt efektīvi un elastīgi. Pēc ieviešanas atjauninājumi kļūst aktīvi un uzreiz redzami visu veidu ierīcēs.

Tīmekļa lietotņu atjauninājumiem nav jāiet caur lietotņu veikalu(piem. MarketPlace, vai AppStore), tas nozīmē, ka lietotājam atjauninājumi nav jāpārvalda manuāli. Jaunākā versija vienmēr tiek ielādēta, kad lietotājs atver tīmekļa lietotni.

Tomēr lietojumprogrammām, kas veidotas, izmantojot tīmekļa pieeju, ir ierobežojumi.

- **Sliktāka veiktspēja**

Tīmekļa lietojumprogrammas veiktspēja neatbilst dzimtam lietotnēm. Tie mēdz būt lēnāki un diezgan ierobežoti funkcionalitātē.

- **Aktīvs internets.**

Tīmekļa lietojumprogrammai ir nepieciešams aktīvs internets, lai tā darbotos ierīcēs.

- **Sistēmas resursi.**

Tīmekļa lietojumprogramma nevar izmantot sistēmas piedāvātos resursus (tas var izmantot tikai tik daudz, cik paredzēts pārlūkam), kur dzimtas mobilas lietotnes var izmantot tik daudz, cik tas nepieciešams.

- **Lietotāja saskarne**.

Saistībā ar lietotāja saskarni, izstrādājot tīmekļa lietojumprogrammas, tās ir grūti noformēt, jo jāņem vērā katrs pārlūks un ekrāna lielums. Web lietotnes ir mazāk interaktīvas un intuitīvas salīdzinājumā ar dzimtajam lietotnēm.

Tīmekļa pieeja pati par sevi iedalās divās apakš pieejās:

- adaptīvs tīmekļa dizains;
- progresīvs tīmekļa dizains.

1.2.1.1 Adaptīvs tīmekļa dizains

Adaptīva lietotne ir lietojumprogrammā, kurā tiek piemērots adaptīvs projektējums, jeb dizains. Savukārt adaptīvs tīmekļa projektējums ir paņēmienu kopums, ko izmanto vienas tīmekļa vietnes izveidošanai, kura pielāgojas dažādām ierīcēm un spēj sevi pārveidot atkarībā no dažādiem ekrāna izmēriem, izšķirtspējas un orientācijas no lielākajām ierīcēm, piemēram, interneta TV, līdz mazākajām – mobilajām ierīcēm[18][19].

Adaptīvais tīmekļa dizains pamatā ir trīs galvenās izplatītas metodes, kā sagatavot adaptīvu vietnes dizainu:

- **Elastīgi izkārtojumi.**

Elastīga režģa izmantošana vietnes izkārtojuma izveidošanai, kas dinamiski mainās uz jebkuru platumu.

- **Multivides vaicājumi.**

Plašsaziņas līdzekļu veidu paplašinājums, priekš mērķauditorijas un iekļautu stilu atlasījumām. Multivides vaicājumi ļauj dizaineriem norādīt dažādus stilus konkrētiem pārlūka un ierīces apstākļiem.

- **Elastīgs multivide.**

Padara multividi (attēlus, video un citus formātus) mērogojamu, mainot apdrukājamo materiālu lielumu, mainoties skata porta lielumam.

Lai skaidri izprastu adaptīvas tīmekļa lietojumprogrammas priekšrocības, mēs varam salīdzināt tradicionālo tīmekļa lietojumprogrammu ar adaptīvo tīmekļa lietotni.

Tradicionāla, jeb nē-adaptīva, dizaina lapas piemērs ir lapa, kurā teksts ir labi lasāms darbvirsmas pārlūkprogrammās, bet viedtālruņos tas ir ļoti mazs, nesalasāms, bieži vien tāpēc, ka ir pārāk daudz kolonnu vai attēlu, kas ir pārāk lieli, lai ietilptu viedtālruņa ierobežotajā skata porta displeja platums.

Turpretī ar adaptīvu dizainu tīmekļa izstrādātājiem nav jākoncentrējas uz noteiktiem displeja izmēriem; drīzāk viņu adaptīvais tīmekļa kods ir paredzēts, lai pielāgotos displeja izmēru diapazonam, parādot dažādus izkārtojumus, pamatojoties uz mobilās ierīces vai datora displeja izmēru un iespējām.

1.2.1.2 Progresīvs tīmekļa dizains

Progresīvā tīmekļa lietotne ir mobilo tīmekļa lietojumprogrammu izstrādes metožu kopums, kas ietver tādu lietotņu veidošanu, kuras jūtas un izskatās pēc dzimtām lietotnēm, tomēr tie ir joprojām parastas tīmekļa lapas. Izmantojot tīmekļa kaudzi (JS, HTML un CSS), progresīvās tīmekļa lietotnes apvieno bagātīgu funkcionalitāti un vienmērīgu lietotāja pieredzi, kas saistīta ar dzimtajām lietotnēm [20].

Progresīvās tīmekļa lietotnem piemīt tādas īpašības:

- **Pilnīga atsaucība un pārlūka savietojamība.**

Progresīvā tīmekļa lietotne darbojas ar visiem pārlūkiem un ir saderīgi ar jebkuru ierīci neatkarīgi no ekrāna lieluma un citām specifikācijām. Planšetdatoru un mobilo ierīču lietotājiem būs tāda pati pieredze. Ja nepieciešams, ir iesējams pielāgot lietotni darbvirsmai.

- **Savienojamības neatkarība.**

Progresīvas tīmekļa lietojumprogrammas var darboties gan bezsaistē, gan zemas kvalitātes tīklos.

- **Ātrie ielādēs laiki.**

Neatkarība no interneta savienojumam vietnes var ielādēties milisekundēs laikā.

- **Pašpiegādes paziņojumi, jeb push-notifikācijas.**

Progresīvas tīmekļa lietojumprogrammas var nosūtīt pašpiegādes paziņojumus lietotājiem.[20]

1.2.2 Hibrīda pieeja

Hibrīdas pieeja mobilo lietojumprogrammu izstrādei ir iepriekšējo divu pieeju (dzimtas un tīmekļa pieejas) kombinācija. Hibrīdas lietotnes tiek izstrādātas, izmantojot HTML5, JavaScript un CSS - līdzīgi tīmekļa lietojumprogrammām [13][16]. Tomēr viņiem ir arī dzimts apvalks, kas tiks publicēts Android Marketlace vai AppStore tiešsaistes veikalā, galvenā atšķirība starp hibrīda lietojumprogrammu un tīmekļa lietojumprogrammu ir tā, ka tīmekļa lietojumprogrammu var izmantot tikai ar pārlūku, savukārt hibrīda lietojumprogrammu vispirms ir jāinstalē uz savas ierīces [16]. Lietotāja saskarne tiek izstrādāta, izmantojot HTML5 vai JavaScript, un loģiku nosaka JavaScript.

Tādas lietotnes tiek izstrādātas, izmantojot atvērtā pirmkoda bibliotēkas un izmantojot spraudņus, šīm lietotnēm var būt pilnīga piekļuve mobilās ierīces funkcijām, piemēram,

kamerai, GPS vai failu sistēmai utt. Piekļuves pakāpe ierīces funkcijām nav salīdzināma ar dzimtajām mobilajām lietojumprogrammām, bet gan, tas ir labāk nekā mobilajās tīmekļa lietojumprogrammās. Būtībā hibrīda lietojumprogramma ļauj izstrādātājam apvienot tīmekļa izstrādes ātrumu ar iespēju izmantot ierīces funkcijas, lai nodrošinātu uzlabotu lietotāja pieredzi.

Izmantojot hibrīdas izstrādes pieeju, ir iespējams izveidot lietotni, kas atbalsta bagātinātu multividi. Svarīgi pieminēt arī to, ka hibrīdām lietojumprogrammām ne vienmēr ir nepieciešams interneta savienojums. Kā hibrīda lietojuma piemēru mēs varam ņemt Instagram kas ir bezmaksas fotoattēlu un video koplietošanas lietotne. Lai gan Instagram plūsmu nevar atjaunot, ja nav interneta savienojuma, bet joprojām ir iespējams piekļūt datiem, kas jau ir ielādēti.

Rīks, piemēram, Appcelerator Titanium, tiek izmantots šāda veida lietojumprogrammu izstrādē. Detalizētāks rīka apraksts ir pieejams nodaļā 2.4 Appcelerator Titanium. Salīdzinot ar citām pieejām, šajā pieejā ir tādas priekšrocības:

- Tīmekļa izstrādātāji var izmantot savas prasmes un zināšanas mobilo lietotnēs izstrādei;
- Hibrīda lietotnēm nav nepieciešams tīmekļa pārlūks.
- Hibrīda lietotnēm ir piekļuve ierīces aparatūrai, kā arī tādām ierīces funkcijām kā kamera vai GPS.
- Hibrīdajām lietotnēm ir nepieciešama tikai viena kodu bāze, tāpēc hibrīdās lietotnes ir lētākas nekā dzimtas, un šīs lietotnes tiek izstrādātas daudz ātrāk.

Pieejai piemīt sekojoši ierobežojumi:

- Hibrīdas lietotnes ir daudz lēnākas nekā dzimtas lietotnes.
- Slikta lietotāju pieredze. Tā kā lietojumprogramma tiek veidota vairākām platformām vienlaikus, nav iespējams uzlabot lietotņu saskarni Android lietotājiem, nesabojājot lietotāju pieredzi IOS lietotājiem.
- Hibrīdas lietotnes kodolā ir vietnes, un, lai lietotājiem nodrošinātu visu funkciju klāstu, tām ir nepieciešams pastāvīgs interneta savienojums. Bet lietotne var strādāt arī bezsaistē.
- Hibrīdas lietotnēm ir ierobežota pieeja aparatūras funkcijām, veiktspējas samazināšanās [13].

1.2.3 Interpretēta pieeja

Interpretētu pieeju izmanto, lai izveidotu šķērs-platformu lietojumprogrammas no vienas koda bāzes, un vissvarīgāk, izmantojot kopējo programmēšanas valodu - JavaScript ir viens no ievērojamākajiem piemēriem. Lielākā daļa interpretētās lietojumprogrammas tiek tulkota vietējā kodā, bet pārējā daļa tiek interpretēta izpildlaika laikā [16] [35]. Pretstatā hibrīdajai pieejai, izstrādātāji neveido vietni, bet izmanto JavaScript, lai atveidotu platformas vietējos lietotāja interfeisa komponentus [4] [16]. Divi no rīkiem, kas izmanto šo platformu pieeju, ir NativeScript (detalizētāk aprakstīts 2.5. NativeScript) un ReactNative (detalizētāk aprakstīts 2.3. ReactNative) [4] [35].

1.2.4 Šķērskompilēta pieeja

Šķērskompilētā pieejā mobilās lietotnes izstrādei izmanto kopēju programmēšanas valodu, kas ir līdzīgs interpretētai pieejai [16]. Tomēr šķērskompilētas pieejas gadījumā viss avota kods tiek apkopots tieši dzimtajā kodā, kuru var izpildīt mobilajā platformā. Kā šķērskompilēta rīka piemēru var minēt Xamarin un Flutter [8]. Visievērojamāka šīs pieejas priekšrocība ir tas, ka lietojumprogrammas spēj sasniegt vietējiem lietotnēm līdzīgu veiktspēju un nodrošināt visas vietējo lietojumprogrammu funkcijas kopā ar vietējās saskarnes komponentiem[16].

Hibrīdas mobilās lietojumprogrammas rada arī tādu pašu izvadi un procesu. Bet atšķirība ir tajā, cik jaudīgas un elastīgas ir attīstības metodes, kuras izmanto katrs no viņiem. Hibrīdas attīstības platformas ietver HTML5 un Javascript izmantošanu, kas ir tīmekļa tehnoloģijas.

No otras puses, šķērskompilēta izstrāde ietilp tehnoloģijas, kas nav saistītas ar tīmekļa izmantošanu, piemēram, uz .NET satvara orientētas tehnoloģijas, tādas kā, Xamarin.

Līdz ar to hibrīda mobilā izstrāde attiecas uz tīmekļa tehnoloģiju un valodu izmantošanu, turpretim šķērskompilēta ir iekļauti arī ar tīmekļiem nesaistītas tehnoloģijas vai rīki, kā arī patstāvīgi rīki.

Galvenās īpašības, kas piemīt šķērskompilētājai pieejai ir sekojošas:

- Atkārtoti izmantojami koda komponenti. Tā vietā, lai rakstītu jaunu kodu priekš katrai platformai, var atkārtoti izmantot to pašu kodu vairākās platformās. Atkarībā no rīka līdz 80% no viena un tā paša koda var izmantot dažādām platformām.

- Izstrādes ātrums. Lietotņu izstrāde ir daudz ātrāka nekā lietotnes izstrāde izmantojot dzimtu pieeju , jo tiek izmantota viena koda bāze.
- Līdzīgi dzimtām lietotāja pieredze. Būtībā šķērsplatformu tehnoloģijas galvenais mērķis ir piegādāt vietnēm līdzīgas lietojumprogrammas. Izmantojot dažādus rīkus, izstrādātā lietojumprogramma, kas var šķist diezgan līdzīga vietējām lietotnēm.

Tomēr ir arī sekojoši ierobežojumi:

- Veiktspējas izaicinājumi: daudz platformu lietotnēm ir problēmas ar integrāciju ar mērķa operētājsistēmām. Tas ir saistīts ar nekonsekventu saziņu starp ierīces vietējiem un vietējiem komponentiem. Tas ietekmē lietojumprogrammu optimālu darbību.
- Lēna koda veiktspēja ar ierobežotu rīka pieejamību: Savstarpēja atbilstība izstrādes posmā padara kodu gausu un pat samazina ātrumu. Dažreiz izstrādātājiem kļūst obligāti jāizmanto rīki, kas ir piemēroti tikai konkrētai lietotnei.
- Ierobežota lietotāja pieredze: vairāku platformu lietojumprogrammas nespēj pilnībā izmantot tikai dzimtas iezīmes, lai nodrošinātu izcilu lietotāja pieredzi. Tas ir dažādu ekrāna izkārtojumu, platformu un funkcionalitātes dēļ.

1.3 Pieeju salīdzinājums

Šajā nodaļā tika analizētas visu iepriekšminēto pieeju priekšrocības un trūkumi. Izmantojamās izstrādes pieejas izvēlei ir izšķiroša nozīme ir, kā arī tai ir liela ietekme visā lietojumprogrammas dzīves ciklā.

Hibrīda, šķērskompilētas, interpretētas lietojumprogrammas izstrādes pieejas ir diezgan līdzīgas mobilo platformu pārklājuma ziņā, tomēr galvenās tehnoloģijas, ko izmanto lietotnes izstrādei, atšķiras. Šķērskompilētas mobilās lietotnes tiek izstrādātas, izmantojot starpposma valodu, piemēram, C # vai Dart, kas nav dzimtā ierīces operētājsistēmā, tālāk avotu kods pilnība tiek apkopots dzimtajā valodā. Savukārt, interpretētā pieejā daļa no koda tiek interpretēta izpildes laikā.

Lietojumprogrammas, kas izveidotas, izmantojot hibrīdu pieeju, parasti ir tīmekļa lietotnes, kas ir ievietoti vietējās lietotnes apvalkā, šīs lietotnes var instalēt un palaist tāpat kā jebkuru citu attiecīgās platformas vietējo lietotni. Izstrādes pieeja un galaprodukti ir ļoti līdzīgi tīmekļa lietotnēm. Tāpēc hibrīdās lietotnes parasti cīnās par vietējo lietotņu vienmērīgu un slidenu lietojamību. Var rasties arī platformai raksturīgas problēmas, piemēram, lapu pārejas, kas nedarbojas vienmērīgi operētājsistēmā Android, jo trūkst CSS / CSS3 ieviešanas [3][6].

Zemāk ir sniegta tabula (sk. 1.1.tabula), kurā ir apkopotas katras pieejas galvenās iezīmes.

1.1.tabula

Pieeju salīdzinājums

	Dzimta	**Tīmekļa**	**Hibrīda**	**Šķērs-kompilēta**	**Interpretēta**
Veiktspēja	Labāka iespējama	Zema	Vidēja	Tuvu dzimtajam	Tuvu dzimtajam
Piekļuve ierīces funkcijām	Pilna	Iero-bežota	Atkarīga no spraudņiem	Pilna	Pilna

Kodu atkārtota izmanto-jamība	Nevar izmantot atkārtoti	Viena koda bāze, lietotne darbojas pārlūkā	Kopīgota visas platformas "backend" daļā. Lietotnei nepieciešams vietējais apvalks	Kopīga kodu bāze visām atbals-tītajām platformām	Kopīga kodu bāze visām atbalstītajām platformām
Izstrādes ātrums	Laikietilpīgs	Ātrs	Ātrs	Diezgan ātrs	Diezgan ātri
Programmu izstrādes tehnoloģiju kaudze	IOS: Objectic-C/Swift. Android: Java / Kotlin. Blackberry: Java ME. Windows Phone: C#.	HTML, CSS, JS	JS, CSS, HTML5,	JS, Dart, C#	JS
UI / UX kvalitāte	Izcila	Slikta	Vidēja	Laba	Laba

Nepieciešamība pēc piekļuves internetam	Nav nepieciešama	Nepieciešama	Nepieciešama, taču ielādēto jau saturu var rādīt pat bez savienojuma	Nav nepieciešama	Nav nepieciešama

Mobilo lietojumprogrammu izstrādei ir dažādas pieejas. Bakalaura darba ietvaros bija izpētītas piecas no tām. Tā kā aprakstītie kritēriji 3.1. Izvirzītie kritēriji nodaļā norāda, ka lietojumprogrammai ir jānodrošina labu kopējo veiktspēju un dzimtajam lietotnēm tuvu lietotāja pieredzi un saskarnes dizainu, Tīmekļa un Hibrīda pieejas tika uzskatītās par nepiemērotām, kopumā sliktas lietotnes UI kvalitāte un veiktspēja dēļ. Škērskompilētas un Interpretētas lietojumprogrammās veiktspēja, protams, nav tāda pati kā dzimtam lietojumprogrammām, tomēr, ņemot vērā izpētēs iegūtas rezultātus, tika nolēmts, ka škērskompilētas un interpretētas šķersplatformas piejas risinājumi ir perspektīvākās no visām izpētītām pieejām.

2 RĪKI ŠĶĒRSPLATFORMU IZSTRĀDĒ

Šajā nodaļā tiek apskatīti un salīdzināti vairāki šķērsplatformas izstrādes rīki.

2.1 Xamarin

Xamarin tiek izmantots, lai izveidotu modernas un izpildāmas lietojumprogrammas tādām operētājsistēmām kā: iOS, Android un Windows izmantojot .NET platformu. Vairumā gadījumu 80% lietojumprogrammas koda ir koplietojami [6][22]. Xamarin ļauj izveidot dzimtu lietotāja sakārni katrā platformā, lietotnes izstrādei galvenokārt tiek izmantota C# programmēšanas valoda, kā arī XAML (sk. 2.1.4 Paplašināma Lietojumprogrammu Iezīmēšanas Valoda). Lietotnes izstrādei tiek rekomendēta Visual Studio integrēta izstrādes vide.

Pētījumā laikā tika atklātās vairākas Xamarin ietvara īpašības. Pirmkārt, ietvarā ir pieejamas dažādas pieejas saskarnes dizainām. Otrkārt, Xamarin ir pilnīga piekļuve vietējām API un instrumentu kopām, kuras tiek izmantotas Android, iOS un Windows platformās. Līdz ar to tas var nodrošināt vietēju (vai gandrīz vietējo) dizainu un veiktspēju katram lietojumam. Tomēr Xamarin lietotnēm ir salīdzinoši lielais lietotņu failu izmērs - tie var būt divreiz lielāki nekā vietējie faili, un tiem nepieciešama papildu pielāgošana.
Xamarin rīkam ir divas alternatīvas: Xamarin Native vai Xamarin.Froms.

2.1.1 Xamarin Native

Xamarin.Android un Xamarin.iOS bieži sauc par Xamarin Native vai Xamarin Traditional.

Konsekventa un tuvu dzimtajai lietotāja saskarnes izstrāde ir parastais šķērslis daudzu platformu satvariem. Xamarin Native izstrādes pieeja ir izveidot vienotu kopīgu piekļuvi biznesa loģikai un datu bāzei un pēc tam izmantojo C# valodu izveidot atšķirīgas lietotāja saskarnes Android un iOS platformām. Tas tiek panākts atsevišķi ar Xamarin.iOS un Xamarin.Android, tāpēc UI faktiski netiek koplietots.

2.1.2 Xamarin.Forms

Microsoft definē Xamarin.Forms kā vēl vienu atvērtā koda lietotāja saskarnes satvaru, kas papildina Xamarin. Tas ļauj izstrādātājiem ātri izveidot prototipus un saskarnes izmantojot XAML (ir detalizētāk aprakstīts 2.1.3 paplašināma lietojumprogrammu iezīmēšanas valoda), virs koplietojamas C# kodu bāzes šķērsplatformu lietojumprogrammām[6]. Xamarin.Forms vispārējā funkcija ir līdzīga Xamarin Native, taču galvenā atšķirība ir, ka Xamarin.Froms nodrošinā netikai vienotu koda bazi bizness loģikai, bet UI kodu var arī koplietot vairākām platformām, tāda iespēja savukārt var dramatiski samazināt darba apjomu un paātrināt izstrādes procesu. Turklāt atšķirībā no Xamarin Native, Xamarin.Forms ir liela pakotņu ekosistēma, kas lietotnēm piešķir daudzveidīgu funkcionalitāti[6].

2.1.3 Paplašināma Lietojumprogrammu Iezīmēšanas Valoda

XAML ir paplašināma lietojumprogrammu iezīmēšanas valoda (jeb, eXtensible Application Markup Language), ko Microsoft ir izveidojusi kā alternatīvu programmēšanas kodam objektu izveidošanai un inicializēšanai, kā arī šo objektu sakārtošanai vecāku un bērnu hierarhijās [6]. XAML ir XML balstīta valoda, kas tiek lietota Xamarin.Forms ietvarā. Tas ļauj noteikt lietotāja saskarnes lietojumprogrammās, izmantojot iezīmēšanas valodas, nevis kodu. XAML salīdzinājumā ar līdzvērtīgu kodu ir vairākas priekšrocības, piemēram, kad tiek izmantots XAML ir viegli saprast, kā vadības bloku vizuālais koks ir izkārtots lappusē. Kopumā kods ir arī lasāmāks un kodolīgāks nekā līdzvērtīgs kods [6][31]. Lietojumprogrammu lietotāja saskarnes izstrāde izmantojot XAML ir vienkāršs process. Tas ievērojami samazina C # koda daudzumu, kas bija jāraksta, jo XAML un C # var veikt vienādus pienākumus [31][32].

2.2 Flutter

Flutter ir šķērsplatformu mobilo lietojumprogrammu izstrādes rīks, kas paredzēts augstas veiktspējas un augstas precizitātes lietotņu izstrādei. Visas lietotnes tiek izstrādātas uz vairākām platformām vienlaikus, no viena koda bāzes. Pašlaik Flutter atbalsta IOS un Android mobilās operētājsistēmas [7][17]. Papildus mobilajām lietotnēm Flutter arī atbalsta tīmekļa satura ģenerēšanu, kas tiek nodrošināts, izmantojot uz standartiem balstītas tīmekļa tehnoloģijas: HTML, CSS un JavaScript, tomēr tīmekļa atbalsts joprojām ir beta versijā [7].

Tiek atbalstīti spraudņi priekš Android Studio, IntelliJ IDEA un VS Code integrētajām izstrādes vidēm. Visas lietotnes tiek rakstītas izmantojot Dart programmēšanas valodu (mūsdienīga, kodolīga, objekt orientēta valoda) [17]. Lietotāja saskarnes Flutter ievtarā tiek veidotas, izmantojot logrīkus. Tie definē, kā jāizskatās lietojuma izkārtojumam, ņemot vērā to pašreizējo konfigurāciju un stāvokli [7]. Atšķirībā no citiem ietvariem vai vietējās platformas rīkiem, kas atdala skatus, skatu kontrolierus, izkārtojumus un citas īpašības, Flutter ir konsekvents, vienots objekta modelis – logrīks. Logrīki ir šī ietvara galvenie un vienīgie interfeisa komponenti. Būtībā logrīki pamatā ir celtniecības bloki, konfigurācija, elementu izveidošanai [17]. Šī rīka mērķis ir dot iespēju izstrādātājiem piegādāt augstas veiktspējas lietotnes, kas dažādās platformās jūtas dabiski, kā dzimtas lietotnes. Flutter ir pilnīga vide ar ietvaru un logrīkiem, kas dod iespēju efektīvi rakstīt mobilās lietotnes.
Ar Flutter palīdzību var:

- Izstrādāt lietotni operētājsistēmai iOS un Android no vienas kodēšanas bāzes;
- Darīt vairāk ar mazāk koda rindu skaitu, ar modernu, izteiksmīgu valodu un deklaratīvu pieeju[7];
- Izstrādāt skaistu, ļoti pielāgotu lietotāju pieredzi
- Izmantot bagātīgo Material Component un Cupertine logrīku bibliotēkas, kas ir izveidotas, izmantojot paša Flutter ietvaru
- Realizēt pielāgotus, skaistus, uz zīmolu balstītus dizainus logrīku komplektus [7].
- Flutter ir bezmaksas un atvērts avots,
- Ir balstīts uz Dart - ātru, uz objektu orientētu programmēšanas valodu, kuru pats par sevi ir viegli iemācīties[17];
- Pateicoties bagātīgajiem logrīkiem, Flutter lietotnes izskatās un jūtas lieliski (ir iespējams izveidot savu pielāgoto lietotņu dizainu, kā arī izmantot viegli pieejamus lietotāja interfeisa elementus, ievērojot konkrētu platformu vadlīnijas)[17];
- Flutter arhitektūra ir balstīta uz mūsdienās ļoti populāro reaktīvo programmēšanu[7];
- Tas kļūst par nopietnu konkurentu vietnei React Native, kā arī šķersplatformas lietotņu izstrādes rīks[17].

Flutter atšķiras no vairuma citu mobilo lietotņu izstrādes rīkiem, ar savu augstas veiktspējas renderēšanas dzinēju, kas var izmantot logrīku izveidei. Flutter implementē lielāko daļu savas sistēmas (kompozīciju veidošana, žesti, animācija, ietvars, logrīki utt.) izmantojot Dart valodu. Tā vietā, lai izmantotu dzimtu lietotāja saskarnes komponentus, ko dara piemēram

React Native (sk. 2.3 ReactNative) un Xamarin(sk.s 2.1 Xamarin), tas patiešām "zīmē" lietotāja interfeisu no nulles uz "ekrāna audekla", izmantojot ātru C ++ 2D grafikas bibliotēku - Skia (kas arī kalpo kā Google grafikas dzinējs).

2.3 ReactNative

React Native ir satvars reālu, dzimto mobilo lietojumprogrammu rakstīšanai. Tas ir balstīts uz ReactJS [13 - 9lpp] - JavaScript bibliotēku lietotāja saskarņu veidošanai, bet tā vietā, lai mērķētu uz pārlūku, tā ir paredzēta mobilajām platformām. ReactNative izmanto JavaScript skriptu sastādīšanas valodu – JSX. Tā apzīmē JavaScript XML, tas ir JavaScript sintakse pagarinājums, kas izskatās līdzīgi XML [15]. JSX aprakstīta, kā jāizskatās lietotāja saskarnei [14]. Priekšapstrādātāji izmanto sintakse, lai XML līdzīgu tekstu, kas atrodams JavaScript failos, pārveidotu par standartiem JavaScript objektiem, kurus JavaScript dzinējs parsēs.

Kaut arī lietojumprogrammas bāze ir rakstīta JS, tā tiek apkopota "dzimtā" kodā, līdz ar to tā uzlabo lietojumprogrammas vispārējo veiktspēju. Aptuveni 70-90% no koda varbūt koplietoti starp platformām [13].Pašlaik ReactNative atbalsta divas mobilās platformas: IOS un Android [11] [13]. Visual studio ir rekomendēta intgrēta izstrādes vide.

ReactNative piemīt sekojošas priekšrocības:

- Karstā pārlādēšana. karstā pārlādēšana ir ReactNative funkcija, kas ļauj saņemt gandrīz tūlītēju atsauksmi par izmaiņām React komponentos. Ja ātra atsvaidzināšana ir iespējota lielākajai daļai labojumu vajadzētu būt redzamiem sekundes vai divu laikā.
- Ātrāka izstrāde. Satvars nodrošina daudzus lietošanai gatavus komponentus, kas var paātrināt izstrādes procesu.
- ReactNative piedāvā patiesi dzimtu lietotāju pieredzi atšķirībā no hibrīda pieejas rīkiem, kas tikai nodrošina dzimta stila iesaiņojumu pārlūka lietotnēm. Tā kā ReactNative izmanto dzimtus komponentus, galaprodukti izskatās un jūtas ļoti reāli - it kā būtu radīti ar dzimto tehnoloģiju. React Native faktiski pārveido marķējumu reālos, dzimtas lietotāja interfeisa elementos. [11]
- Izmantojot dzimtus moduļus, ReactNative uzlabo veiktspēju.

  ReactNative mijiedarbojas ar iOS vai Android paredzētajiem (dzimtiem) komponentiem un tieši un neatkarīgi piešķir kodu dzimtajām API. To darot, tas izmanto atsevišķu pavedienu no lietotāja interfeisa, kā rezultātā tiek palielināta veiktspēja. Galvenais faktors šeit ir ReactNative, izmanto dzimtās API. Ir arī citas opcijas lietotāja interfeisa izveidošanai, piemēram, WebView (atveido tīmekļa saturu dzimtājā skatā).

WebView izmantošana koda atveidošanai, var būt arī citas priekšrocības, taču tās ietekmē veiktspēju.[11]

2.4 Appcelerator Titanium

Appcelerator Titanium ir atvērtā pirmkoda lietojumprogrammu izstrādes rīks, kas izveidots, lai radītu šķērsplatformu "dzimtas" mobilās lietojumprogrammas, izmantojot jaudīgus dzimtus komponentus no JavaScript koda. Tas ietver atvērtā koda SDK ar vairāk nekā 5000 ierīces un mobilās operētājsistēmas API un Studio - jaudīgu Eclipse balstītu IDE. [10]

Titanium nodrošina iespēju veidot mobilās lietotnes, izmantojot JavaScript, un apkopojot tās dzimtājām lietotnēm vairākām mobilajām operētājsistēmām: IOS, Android un BlackBerry [10] [12]. Bija atbalstīts arī Windows Phone SDK, tomēr saskaņā ar oficiālo Titanium dokumentāciju tas ir novecojis [10]. Var atzīmēt, ka dzimtās mobilās lietotnes tiek veidotas, izmantojot tikai JavaScript, bet ir svarīgi pieminēt, ka citas tīmekļa tehnoloģijas, piemēram, CSS3 un HTML5, netiek izmantotas vispār [12][23]. Titanium ir iespējams izmantot gan interpretējamai mobilo lietotņu izstrādei, gan hibrīda lietojumprogrammu izstrādei, otrās pieejas gadījumā tiek izmantoti HTML5 un CSS. Aptuveni 60% - 90% no koda var izmantot atkārtoti, kad tiek atbalstītas vairākas platformas [10].

Lai piekļūtu vietējai funkcijai, Titanium izmanto Hyperloop. Hyperloop ļauj lietojumprogrammai "runāt" no Javascript kodā tieši ar Objective-C / Swift vai Java, nerakstot iesaiņojuma moduļus.

Galvenās Titanium SDK īpašības ir šādas:

- Šķērsplatformas API piekļuvei dzimtiem UI komponentiem, piemēram, navigācijas joslām, izvēlnēm un dialoglodziņiem, kā arī dzimtās ierīces funkcionalitātei, ieskaitot failu sistēmu, tīklu, ģeogrāfisko atrašanās vietu, akselerometru un kartes.
- Caurspīdīga pieeja vietējai funkcionalitātei, ko nodrošina Hyperloop un dzimtie moduļi.
- Web izstrādātāji var imantot savas tīmekļa izstrādes prasmes, lai ātri izveidotu bagātīgas, pilnībā dzimtas mobilās lietotnes un izmantotu jebkuru dzimtās platformas API.
- Atbalsta vairāk platformu - ne tikai IOS un Android, bet arī Blackberry un Windows.
- Appcelerotar Titanium nav Hibrīdas rīsinājum

2.5 NativeScript

NativeScript ir šķērsplatformu JavaScript izstrādes rīks, kas ļauj veidot dzimtās iOS un Android lietotnes no vienas koda bāzes[29]. Rīks nodrošina JavaScript piekļuvi dzimtajām API, lietotāja saskarnei un iOS un Android renderēšanas motoriem. Parasti lietotāja saskarnes dizains tiek izstrādāts un saglabāts XML failos, tomēr veidošana tiek veikta, izmantojot CSS, un biznesa loģika tiek izstrādāta un saglabāta Java vai TypeScript failos.

Izmantojot NativeScript, varat izveidot vienu projektu, kas iebūvēts iOS vai Android lietotnē ar pilnīgi dzimtu lietotāja pieredzi. NativeScript atbalsta integrāciju ar Angular (lietotņu dizaina ietvars un attīstības platforma efektīvu un sarežģītu vienas lapas lietotņu izstrādei), Vue.js (arī JavaScript ietvars lietotāja saskarņu un vienas lappuses lietojumprogrammu veidošanai), kā arī NativeScript nodrošina bagātīgu atbalstu, mūsdienīgu JavaScript, TypeScript, CSS, Flexbox un citas Web prasmes.

Visual Studio kods ir ieteicamais IDE, NativeScript nodrošina paplašinājumu VSCode, kurā ietilpst šis paplašinājums: interaktīva atkļūdošana un integrācija ar ierīces emulatoriem, NativeScript paplašinājums Visual Studio Code nodrošina vispilnīgāko funkciju NativeScript izstrādei. Iespējams, ka tā ir labākā īpašība bez maksas.

Galvenās atšķirības starp React Native un NativeScript ietvariem ir tādas, ka, izmantojot NativeScript, ir iespējams izveidot lietotnes, kuru pamatā ir JavaScript, AngularJS un TypeScript, savukārt ReactNative izmanto tikai JS.

2.6 Rīku salīdzinājums

Šajā nodaļā ir sniegts visu iepriekšminēto rīku salīdzinājums. Visi rīki ir salīdzināti pēc viena un tā paša kritēriju kopuma. Izstrādes rīku salīdzinājums izstrādātājiem var palīdzēt saprast, kurš rīks varētu būt piemērotāks definētajiem mērķiem. Lielākā rīku īpašību daļa bija pieminēta iepriekšējās nodaļās.

Kā viens no svarīgākajiem kritērijiem šķērs-platoformas izstrādē, tika aztīts rīka mobilās operētājsistēmas pārklājums, sk. 2.1. Tabulu. Mobilās operētājsistēmas pārklājums, jo visa šķērs-platformas pieejas pamatā ir iespēja rakstīt lietojumprogrammu vairākām mobilajām platformām ar vienu koda bāzi

Zemāk redzamajā tabulā (sk. 2.1. tabula) var novērot, ka ar jebkuru no aprakstītajiem rīkiem ir iespējams izveidot lietojumprogrammu gan, Android, gan IOS platformām no vienas

koda bāzes. Turklāt, vecāki rīki - Xamarin un Appcelerator Titanium, nodrošina atbalstu arī citām mobilajām operētājsistēmām. Xamarin atbalsta Windows Phone un Titanium BlackBerry.

2.1 tabula

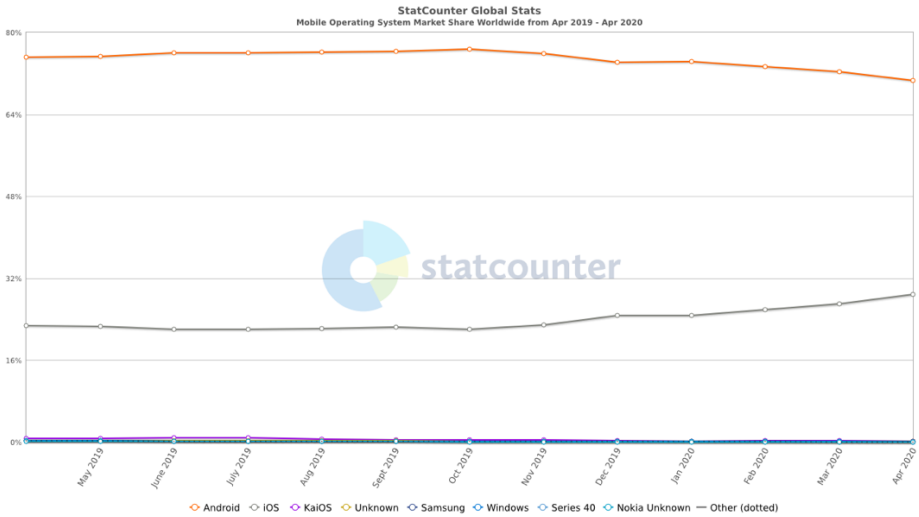
Mobilās operētājsistēmas pārklājums

	Xamarin 2011	**Flutter 2017**	**ReactNative 2015**	**Appcelerator Titanium 2008**	**NativeScript 2014**
Android	X	X	X	X	X
IOS	X	X	X	X	X
Windows Phone	X				
BlackBerry				X	

Zemāk ir pievienots attēls no StatCounter (sk. 2.1 att. Mobilās operētājsistēmas tirgus daļa visā pasaulē[23]), kas ir tīmekļa trafika analīzes vietne. Attēlā ir sniegti dati par tirgus daļu, kas katrai mobilajai OS pieder visā pasaulē. Dati attiecas uz laikposmu no 2019 gadā aprīļa līdz 2020. gada aprīlim. Līniju grafikā tiek piemīnētas dažas mobilās operētājsistēmas, starp tiem arī ir Android un IOS.

Var novērot, ka pašā Augšā atrodas Android OS un zem tās ir redzamā IOS, atšķirība starp abām ir diezgan ievērojama, taču ir svarīgi atzīmēt, ka saskaņā ar šīs līnijas diagrammu uz 2020.gadā aprīlī Android (70.68%) un IOS (28.79%) kopā aizņem nedaudz vairāk par 99% no tirgus daļas [23].

Šie dati būtībā novērš atbalstu Blackberry OS, Windows Phone vai jebkuras citas mobilās operētājsistēmas kā priekšrocību. Turklāt ir paredzēts, ka BlackBerry OS atbalsts beigsies līdz 2020. gada beigām.



2.1 att. **Mobilās operētājsistēmas tirgus daļa visā pasaulē[23]**

Tabulā 2.2. Programmēšanas Valodas, kā tas redzams 3 no 5 rīkiem, tiek izmantota JavaScript valoda, Xamarin lieto C # un Flutter izmanto samērā jaunu programmēšanas valodu Dart , kas atšķirībā no C # vai JavaScript vēl nav īsti labi zināms.

Gan JS, gan C # pirmo reizi parādījās aptuveni pirms 20 gadiem, un šobrīd tās ir stabilas un nobriedušas valodas. Šīm valodām izdevās izveidot spēcīgu kopienu, jo dēļ tā internetā ir pieejams daudz resursu, tiešsaistē ir pieejami daudzi ietvari un bibliotēkas, kā rezultātā izstrādātāji var izmantot esošo kodu ātrākai lietotņu izstrādei, kas kopumā atvieglo izstrādes procesu. Gluži pretēji, Dartā ir ļoti maza izstrādātāju kopiena, tāpēc ir grūtāk saņemt palīdzību jautājumos. Dart ir drošs tipam un var tikt sastādīts gan ar AOT, gan JIT kompilatoriem. Savukārt C# darbojas virs .NET ietvara, kas nodrošina daudzas bibliotēkas, kurās ir klases, kas tiek izmantotas tādiem kopīgiem uzdevumiem kā pieslēgšanās internetam, loga parādīšana vai failu rediģēšana. Atšķirībā no JavaScript vai daudzām citām valodām, nav jāizvēlas no liela skaita bibliotēku, katram mazam uzdevumam. JavaScript ir diezgan liels bibliotēku skaits, dažas no tām ir sliktas kvalitātes.

2.2 tabula

Programmēšanas valodas

	Xamarin	**Flutter**	**ReactNative**	**Appcelerator Titanium**	**NativeScript**
C#	X				
Dart		X			
JavaScript			X	X	X

Kā redzams tabulā 2.3 Lietotāja Saskarnes Izstrāde, visi rīki, izņemot Xamarin Native, atbalsta lietotāja saskarnes izstrādi no vienotās kodu bāzes. Xamarin Native gadījumā būs nepieciešams izstrādātājs, kam ir zināšanas vietējās lietotāja saskarnes skatos, kuri ir raksturīgi visām esošajām platformām. Xamrin nav pieejams lietotāja saskarnes veidotāja m kas atsevišķiem lietotāja interfeisa elementiem nodrošina "vilkt un nomest" funkcionalitāti. Pašlaik ne Flutter, ne ReactNative vai NativeScript nenodrošina "vilkt un nomest" rīku lietotāja saskarņu izveidošanai, tomēr vairāki trešo pušu rīki nodrošina atbalstu šiem trim ietvariem. Vienīgais rīks, kas nodrošina UI veidotāju, ir Titanium, taču "vilkt un nomest" UI veidotājs nav bezmaksas.

Jāatzīmē arī iezīmēšanas valodu pieejamību, kā parādīts tabulā zemāk, katram rīkam, izņemot Xamarin Native un Flutter, ir savas iezīmēšanas valodas, kuras ir vai nu XML, vai balstītas uz XML. Kā jau tika minēts iepriekš, Xamarin Native neļauj koplietot UI kodu, tādēļ iezīmēšanas valodās nav nepieciešama, izkārtojums parasti tiek veidots, izmantojot vietējos rīkus. Flutter gadījumā dizains tiek veikts ar logrīku, jeb Widget palīdzību. Logrīki ir klases, ko izmanto, lai izveidotu lietotāja saskarnes. Tos izmanto gan izkārtojumiem, gan UI elementiem.

2.3 tabula

Lietotāja saskarnes izstrāde

	Xamarin		**Flutter**	**React Native**	**Titanium**	**NativeScript**
	Xamarin Native	***Xamarin .Forms***				
UI koplietošana	Nav	Ir	Ir	Ir	Ir	Ir
Iezīmešanas valodas	Nav	XAML	Nav	JSX	XML un tss stilam	Uz XML balstīts marķējums
UI Veidotājs	Nav	Nav	Trešo pušu rīki	Trešo pušu rīki	"App Designer" rīks	Trešo pušu rīki

Visi minētie rīki atbalsta piekļuvi dzimtajam API, tas katram rīkam ļauj piekļūt platformas specifiskajām funkcijām vai funkcijām, kuras, pagaidām nav pieejamas šķērsplatformās API. Ikviens rīks, izņemot Appcelerator Titanium un Xamarin.Native, atbalsta karstā pārlādēšanas funkcijas. Karstā pārlādēšana atsvaidzina atjauninātos failus, nezaudējot lietotnes stāvokli, un tas būtībā novērš nepieciešamību sastādīt lietojumprogrammu katru reizi, kad tiek veiktas izmaiņās lietojumprogrammā, tādēļ tas izstrādāšanas procesā ietaupa daudz laika. Turklāt ReactNative un NativeScript nodrošina arī tiešās pārlādēšanas funkciju. ReactNative ietvarā tieša pārladēšana saucās - Live reload, savukārt NativeScript ietvarā tā saucās - LiveSync. Tiešā pārlādēšana atjauno visu lietotni, kad mainītais fails tiek saglabāts, tas notiek automātiski un nekādas papildu darbības nav nepieciešamās.

2.4. tabula

Papildus ietvaru īpašības

	Xamarin	**Flutter**	**ReactNative**	**Appcelerator Titanium**	**NativeScript**
Piekļuve dzimtajam API	Ir (izmantojot Xamarin.Essentials)	Ir	Ir	Ir (izmantojot Hyperloop)	Ir
Kārsta pārladēšana	Only for Xamarin.Forms	Ir Hot Reload	Ir LiveReload	Nav	Ir

Gandrīz visi mīnētie rīki ir pilnīgi brīvi izmantojami. Tomēr, lai piekļūtu visām Titanium piedāvātajām funkcijām, ir jāiegādājas abonēšanas plāns, kas saskaņā ar Appcelerator Titanium oficiālo tīmekļa vietni maksā 199 dolāru [10].

Tādiem rīkiem kā ReactNative, Flutter, NativeScript nav nodrošināta noteikta IDE, tomēr tiek piedāvāti spraudņi citām IDE. Piemēram, VSCode, kas arī tika izmantots bakalaura darba praktiskās daļas ietvaros, atbalsta vairāku minēto rīku - Flutter, ReactNative, NativeScript, Titanium - spraudņus.

Sakaņā ar "NativeScript un React Native ietvara analīze un eksperimenti" pētījumu, kuru veice veikts Bc. Dominiks Veselijs, NativeScript lietotņu izveides laiks ir ievērojami lēnāks nekā ReactNative lietotņu, turklāt lietotņu lielums, kas izveidots, izmantojot NativeScript, parasti ir daudz lielāks nekā ReactNative lietotnes izmērs, dažos gadījumos divreiz lielāks[39].

3 IZSTRĀDĀTĀS LIETOTNES

Iepriekšējā nodaļā tika aprakstīts rīku salīdzinājums, pamatojoties uz šo salīdzinājumu tika izvēlēti divi no pieciem rīkiem turpmākam, dziļākam salīdzinājumam. Šajā nodaļā tiek aprakstīti izvēlētie rīki un šķērsplatformu rīku salīdzināšanas vērtēšanas kritēriji. Vēlāk, atsaucoties uz minētajiem kritērijiem, tika izstrādātas lietotnes

3.1 Izvēlētie rīki

Balstoties uz 2.6 Rīku Salīdzinājums bija izvelēti divi rīki, tie ir Flutter un ReactNative. Gan Flutter, gan ReactNative ir salīdzinoši jaunas tehnoloģijas, tās ir atvērtā pirmkoda un brīvi lietojamas. Atšķirība no Appcelerator Titanium ietvara, ReactNative un Flutter ir pilnība bezmaksas ietvari, kā arī tiek nodrošināta karstā pārlādēšanas funkcija. Tiek apgalvots, ka abi rīki: Flutter un React Native izceļas no konkurentiem, pateicoties ātrai lietotnes izstrādei, vietējai veiktspējai un skaistajai lietotāja saskarnei. Flutter katru skata komponentu atveido, izmantojot savu augstas veiktspējas renderēšanas dzinēju. Šī īpašība dod iespēju veidot lietojumprogrammas, kas ir tikpat augstas veiktspējas kā vietējās lietojumprogrammas. Lietotājprogrammas ieviešanas veids ir diezgan unikāls, katrs lietotnes komponents ir logrīks, un logrīki ir sakārtoti vecāku un bērnu logrīku kokā. Ir arī daudzas bibliotēkas ar gatavām funkcijam, kuras nodrošina arī google. Savukārt lietotāja ReactNative ietvarā tiek izmantotas tīmekļa tehnoloģijas, kas atvieglo izstrādes procesu izstrādātajiem, kuriem ir iepriekšējas tīmekļa tehnoloģijās zināšanas.

3.2 Izvirzītie kritēriji

Pētījumā laikā man izdevās atrast 2020. gada grāmatu “Seriously good Software”[21]. Šajā grāmatā aprakstīti programmatūras funkcionālās un nefunkcionālās kvalitātes novērtēšanas veidi. Lai pareizi novērtētu katru izvēlēto starp platformu attīstības rīku, ir noteikts kritēriju kopums. Balstoties uz šo grāmatu un citiem avotiem bija izvirzīti daži analīzes kritēriji.

1. Veiktspēja;

Veiktspēja ir viena no vissvarīgākajām lietojuma īpašībām, kas attiecas uz jebkuru programmatūru. Saskaņā ar "Sistēmas reakcijas laiku un lietotāju apmierinātību: pārlūkprogrammu balstītu lietojumprogrammu eksperimentāls pētījums" pastāv tieša korelācija starp lietojumprogrammas veiktspēju un lietotāju apmierinātību [21][25]. Veiktspēja ir viens no lietotņu kvalitātes raksturlielumiem, kas ir saistīts ar tā izturēšanos, iedzīvojot noteiktus slodzes lielumus un citas situācijas. Vai lietotne ielādējas lēnāk vai avarē kādā konkrētā gadījumā reizi. Labi darvojošā lietotne ir tāda, kas lietotājam ļauj veikt doto uzdevumu bez liekas uztveramās kavēšanās vai kairinājuma. Ja mobilās lietotnes veiktspēja neatbilst lietotāju gaidām, pastāv ļoti liela iespēja, ka viņi atteiksies no lietotnes vispār, kas savukārt radīs ieņēmumu zaudējumus. Veiktspēja ir atkarīga no vairākiem faktoriem: servera, mobilās ierīces, tīkla un pašas lietotnes programmēšanas veida [21]. Tā kā tīkla savienojums vai servera savienojums nav tieši saistīts ar šo darbu, lietojumprogrammu analīzē tie netiks ņemti vērā.

2. Lietotajā saskarne;

Otrais izvēlētais kritērijs bija Lietotāja Saskarne, kā jau bija minēts iepriekš, lietotāja saskarnes ir arī viena no būtiskajām īpašībām mobilajā lietojumprogrammā. Ir svarīgi būt konsekventiem lietojumu projektēšanā un dizaina modeļos. Izstrādājot lietojumprogrammas lietotāja saskarni, ir svarīgi ņemt vērā vietējas (Android un IOS) lietotnes dizainu, dizaina modeļus un mijiedarbību starp komponentiem, lai izstrādātu lietotni, kas atbildēs katras platformas lietotāja dizaina un mijiedarbības stratēģiju pazīmēm. To uzsver dizaina ieteikumi lietotņu izstrādātājiem, lai ievērotu platformas konsekvenci [24][26]. Kad lietotne ir izstrādāta, ievērojot platformas standartus un parastās mijiedarbības stratēģijas, lietotāji var intuitīvi izmantot lietojumprogrammu, balstoties uz viņu iepriekšējo lietotņu lietošanas pieredzi, tādējādi uzlabojot lietojumprogrammas mācīšanās līkni un vispārējo apmierinātību. Dizains nav intuitīvs, ja netiek ievēroti piemērošanas konvencijas, tas lietotājiem rada neapmierinātību, jo lietotājiem ir tendence tērēt vairāk laika, lai veiktu dažas pamata darbības lietojumprogrammā [26].

3. Dokumentācija;

Dokumentācija - dokumentācija ir pirmais atskaites punkts izstrādātājam, un ir svarīgi, lai dokumentācija būtu detalizēta un ļoti labi strukturēta, jo tā galvenokārt ietekmē izstrādes procesu un tam patērēto laiku.

4. Lietotnes izmērs;

Mobilās lietotnes lielums ir raksturlielums, kas jāņem vērā, izstrādājot lietojumprogrammu, ja lietotne ir pārāk liela, pirmkārt, lietotāja lejupielādēšanai tas prasīs

daudz laika, un tas patērēs lielu interneta trafiku, kā arī tas aizņems vairāk ierīču glabāšanas vietas, kā rezultātā lietotājs nāksies izdzēst šo lietojumprogrammu.

5. Karsta pārlādēšanas, jeb Hot Reload funkcija.

Karstā pārlādēšanas funkcija ir faktors, kas ir stingri saistīts ar izstrādes procesu. Tas gan atvieglo, gan ietaupa papildu laiku izstrādes procesā, jo izstrādātājs gandrīz acumirklī var redzēt nelielas izmaiņas lietojumprogrammās, tā vietā, lai veltītu papildu laiku programmas sastādīšanai un apkopošanai.

4 PRAKTISKĀ DAĻA

Darba ietvaros tika izstrādātas vairākas lietotnes, lai salīdzinātu Flutter un ReactNative šķērsplatformas izstrādes rīkus atbilstoši izvirzītiem kritērijiem (sk.3.1 Izvirzītie kritēriji).

Pārbaudes un salīdzināšanas veicināšanai, tika izstrādātas 4 nelielas vienas lapas lietojumprogrammas izmantojot: ReactNative, Flutter, Android SDK un IOS SDK. Vēlāk visas lietotnes tika salīdzinātas, lai noteiktu, kurš no izvēlētajiem diviem šķērsplatformu izstrādes rīkiem nodrošina labāku veiktspēju, kā arī lai noteiktu, cik tuvu šķērsplatformu lietojumprogrammas veiktspēja ir vietējām lietojumprogrammām.

Lai notestētu veiktspēju, tika izvēlēta tieša stratēģija, tika ieviesta "MovieList" lietotne, šīs lietojumprogrammas mērķis ir sniegt lietotājam filmu sarakstu un filmas nosaukumu. Lietotne sastāv no viena ekrāna ar ritināmu fiktīvu filmu sarakstu. Katra saraksta rinda sastāvēja no:

- Filmas attēla;
- Filmas nosaukuma.

Android lietojumprogramma tika izveidota, izmantojot Java programmēšanas valodu un ieteicamo IDE - Android studio. Android izstrādātāji var implementēt sarakstu vairākos veidos, kas galvenokārt ir atkarīgs no tā, kas jādara. Populārākās izmantotās metodes ir ListView vai RecyclerView. Šī darba ietvaros tika izvēlēts RecyclerView, Android dokumentācija definē RecyclerView kā: "Elastīgs skats ierobežota loga nodrošināšanai lielā datu kopā" [27]. RecyclerView pamatā ir ListView uzlabojums. RecyclerView izmanto pārstrādes modeli, kas nozīmē, ka tas rada tikai nepieciešamo skatu turētāju(jeb view holder)

daudzumu un atkārtoti tos izmanto kā lietotāja ritinājumus. ViewHolder apraksta vienuma skatu un metadatus par tā vietu RecyclerView[27]. Skatu turetājs izveido objektu katrma saraksta elementam, objekts savukārt glabā saites uz atsevišķu skatu elementa iekšpusē.

Tā vietā, lai izveidotu katra saraksta vienuma skata turētāju, tas rada tikai tādu daudzumu, kas attēlo ekrānā redzamos vienumus, tāpēc ātra un vienmērīga ritināšana un saraksta izveidošana neaizņem daudz laika.

IOS lietojumprogrammas gadījumā tika izmantota Swift valoda un ieteicamo IDE - XCode. Un saraksta ieviešanai ir izvēlēta UITableView. UITableView ir komponents, kas parāda vienumu sarakstu, un tā šūnu var pārstrādāt atmiņas efektivitātes nolūkā [38]. Tomēr to var izmantot tikai vertikāliem sarakstiem. Apple izstrādātāju dokumentācijā UITableView tiek

definēts šādi: “UITableView pārvalda tabulas pamata izskatu, bet lietotne nodrošina šūnas (UITableViewCell objekti), kas parāda faktisko saturu”[38].

Lai izstrādātu lietotnes ar Flutter un ReactNative rīku, tika izmantota VS Code integrēta izstrādes vide, kas atblasta šo ietvaru spraudņus. Tā kā vietējā lietojumprogrammā tika izvēlēts veiktspējīgs saraksta ieviešanas veids, attiecīgais veids bija izvēlēts arī Flutter un ReactNative. ReactNative gadījumā FlatList komponente tika izvēlēta. Saskaņā ar ReactNative dokumentāciju FlatList ir izpildāmā saskarne vienkāršu sarakstu izveidošanai, kas atbalsta visērtākās funkcijas. Tas nodrošina arī izvēles horizontālo skatu. Šis lietotāja interfeisa komponents ir ReactNative ekvivalents Android RecyclerView komponentām. Lai taupītu atmiņu, blīvam sarakstam ar daudziem vienumiem, FlatList uzreiz veido tiekai noteiktus vienumus, kas jau ir redzami ekranā. Atsevišķos gadījumos, piemēram, ritinot ar lielu ātrumu, tukšie bloki tiks parādīti kā vietturi. FlatList komponets nenodrošina konkrētu saraksta dizainu, tomēr tas palīdz efektīvi padarīt datu avotu, tas iegulst ritināšanas iespējas sarakstos, tas atbalsta tikai vienu datu dimensiju, tomēr šī testa ietvaros tas ir piemērots testēšanas mērķim [28].

Flutter bija divas alternatīvas saraksta izvedei, ListView un ListView.builder(). Flutter lietotnē saraksts bija implementēts izmantojot ListView.builder(). Tomēr saskaņā ar Flutter dokumentāciju standarta ListView konstruktors labi darbojas maziem sarakstiem, lai strādātu ar sarakstiem, kuros ir liels vienumu skaits, vislabāk ir izmantot konstruktoru ListView.builder(). Noklusējuma saraksta ListView konstruktors, “prasa” izveidot visus vienumus vienlaikus, gluži pretēji, ListView.builder () konstruktors izveido vienumus, kad tie tiek ritināti un kļūst redzami uz ekrāna, tādēļ saraksta ieviešanai bija izvēlēts ListView.builder() [21].

Pārbaudes nolūkos tika izveidoti pilnībā automatizēti testēšanas skripti, šie testēšanas skripti ir atkarīgi no platformas, tādējādi viens skripts darbojas tikai tām lietojumprogrammām, kuras tiek darbinātas Android ierīcē (sk. 4.1 att. Automatizēta skripta fragments. Android), bet otrs - IOS ierīcē (sk. 4.2 att. Automatizēta skripta fragments. IOS.). Skripti tika izveidoti, izmantojot Appium, kas ir atvērtā avota testa automatizācijas sistēma, kas paredzēta lietošanai ar vietējām, hibrīdām un mobilajām tīmekļa lietotnēm. Tas virza iOS, Android un Windows lietotnes, izmantojot WebDriver protokolu. Appium atbalsta dažādas programmēšanas valodas. Skriptus ir iespējams rakstīt rakstīt tādās valodās kā: Ruby, Python, Java, JavaScript, C# un citās [37]. Šī darba ietvaros skriptu izveidei tika izvēlēta Java programmēšanas valoda.

Skripta izmantošana visām lietotnēm nodrošināja precīzus testēšanas rezultātus, jo visi saraksti tika ritināti vienādā tempā uz Android ierīcēs un vienāda tempā uz IOS ierīcēs. Tas ļāva tieši salīdzināt testa rezultātus.

Skripti izpildīja sekojošas darbības:

1. Atvērt lietotni;
2. Gludi ritināt līdz pēdējam saraksta elementam;
3. Iziet no lietojumprogrammas.

Visi Android platformai paredzētie lietojumprogrammu testi tika veikti uz Android Emulator Pixel XL API 27, kas ir iekļauts Android Studio IDE, savukart testi IOS platformai tika veikti uz IPhone 11 13.3 simulatora, kas ir iekļauts XCode.

Lai pārbaudītu katra izstrādātā lietojumprogrammas veiktspēju, tika izmantots XCode un Android Studio iebūvēti profilētāji. Pārbaudes laikā tika izmantots saraksts, kas sastāvēja no simts elementiem. Lai iegūtu pareizu rezultātu, profilēšanas rīks tika palaists vismaz piecas reizēs un pēc tam tika aprēķināts vidējais skaits, skripta izpildes laikā laika profilētājs savāca datus. Tālāk savāktie dati tika apkopoti un bija aprēķināts vidējais programmas izpildēs laiks.

```
public static void scrollTillView(String Text){
        ((AndroidDriver<MobileElement>)MobileDriver.getDriver())
        .findElementByAndroidUIAutomator(
            "new UiScrollable(new UiSelector()"
         + ".scrollable(true).instance(0))"
         + ".scrollIntoView(new UiSelector()"
         + ".text(\"" + Text + "\").instance(0))")
        .click();
        }
```

4.1 att. **Automatizēta skripta fragments. Android**

```
public static void scrollTillViewIOS(String Text){
        JavascriptExecutor js=(JavascriptExecutor) MobileDriver.getDriver();
        HashMap scrollObject = new HashMap();
        scrollObject.put("predicateString", "value == '" + Text + "'");
        scrollObject.put("direction", "down");
        js.executeScript("mobile: scroll", scrollObject);

        Map<String, Object> args = new HashMap<String, Object>();
        args.put("timeout", 60 * 10000);
        args.put("profileName", "Time Profiler");
        MobileDriver2.getDriver().executeScript("mobile: startPerfRecord", args);
    }
```

4.2. att. **Automatizēta skripta fragments. IOS**

Turklāt lietotāja saskarnes dizaina salīdzināšanai bija apskatīti vairāki UI komponenti, piemēram IOS lietotnēm tika apskatīti tādi dzimti komponēti, kā UIDatePicker - vietējais IOS komponents, ko izmanto datuma un laika vērtību ievadīšana, Segment Controller - ir lineārs segmentu kopums, katrs no segmentiem darbojas kā savstarpēji izslēdzoša poga. Bieži tiek izmantots dažādu skatu parādīšanai [37]. Android lietotnēm tika apskatīti tādi komponēti, kā DatePicker - nodrošina datuma izvēles logrīku, ar vairākiem režīmiem: vērpējs rēžīms un kalendāra režīms, Peldošās darbības poga, kas izraisa darbības [38].

Izstrādes laikā bija izmantotas rīku dokumentācijas, kā arī pēc šo lietotņu izstrādes pabeigšanas bija iespējams salīdzināt lietotnēs izmēru.

5 IZSTRĀDĀTO LIETOTŅU ANALĪZE

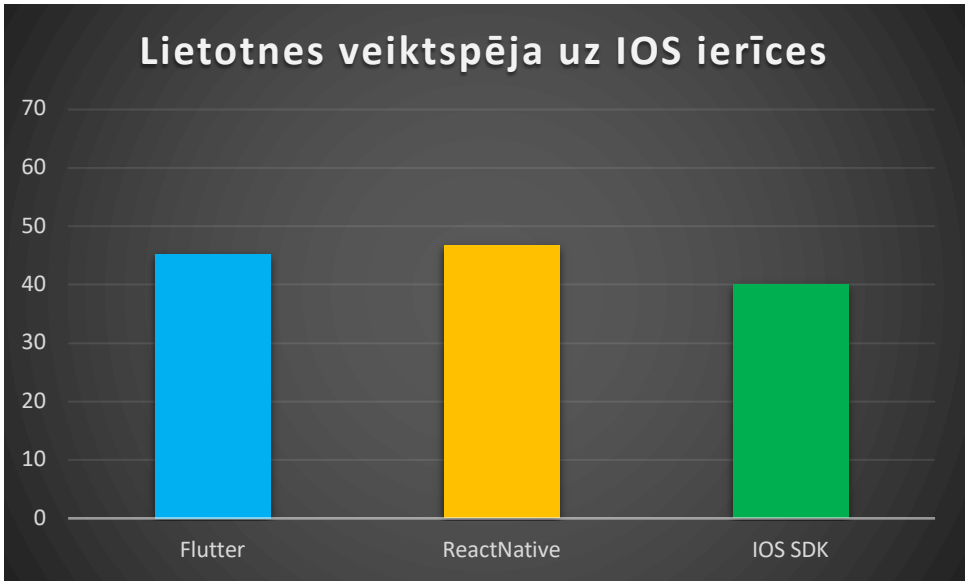
Šajā nodaļā sniegta katras ieviestas lietojumprogrammas analīze. Lietojumprogrammas, kas tika izveidotas, izmantojot vairāku platformu rīkus, tiek salīdzinātas ar vietējo lietojumprogrammu un sava starpa.

Kā minēts iepriekš (sk. 4. Izstrādātas lietošanasnes), lai pārbaudītu katras šķērsplatformu lietotnes veiktspēju, ir izstrādāta vienkārša vienas lapas lietojumprogramma, izmantojot izvēlētos šķērsplatformu izstrādes rīkus, kā arī kā atsauces punkts bija izveidotas arī vietējās lietojumprogrammās. Attēla 5.1 Lietojumprogrammu veiktspējas rezultāti IOS platformā un attēla 5.2 att. Lietojumprogrammu veiktspējas rezultāti Android platformā. ir attēloti pirmās testēšana gadījuma rezultāti. Abi attēli parāda, cik daudz laika, sekundēs, aizņem katras lietotnes izpilde.

Pirmais attēls parāda katras programmas izpildes rezultātu IOS platformā. Var novērot, ka kopējais izpildes laiks nepārsniedz 50 sekundes. Dzimtas IOS lietojumprogrammas vidējais izpildes rezultāts ir 40,02 sekundes. Nākamais labākais ir Flutter lietojumprogramma, kas par 5 sekundēm tiek izpildīta ilgāk nekā vietējā lietojumprogramma, vidējais lietotnes izpildes laiks ir 45,22 sekundes. Līdz šim sliktāko rezultātu parādīja lietojumprogramma, kas bija izstrādāta ar ReactNative ietvaru, vidējais izpildes laiks ir 46,73 sekundes, kas ir par 1,5 sekundēm ilgāk nekā Flutter lietotnes izpilde. Bija aprēķināts arī katras lietotnes palaišanas laiks.

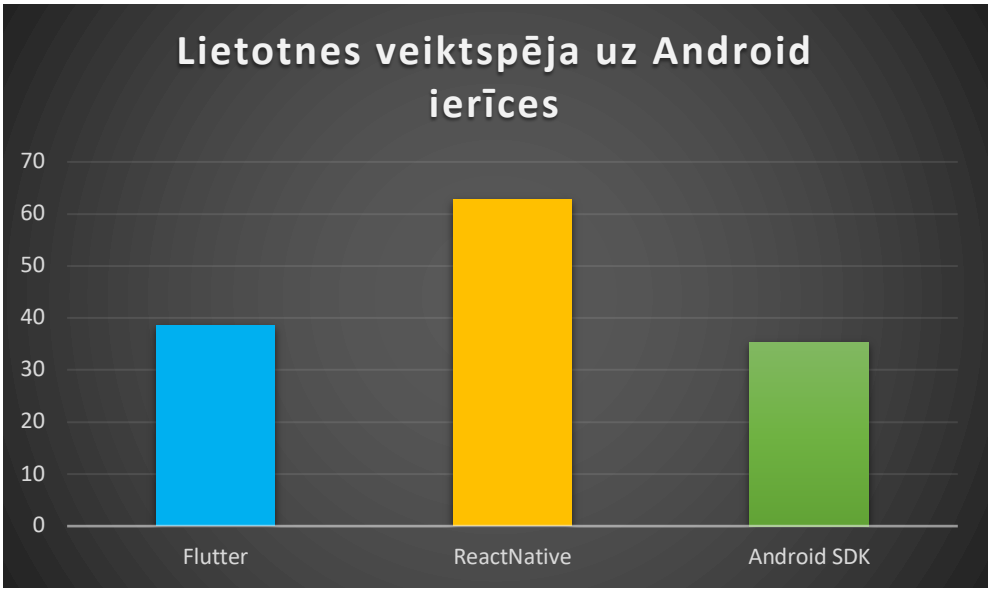
Lietotnes palaišanas laiks bija saskaitīts, apkopojot pre-main()laiku, kur pre-main () laiks ir laiks pirms lietotnes galvenas, jeb main() metodes atgriešanās, un post-main() laika, post-main () laiks, kur post-main () apraksta laiku, kas nepieciešams, lai lietojumprogrammu dati faktiski būtu redzami lietotājam [21].

Kopējais programmas palaišanas laiks IOS paltformā nepārsniedza 6 sekundes, Flutter lietotnes palaišanas vidēji aizņēma 2,7 sekundes un ReactNative lietotnes palaišana aizņēma 5,6 sekundes. Tomēr IOS vietējās lietojumprogrammas palaišana prasa līdz 3,4 sekundēm, kas par 0,7 sekundēm aizņem vairāk laikā nekā ar Flutter izveidotas lietotnes palaišana.



5.1.att. **Lietojumprogrammu veiktspējas rezultāti IOS platformā**

Attēlā 5.2. Lietojumprogrammu veiktspējas rezultāti Android platformā attēls parāda lietotnēs izpildes rezultātus android platformā. Var redzēt, ka salīdzinājumā ar Flutter lietotni, ar ReactNative izstrādāta lietotne parāda zemākas kvalitātes rezultātus. Vidējais ReactNative programmas izpildes laiks Android platformā bija 62,76 sekundes, kas ir gandrīz divas reizes ilgāk nekā uz Flutter izstrādātas programmas vidējais izpildes laiks, kas ir 38,55 sekundes. Var novērot, ka Flutter izstrādātā lietotnes veiktspēja ir diezgan tuvu vietējās lietotnes veiktspējai, kas ir 35.29s. Android platformā lietojumprogrammas palaišanas laiks vidēji bija 3,4 sekundes (Flutter), 4,8 sekundes (ReactNative) un 3,1 sekunde (vietējā lietotne).

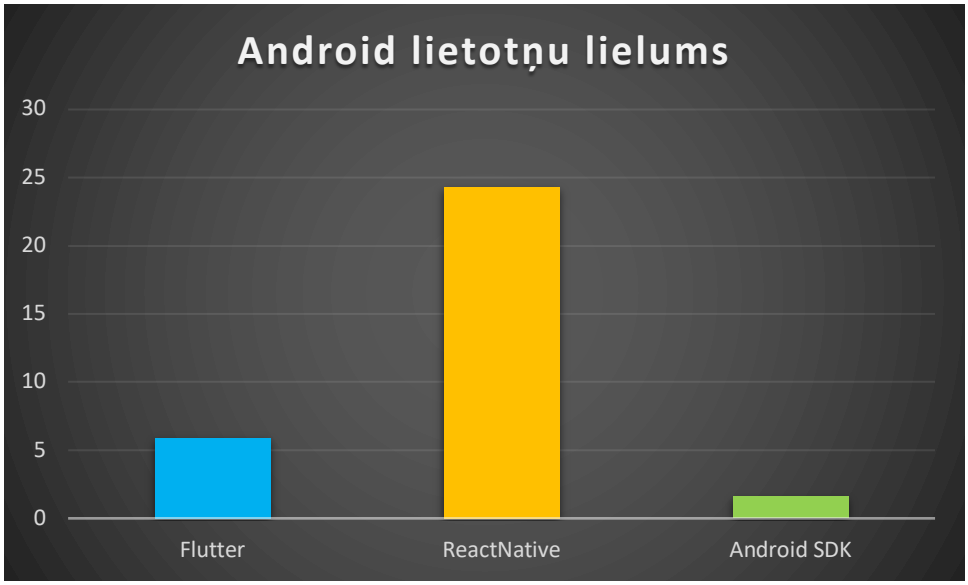


5.2.att. **Lietojumprogrammu veiktspējas rezultāti Android platformā**

Pielikumā ir sniegti attēli ar iegūtiem profilētājā datiem, attēli sniedz sīkāku informāciju par lietojumprogrammas izpildi. Attēlos var novērot CPU izmantošanu katrā lietojumprogrammā.

Saskaņā ar saņemtajiem profilētājā datiem vidējais lietojumprogrammu resursu patēriņš IOS platformā pārsniedz 50% no CPU. Lietotnes palaišanā vidējais CPU lietojums ir diezgan līdzīgs Android vietējai lietotnei un React Native lietotnei. Tomēr Flutter lietojumprogrammas palaišanai ir nepieciešams vairāk resursu. Andoird platformā var novērot, ka Android vietējo lietojumprogrammu resursu patēriņš vidēji ir aptuveni 25%, Flutter lietojumprogramma lielākoties patērē mazāk nekā 50% no CPU, turpretī, salīdzinot ar citām lietojumprogrammām, ir nepieciešams vairāk resursu, lai ReactNative lietojumprogramma darbotos Andorid platformā.

Attēlā Lietotnēs izmēri Android platformā (sk. 5.3att. Lietotnēs izmēri Android platformā) ir sniegti rezultējošās lietotnēs izmēriem dati. Lai noskaidrotu IOS platformai izstrādāto lietojumprogrammu izmēru, ir nepieciešams izveidot lietojumprogrammu laidiena versiju. Savukārt lai izveidotu laidiena versiju ir nepieciešams reģistrēties Apple izstrādātāju programmā. Sakarā ar nespēju reģistrēties izstrādātāju programmā nebija iespējams izveidot lietojumprogrammu laidiena versiju, tādējādi nebija iespējams pārbaudīt IOS lietotņu izmērus. Android platformas gadījumā lietojumprogrammu laidiena versija tika viegli izveidota, izmantojot vienu komandu: ./gradlew installRelease komandu uzvednē. Flutter lietojumprogrammas izmērs ir 5.9MB, kas ir rupji 3.5. reizes lielāks nekā vietējās android lietojumprogrammas izmērs, kas ir 1.6 MB. Tomēr, salīdzinot ar citām lietotnēm, React Native lietojumprogrammas lielums bija milzīgs – 24.3 MB.



5.3 att. **Lietotnēs izmēri Android platformā.**

Izstrādes laikā bija izmantoti gan Flutter, gan ReactNative dokumentācijas, tās abas ir labi strukturētas un sniedza detalizētus izstrādes soļus. Flutter dokumentācija arī sniedza ļoti detalizētus soļus lietojumprogrammas izlaišanas versijas ģenerēšanai. ReactNative tieša pārlādēšanas funkcija darbojās uzreiz pēc izmaiņu saglabāšanas, kas palīdzēju ietaupīt daudz laiku. Strādājot ar Flutter, karstā pārlādēšana strādāja ātri un tika atkārtoti ielādēta tikai dažu sekunžu laikā, tomēr, strādājot ar ReactNative, dažos gadījumos karstā pārlādēšanas funkcija varēja aizņemt divreiz vairāk laika.

No lietotja saskārnes skata punktā bija izpētīti daži elementi. Izmantojot vietējās IOS SDK ir vairāki veidi, kā ieviest saskarni, viens veids ir izmantot Swift UI pieeju. Swift UI ļauj izveidot lietotāja saskarnes jebkurai Apple ierīcei, izmantojot tikai vienu rīku un API komplektu. Tas nodrošina deklaratīvu Swift sintakse, kas ir viegli lasāma un ērta rakstīšanai [38]. Otrs veids ir izmantot interfeisa veidotāju - StoryBoard. StoryBoard ir iOS lietojumprogrammas lietotāja interfeisa vizuāls attēlojums, parādot satura ekrānus un savienojumus starp šiem ekrāniem. Turklāt XCode nodrošina objektu bibliotēku UIKit, kas satur daudzu lietotāja interfeisa elementu, piemēram, pogas, skati, un citu UI komponentus[38]. Lai pievienotu lietotāja interfeisa elementus, izstrādātājs var vilkt šos lietotāja interfeisa objektus uz interfeisa veidotāja audekla, kur vēlāk tie tiek sakārtoti, iestatīti to atribūti un izveidoti savienojumi starp tiem un kodu avota failos. Savukārt Android nodrošina dažādus jau iebūvētus lietotāja interfeisa komponentus, piemēram, strukturētus izkārtojuma objektus un lietotāja interfeisa vadīklas, kas ļauj izveidot lietotnes grafisko lietotāja saskarni. Android nodrošina arī iebūvētu komponentu vilkšanas un nomešanas funkcionalitāti, kad vienums tiek "nomests" ekrānā, uzreiz tiek ģenerēts XML kods, tomēr ir iespējams arī rakstīt XML ar roku.

Flutter ietvars nodrošina bagātīgu logrīku bibliotēku - Cupertino. Šajā bibliotēkā tiek ieviesta pašreizējā IOS dizaina valoda, kuras pamatā ir Apple saskarnes vadlīnijas [7]. Kā piemēru var apskatīt iOS stilizēto DatePicker koda fragments, kas redzams 5.4 att. Flutter DatePicker attēlā. Logrīku konfigurācija patiešām vienkārša un detalizēta informācija par ieviešanu tika labi aprakstīta Flutter dokumentācijā. Flutter sniedz Material Component logrīku bibliotēku, kas ir paredzēta vizuālo, uzvedības un kustību bagāto logrīku ieviešanai, tā ievieš Materiālu dizaina vadlīnijas, Android UI komponentu ieviešanai.

ReactNative bibliotēka nodrošina dažus komponentus ar jau noteiktu dizainu, tomēr tādi raksturlielumi kā platums vai augstums joprojām ir konfigurējami. UI komponentu ieviešanai ir pieejamas arī daudzas citas bibliotēkas, tomēr šī daudzveidība rada papildu laika patēriņu, arī dažas no piedāvātajām bibliotēkām ir sliktas kvalitātes. DatePicker ieviešana koda fragmentu var apskatīt 5.5 att. ReactNative DatePicker.

```
CupertinoDatePicker(
        initialDateTime: _dateTime,
        onDateTimeChanged: (dateTime){
          setState(() {
            _dateTime=dateTime;
          });
```

5.4 att. **Flutter DatePicker**

```
return (
  <View style={styles.container}>
    <DatePickerIOS
      date={chosenDate}
      onDateChange={setChosenDate}
    />
  </View>
);
```

5.5 att. **ReactNative DatePicker**

UI dizaina ieviešana abos rīkos ievērojami atšķiras. React Native izmanto Android un iOS sākotnējos komponentus:

Tas izmanto ārējos UI komplektus, piemēram, React Native Material Kit android stīlā elementu ieviešanai, React Native Elements un NativeBase, tas atbalsta arī komponentus iOS lietotņu saskarnes noformēšanai, kas ļauj izstrādātājiem izvēlēties komponentus atbilstoši viņu prasībām.

Turpretī Flutter nodrošinā savu risinājumu komponentu un logrīku UI ieviešanai. Tie ir iebūvēti un ir pieejami gan Android, gan iOS:

Materiāla dizaina logrīki nodrošina Android lietotņu dizainu, kur Cupertino logrīki atbilst iOS UI dizainam. Flutter lietotāja saskarne ir diezgan elastīga, un tā ātri apstrāda grafiskus elementus, kā arī ļauj ērti pielāgot logrīkus.

REZULTĀTI

Bakalaura darba ietvaros tika veikta detalizēta teorētisko materiālu izpēte. Padziļināti bija izpētīti šķersplatformas izstrādes pieejas, no teorētiskā skata punkta tika noteiktas katras pieejas stiprās un vājās pusēs. Detalizēta pieejās izpēte palīdzēja noteikt, kura no šķērsplatformu izstrādes pieejām potenciāli nodrošinās labākos iespējamos rezultātus un pieredzi lietojumprogrammu izstrādes ziņā. Šī pētījumu daļā tika noteiktas katra pieejās stipras un vājas pusēs, kā arī rezultātā tika izvēlētie pieci šķērsplatformu rīki turpmākai analīzei.

Nākamajā sadaļā tika detalizēti izpētīti izvēlētie šķērsplatformu izstrādes rīki. Bija izvēlēti pieci, gan interpretētie, gan šķērskompilētie rīki lietojumprogrammu izstrādei, pamatoties uz pirmajā nodaļā paveikto izpētī. Tādējādi par katru no izvēlētajiem rīkiem ir veikts vēl viens rūpīgs pētījums. Balstoties uz šķērsplatformu izstrādes rīku izpēti tika definēti katra ietvara priekšrocības, un izvēlēti turpmākai analīzei interesējoši rīki.

Rīku un vispārīga teorētisko materiālu izpēte veicināja praktiskās daļas ieviešanai, jo pētījumā bija iespējams iepazīties ar rīku dokumentācijām pirms izstrādes procesa sākuma.

Tika izstrādātas četras lietotnes. Pirmkārt, tika izstrādātas vietējās lietojumprogrammas, kas palīdzēja izlemt šķērsplatformu lietojumprogrammu ieviešanas veidus.

Tālāk bija izstrādātas šķērsplatformu lietotnes izmantojot ReactNative un Flutter rīkus.

Turklāt, praktiskajā daļā ietvaros, tika izstrādāti divi automatizēti testēšanas skripti, kas veicināja lietotnēs godīgu veiktspējas salīdzināšanai. Pēc visu četru lietojumprogrammu veiksmīgas izstrādes, tās lietotnes tika salīdzinātas savā starpā. Tomēr ir svarīgi pieminēt, ka salīdzinājums bija atkarīgs no platformas, kas nozīmē, ka katra no platformu šķērsplatformu lietotnēm tika palaista gan Android, gan IOS ierīcē, un vēlāk tika salīdzināta ar vietējām lietotnēm. IOS un Android lietojumprogrammas netika salīdzinātas savā starpā. Visas lietotnes tika analizētas pēc izvirzītiem kritērijiem. Tika analizēts veiktspējas laiks, katras android platformai izveidotās lietojumprogrammas lielums, kā arī tika ņemts vērā cik daudz laika ir nepieciešams karstajai pārlādēšanai, rīka dokumentācijas un lietotāja saskarnes kvalitāte.

SECINĀJUMI

Izstrādes pieeju salīdzināšanas laikā tika secināts, ka šķērskompilētas un interpretētas pieejas, visticamāk, nodrošinās tuvu dzimtajām lietotnēm veiktspēju un piemērojamo lietotāja saskarni. Rīku salīdzināšanas laikā tika secināts, ka Flutter un React Native var sniegt labākos iespējamos rezultātus izstrādes ātruma un sarežģītības ziņā.

Šķersplatoformu lietojumprogrammu ieviešanas laikā tika secināts, ka lietojumprogrammas izstrādāta funkcionalitāte tiek koplietotā starp abām platformām. Dizaina ziņā IOS lietojumprogrammas un Android lietotnes ievēro dažādas pamatnostādnes, tāpēc UI komponentu ieviešanā ir atšķīras atkarība no platformas. UI dizaina ieviešanas metodes ievērojami atšķiras starp Flutter un ReactNative. ReactNative izmanto Android un iOS dzimtus komponentus. Turpretī Flutter logrīku zīmēšanai izmanto savu augstas veiktspējas renderēšanas motoru, kā arī šīs ietvars nodrošina jaudīgu parastus logrīkus. Flutter nodrošina platformai specifiskas bibliotēkas, kuras nodrošina logrīku komplektus, kas ir īpaši izveidoti konkrētai platformai, t.i. tiek ņemtas vērā platformas elementu dizaina vadlīnijas.

Abu instrumentu dokumentācija ir diezgan detalizēta un labi strukturēta, kas veicināja izstrādes procesu.

Lietojumprogrammu analīze parādīja, ka veiktspējas ziņā Flutter lietojumprogrammas darbojas ātrāk, kā arī vidēji patērē mazāk CPU resursus, nekā ReactNative lietotnes. Flutter programmas veiktspējas rezultāti bija gandrīz dzimtā valoda bija diezgan tuvu vietējo programmu veiktspējas rezultātiem. Kaut arī abu starp platformu lietojumprogrammu izmērs bija ievērojami lielāks nekā vietējo lietotnes izmērs, salīdzinājumā ar ReactNative izveidota lietotnes izmēru, Flutter lietotnes izmērs bija rupji četras reizes mazāka.

Subjektīvi autore izstrādi Flutter ietvarā vērtē kā labāku un ērtāku nekā vienas un tās pašas lietojumprogrammas izstrāde, izmantojot ReactNative.

IZMANTOTĀ LITERATŪRA UN AVOTI

[1] Number of smartphone users worldwide from 2016 to 2021. [tiešsaiste]. Piekļūts [14.04.2020]. Pieejams: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] Shaun Lewis, Mike Dunn. Native Mobile Development: A Cross-Reference for iOS and Android 1st Edition.

[3] Minh Huynh, Prashant Ghimire, Donny Truong. HYBRID APP APPROACH: COULD IT MARK THE END OF NATIVE APP DOMINATION? [tiešsaiste]. Piekļūts: [17.04.2020]. Pieejams: http://iisit.org/Vol14/IISITv14p049-065Huynh3472.pdf

[4] Andreass Birns-Hansens, TimA.Majchrzak, Tor-MortenGrønli. Progressive Web Apps: The Possible Web-native Unifier for Mobile Development. [tiešsaiste]. Piekļūts [05.04.2020]. Pieejams: https://www.scitepress.org/papers/2017/63537/63537.pdf

[5] Charles Petzold. Creating Mobile Apps with Xamarin. Forms

[6] Xamarin Ietvara Dokumnetācija. [tiešsaiste]. Piekļūts [20.04.2020]. Pieejams: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

[7] Flutter Ietvara Dokumnetācija. [tiešsaiste]. Piekļūts [14.04.2020]. Pieejams: https://flutter.dev

[8] Arno Puder, Victor Woeltjen, Alon Zakai. Cross-Compiling Java to JavaScript via Tool-Chaining. [tiešsaiste]. Piekļūts [16.04.2020]. Pieejams: https://dl.acm.org/doi/abs/10.1145/2500828.2500831

[9] Blackberry Platformas Documentācija. [tiešsaiste]. Piekļūts [14.04.2020]. Pieejams: https://developer.blackberry.com/native/documentation/getting_started/first_app/index.html

[10] Appcelator Titanium ietvara mājās lapa. [tiešsaiste]. Piekļūts [15.04.2020]. Pieejams: https://www.appcelerator.com/mobile-app-development-products/

[11] Bonnie Eisenman. Learning React Native: Building Native Mobile Apps with JavaScript Kindle Edition.

[12] John Anderson. Appcelerator Titanium: Up and Running.

[13] Vladimir Novick. React Native - Building Mobile Apps with JavaScript.

[14] React Ietvara Dokumentācija. [tiešsaiste]. Piekļūts [10.05.2020]. Pieejams: https://reactjs.org/docs/introducing-jsx.html

[15] Akshat Paul, Abhishek Nalwaya. React Native for Mobile Development. Harness the Power of ReactNative to Create Stunning iOS and Android Applicaitons. Second Edition.

[16] Mika Kuitunen. Cross-Platform Mobile Application Development With React Native. [tiešsaiste]. Piekļūts [20.05.2020]. Pieejams: https://trepo.tuni.fi/bitstream/handle/123456789/27139/Kuitunen.pdf?sequence=4&isAllowed=y

[17] Marco L. Napoli. Beginning Flutter: A Hands On Guide to App Development.

[18] Smith, A., (2010). Mobile Access 2010. Pew Internet & American Life Project, Jusly 7, 2010.

[19] S. Hay "Responsive Design Workflow", 1st ed. New Riders, United Stated of America, 2013.

[20] Tal Ater. Building Progressive Web Apps: Bringing the Power of Native to the Browser

[21] M. Faela. Seriously Good Software. Shelter Island, NY: Manning Publications Co, 2020.

[22] Jonathan Peppers, George Taskos, Can Bilgin. Xamarin: Cross-Platform Mobile Application Development.

[23] Mobile Operating System Market Share Worldwide Statistics. [tiešsaiste]. Piekļūts [14.05.2020]. Pieejams: https://gs.statcounter.com/os-market-share/mobile/worldwide

[24] Esteban Angulo, Xavier Ferre. A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX. [tiešsaiste]. Piekļūts [14.05.2020]. Pieejams: https://dl.acm.org/doi/pdf/10.1145/2662253.2662280

[25] John A. Hoxmeier, Chris DiCesare. System Response Time and User Satisfaction: An Experimental Study of Browser-based Applications. [tiešsaiste]. Piekļūts [14.05.2020]. Pieejams: https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1799&context=amcis2000

[26] Niesen Norman Groups, World Leaders in Research-Based User Experience. Does User Annoyance Matter? [tiešsaiste]. Piekļūts [14.05.2020]. Pieejams: https://www.nngroup.com/articles/does-user-annoyance-matter/

[27] Android Ietvara Dokumentācija. [tiešsaiste]. Piekļūts [15.05.2020]. Pieejams: https://developer.android.com/

[28] Adam Boduch. React and React Native: Complete guide to web and native mobile development with React, Second Edition, 2018.

[29] NativeScript Ietvara Dokomentācija. [tiešsaiste]. Piekļūts [16.05.2020]. Pieejams: https://www.nativescript.org

[30] Ivano Malavolta , Stefano Ruberto , Tommaso Soru , Valerio Terragni. End Users' Perception of Hybrid Mobile Apps in the Google Play Store. [tiešsaiste]. Piekļūts [16.05.2020]. Pieejams: https://www.researchgate.net/profile/Ivano_Malavolta/publication/307910225_End_Users%27_Perception_of_Hybrid_Mobile_Apps_in_the_Google_Play_Store/links/58bf33cca6fdccff7b1f9d38/End-Users-Perception-of-Hybrid-Mobile-Apps-in-the-Google-Play-Store.pdf

[31] Dan HermesNima Mazloumi. Building Xamarin.Forms Mobile Apps Using XAML. Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals

[32] Austin Borop. Xamarin Forms vs Native Platform Development. [tiešsaiste]. Piekļūts [14.05.2020]. Pieejams: https://digitalcommons.olivet.edu/cgi/viewcontent.cgi?referer=https://scholar.google.com/scholar?start=10&q=xaml+xamarin&hl=ru&as_sdt=0,5&httpsredir=1&article=1008&context=csis_stsc

[34] Gatis Vitols , Ingus Smits and Oleg Bogdanov. Cross-platform Solution for Development of Mobile Applications. [tiešsaiste]. Piekļūts [17.05.2020]. Pieejams: https://pdfs.semanticscholar.org/d691/a2a19b46fd7df7d57f3231fb73313d997279.pdf

[35] Patricia Pesado, Claudio Aciti. Computer Science – CACIC 2018: 24th Argentine Congress, Tandil, Argentina 2018.

[36] Wafaa S. El-Kassas*, Bassem A. Abdullah, Ahmed H. Yousef, Ayman M. Wahba. Taxonomy of Cross-Platform Mobile ApplicationsDevelopment Approaches. [tiešsaiste]. Piekļūts [14.04.2020]. Pieejams: https://www.researchgate.net/publication/283299324_Taxonomy_of_Cross-Platform_Mobile_Applications_Development_Approaches

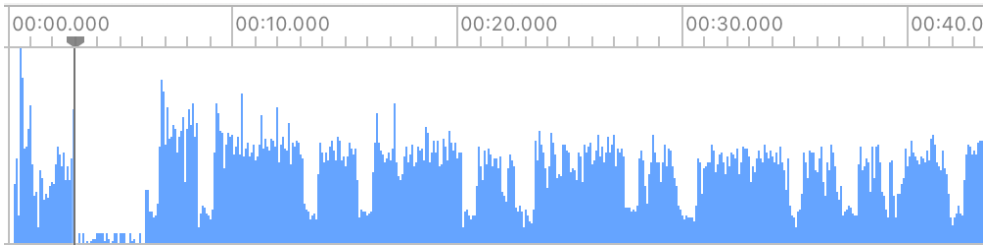
[37] Appium Rīka Dokumentācija. [tiešsaiste]. Piekļūts [14.04.2020]. Pieejams: http://appium.io

[38]Apple izstrādātāja Dokumnetācija. [tiešsaiste]. Piekļūts [14.04.2020]. Pieejams: https://developer.apple.com/documentation

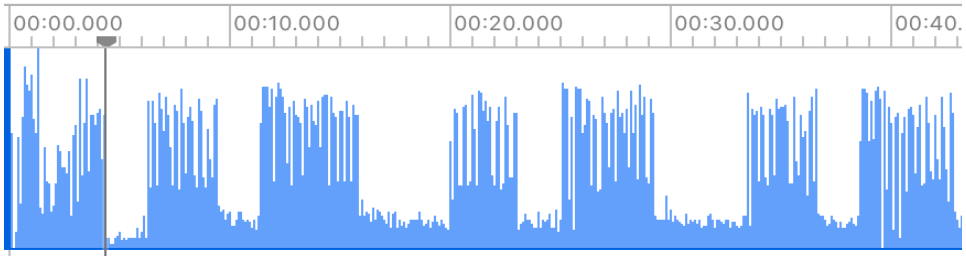
[39] Bc. Dominik Veselý. Analysis and experiments with NativeScript and React Native framework. [tiešsaiste]. Piekļūts [14.04.2020]. Pieejams: https://is.muni.cz/th/448261/fi_m/thesis.pdf

PIELIKUMI

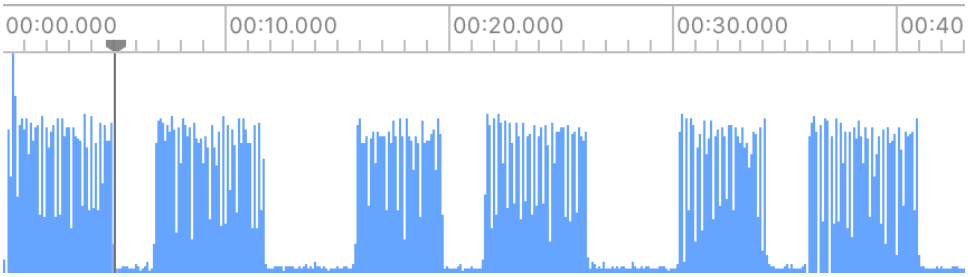
Lietotņu profilēšanas rezultāti



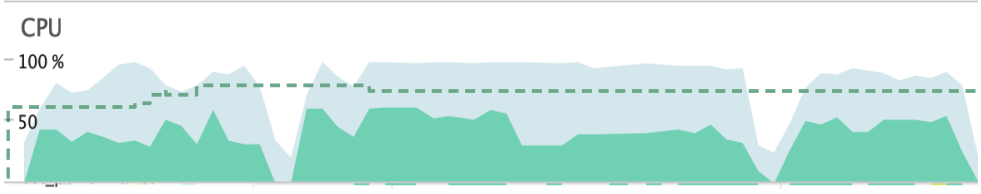
1.att. **Flutter lietotnes profilēšanas rezultāti, iOS platformā**



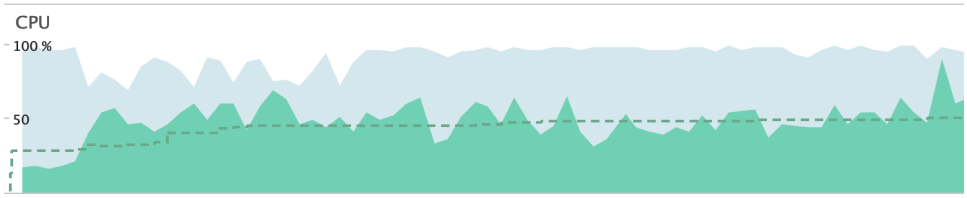
2.att. **ReactNative lietotnes profilēšanas rezultāti, iOS platformā**



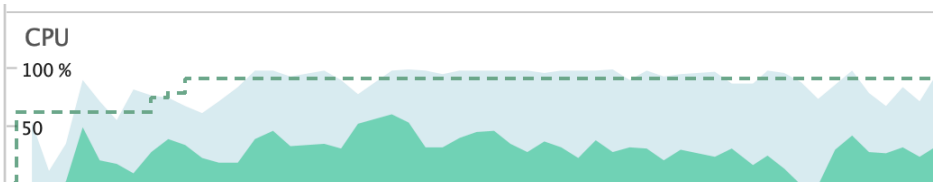
3.att. **Dzimtas iOS lietotnes profilēšanas rezultāti, iOS platformā**



4.att. **Flutter lietotnes profilēšanas rezultāti, Android platformā.**



5.att. **ReactNative lietotnes profilēšanas rezultāti, Android platformā**



6.att. **Dzimtas Android lietotnes profilēšanas rezultāti, Android platformā**

Bakalaura darbs „Šķērsplatformu Izstrādes Rīku Salīdzinājums Mobilās Lietojumprogrammas Rakstīšanai” 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: _____ Jevgēnija Gluškova <paraksts>

Rekomendēju/nerekomendēju darbu aizstāvēšanai *(nederīgo svītro vadītājs)*

Vadītājs: Dr.dat. Uldis Straujums <paraksts>_____ __.06.2020.

Recenzents: Asociētais profesors Dr.dat. Edgars Celms

Darbs iesniegts Datorikas fakultātē __.06.2020.

Dekāna pilnvarotā persona: vecākā metodiķe Ārija Sproģe _____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

____.06.2020. prot. Nr. _____

Komisijas sekretārs(-e): _____