TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut

IDK40LT
Oliver Kuldmäe 134572

HÜBRIID- JA PLATVORMIPÕHISTE MOBIILIRAKENDUSTE JÕUDLUSE ANALÜÜS

Bakalaureusetöö

Juhendaja: Deniss Kumlander
Tehnikateaduste doktor
Vanemteadur

Tallinn 2016

Autorideklaratsioon

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

Autor: Oliver Kuldmäe

21.05.2016

Annotatsioon

Käesoleva töö eesmärgiks on võrrelda hübriid- ja platvormipõhiseid mobiilirakendusi lähtuvalt kiirusest ning jõudlusest. Samuti pakub see autorile võimaluse tutvuda mobiilitarkvara arendamisega ning annab aimu sellest, kas hübriidrakendused on jõudluselt sarnased platvormipõhiste rakendustega.

Töö tutvustab populaarsemaid mobiilseid operatsioonisüsteeme, mobiilirakenduste tüüpe ning jõudluse analüüsimiseks vajalikke teste. Töö käigus on realiseeritud neli samaväärse funktsionaalsusega mobiilirakendust, sisaldades endas nelja erinevat testi.

Lõputöö tulemuseks on analüüs sellest, millised on hübriid- ja platvormipõhiste rakenduste head küljed ning puudujäägid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 44 leheküljel, 4 peatükki, 32 joonist, 1 tabelit.

Abstract

Hybrid and Native Mobile Application Performance Analysis

The goal of this thesis is to compare the performance of hybrid and native mobile applications. It also provides the author a chance to get acquainted with mobile application development and helps determine whether hybrid applications are similar to native applications performance-wise.

The thesis introduces popular mobile operating systems, mobile application types and tests required to analyze the performance of said application types. Four functionally equal mobile applications, each containing four different tests, are created during the writing process of this thesis.

The result of this thesis is an analysis of the advantages and shortcomings of both hybrid and native applications.

The thesis is in Estonian and contains 44 pages of text, 4 chapters, 32 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application program interface*, programmiliides
APK	*Android application package*, arhiiv Android rakenduse hoidmiseks
CSS	*Cascading Style Sheets*, kaskaadlaadistik, kasutatakse HTMLi kujundamiseks
GPS	*Global Positioning System*, globaalne positsioneerimissüsteem
HTML	*HyperText Markup Language*, hüperteksti märgistuskeel, kasutatakse veebilehtede struktuuri loomiseks
IPA	*iOS App Store Package*, arhiiv iOS rakenduse hoidmiseks
JSX	*JavasScript XML*
MB	*Megabyte*, megabait, mälumahu ühik, vastab ligikaudu ühele miljonile baidile
XML	*Extensible Markup Language*, märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

1.1 Taust ja probleem

Mobiilseadmete turg on paljutõotav ja kiirelt kasvav. Uuringu- ja nõustamisettevõte Gartner väidab, et ülemaailmselt müüdi 2015. aasta kolmandas kvartalis ligi 350 miljonit nutitelefoni. See oli 15,5% suurune kasv võrreldes 2014. aasta kolmanda kvartaliga [1].

Tabel 1. Mobiilseadmete ülemaailmne müük tootjate kaupa aastatel 2014 ja 2015

Ettevõte	**3Q15 Seadmete arv[1]**	**3Q15 Turuosa (%)**	**3Q14 Seadmete arv[1]**	**3Q14 Turuosa (%)**
Samsung	83,586.7	23.7	72,929.4	23.9
Apple	46,062.0	13.1	38,186.6	12.5
Huawei	27,262.3	7.7	15,935.0	5.2
Lenovo[2]	17,439.2	4.9	21,314.1	7.0
Xiaomi	17,197.2	4.9	15,772.5	5.2
Muud	161,296.6	45.7	141,246.5	46.3
Kokku	352,844.0	100.0	305,384.0	100.0

Kommunikatsioonifirma Ericssoni 2015. aasta raporti põhjal oli samal aastal 7,4 miljardit aktiivset mobiilside kasutajat ning 2021. aastaks ennustatakse selle arvu tõusu 9,1 miljardini [2]. Mobiilseadmete turu kiire kasvu tulemuseks on suur nõudlus erinevate mobiilirakenduste järgi. Turu rikastumine erinevate seadmetega tekitab uusi riistvara ja tarkvara kombinatsioone, mis vajavad spetsiaalselt nendele platvormidele mõeldud rakenduste loomist. See tähendab, et mitme platvormi toetamiseks luuakse sama funktsionaalsusega tarkvara mitu korda. Probleemile lahenduse leidmiseks on loodud hübriidrakendused, mis pakuvad platvormipõhiste rakendustele sarnast jõudlust [3], kuid töötavad ühe koodibaasi põhjal. Siit tekibki töö põhiliseks probleemiks olev küsimus –

1 Seadmete arvud on tuhandetes.

2 Lenovo tulemused sisaldavad endas nii Lenovo kui Motorola seadmete müüke nii 2015. aasta kui ka 2014. aasta kolmandas kvartalis.

kas hübriidrakendused on tõesti jõudluselt sama võimekad kui platvormipõhised rakendused?

1.2 Ülesande püstitus

Töö eesmärgiks on analüüsida ja võrrelda hübriid- ning platvormipõhiste mobiilirakenduste jõudlust. Seeläbi saab aimu, kas hübriidrakendusi on võimalik kasutada võimeka alternatiivina platvormipõhistele rakendustele tulevaste projektide jaoks. AppDynamicsi poolt läbiviidud uuringu põhjal on mobiilirakenduste kasutajate jaoks tülikaimad probleemid just jõudlusega seotud ning kümnest rohkem kui kaheksa kasutajat on selliste probleemide tõttu rakendusi kustutanud [4]. Seetõttu on oluline, et mobiilirakendused oleks optimeeritud ning töötaksid kiiresti.

1.3 Metoodika

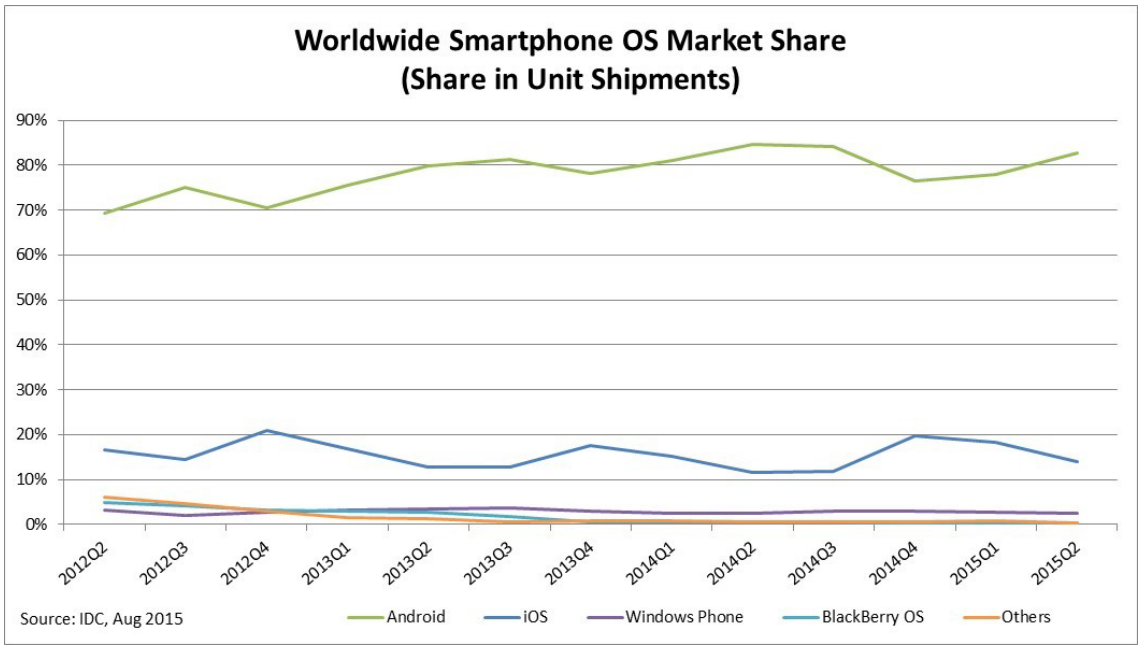
Erinevate rakenduste jõudluse analüüsimiseks ja võrdlemiseks loob töö autor valitud raamistikke ja platvorme kasutades mobiilirakendused, mis sisaldavad endas eesmärgi täitmiseks mõeldud teste. Testide loomisel lähtuti Micro to Mainframe ettevõtte tarkvara testimise metoodikast. Nimetatud metoodika järgi loodi baastestid (*baseline tests*), mis pakuvad mõõdikuid kindlate üksikute protsesside kohta [5].

1.4 Ülevaade tööst

Töös tutvustab autor erinevaid mobiilseid operatsioonisüsteeme ning mobiilirakenduste tüüpe ja nende struktuuri. Samuti kirjeldab autor rakenduste võrdlemiseks kasutatud vahendeid, teste ning testide tulemusi.

2 Mobiilsed operatsioonisüsteemid

Mobiilseadmete turu võrdlemisi lühikese eluea jooksul on loodud mitmeid erinevaid operatsioonisüsteeme, näiteks Symbian, Blackberry OS, Windows Phone, iOS ja Android. Turu-uuringu, -analüüsi ja nõustamisfirma International Data Corporation (IDC) andmetel olid 2015. aasta teises kvartalis kaheks populaarseimaks operatsioonisüsteemiks Android ja iOS, omades vastavalt 82,8% ja 13,9% mobiilseadmete operatsioonisüsteemide turust [6].



Joonis 1. Ülemaailmsed mobiilsete operatsioonisüsteemide turuosad

Võrdlemiseks kasutatavate platvormide valiku tegemisel lähtuti populaarsusest. International Data Corporationi andmetel olid Android ja iOS 2015. aastal teiste mobiilsete operatsioonisüsteemidega võrreldes märgatavalt populaarsemad, seega olid need ka valitud platvormideks. Pärast iOSi tuli Windows Phone 2,6% suuruse turuosaga, mis ei olnud autori arvates selles töös kajastamiseks piisavalt suur näitaja [6].

2.1 Android

Android on mobiilne operatsioonisüsteem, mille arendaja ning haldaja on Google. Android leidis oma alguse Palo Altos, Californias, samanimelise *start-up* ettevõtte käe

all. Google ostis ettevõtte ära 2005. aasta juulis ning on sellest ajast saati seda platvormi edasi arendanud [7]. Nimetatud operatsioonisüsteem on ülemaailmselt äärmiselt populaarne ja on kasutusel paljude erinevate tootjate seadmetel. Seetõttu peab Android toetama pea lõputul arvul erinevaid riistvara konfiguratsioone.

2.2 iOS

iOS on tehnoloogiafirma Apple'i poolt loodud mobiilne operatsioonisüsteem, mis leidis alguse 2007. aastal, tulles välja koos esimese iPhonega ning kandes nime iPhone OS. 2010. aasta juunis nimetati operatsioonisüsteem ümber ning uueks nimeks sai iOS [8]. IDC andmetel oli 2015. aasta teises kvartalis iOS paigaldatud 13,9% mobiilseadmetele [6]. Kuigi see osa on märgatavalt väiksem Androidi turuosast, on see siiski teiste operatsioonisüsteemidega võrreldes suureks osaks. Erinevalt Google'i Androidist paigaldatakse iOSi ametlikult ainult Apple'i enda loodud toodetele, näiteks iPhonedele ja iPadidele. Kuna riistvara konfiguratsioone on võrdlemisi vähe, siis on operatsioonisüsteemi optimeerimise võimalused märgatavalt suuremad, kui Androidi puhul.

3 Mobiilirakendused

Mobiilirakenduste lisavad, täiustavad või muudavad seadmete visuaalset külge või funktsionaalsust. Üldjuhul laetakse tarkvara alla operatsioonisüsteemi või seadme tootja veebipoest kas tasuta või raha eest. Näiteks Apple'i puhul App Storest ning Google'i puhul Google Playst.

3.1 Platvormipõhised ehk *native* rakendused

Platvormipõhised rakendused kasutavad operatsioonisüsteemi loojate poolt valitud arenduskeelt, on üldiselt optimeeritud, võimaldavad kasutada seadmespetsiifilist funktsionaalsust ning pakuvad kindlale platvormile mõeldud kasutajaliidese lahendusi ja arendusvahendeid. Sellised rakendused töötavad ainult nende jaoks mõeldud operatsioonisüsteemidel ning seetõttu on mitmele platvormile arendamisele ajakulukas ja kallis.

3.1.1 Android

Androidi platvormipõhiste rakenduste arenduskeeleks on Java. Google'i poolt ametlikult pakutavaks arendusvahendiks on Android Studio [9]. Stack Overflow poolt läbiviidud arendajate küsitluse andmetel olid Java ja Android 2015. aastal "*Top Tech*" valdkonnas vastavalt teisel ja kolmandal kohal [10]. See teeb Androidi populaarseks nii lõppkasutajate kui ka arendajate seas.

Androidi rakendused kompileeritakse lähtekoodist APK failideks, mis sisaldab endas peale koodi ka kõiki andmeid ja ressursse, mida rakendusel vaja läheb. Android on Linuxi süsteem, kus iga rakendus saab eraldi kasutaja, virtuaalmasina, protsessi ning õigused ainult endaga seotud rakenduse andmete kasutamiseks [11].

Iga rakendus võib koosneda neljast erinevast komponendist: tegevustest (*Activities*), teenustest (*Services*), sisupakkujatest (*Content Providers*) ja teatesaajatest (*Broadcast Receivers*). Tegevused kujutavad endast ühte kindlat kasutajaliidese lehte, näiteks e-posti rakendusel võib üks tegevus olla kirjade lugemine, teine tegevus kirja koostamine.

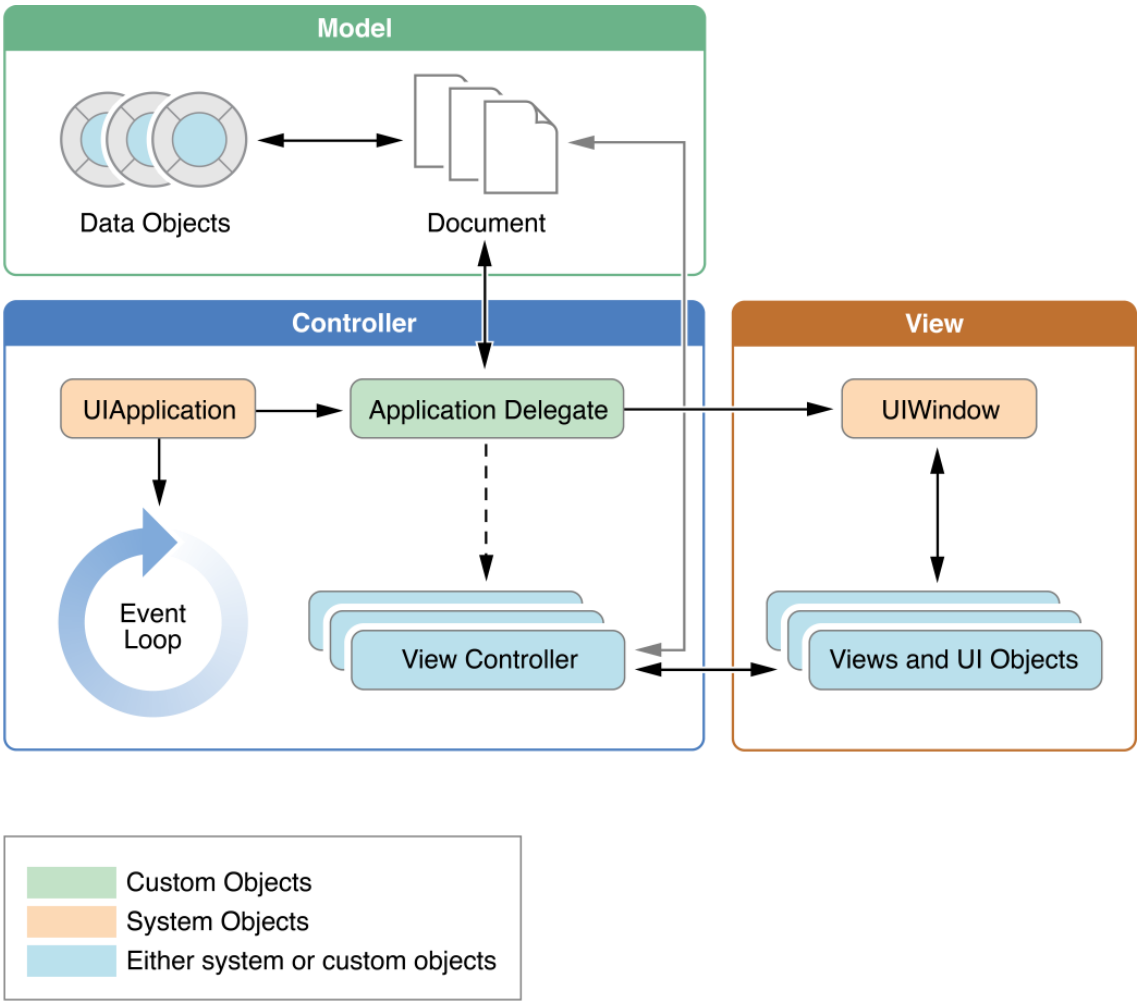
Teenused kasutajaliidest ei paku, vaid tegelevad taustatöödega, näiteks andmete pärimisega ja selle pakkumisega tegevusele. Sisupakkujad on samuti visuaalse küljeta ning võimaldavad pärida lokaalsest andmebaasist või failisüsteemist andmeid. Teatesaajad reageerivad teavitustele või tekitavad neid, et anda rakendusele teada, kui mõni tegevus on alanud või lõppenud [11].

Kolme neist komponentidest, tegevusi, teenuseid ja sisupakkujaid, käivitavad sama liiki asünkroonsed sõnumid, mida kutsutakse kavatsusteks (*Intent*). Kavatsused annavad rakendusele märku, et on vaja tegutseda. Näiteks võib kavatsus anda rakendusele teada, et on vaja alustada tegevust, kuvamaks kasutaja seadmele kasutajaliidest või pärida andmeid. Teatesaajaid käivitatakse siis, kui saadakse spetsiaalse otstarbega sisulahendaja (*Content Resolver*) objektilt päring [11].

Komponendi käivitamiseks on Androidi süsteemil vaja teavet komponendi kohta, mis pannakse kirja rakenduse manifesti. See manifest kannab nime AndroidManifest.xml ja selles on deklareeritud kõik komponendid, rakenduse käivitamiseks vajalikud õigused, minimaalne Android programmiliidese (*API)* tase, erilised riistvara ja tarkvara funktsionaalsused, mida rakendus kasutab, programmiliidese teegid, mida rakendus vajab jne. Kõik eelnimetatud rakenduse osad võimaldavad Androidi süsteemil tarkvara edukalt käivitada [11].

3.1.2 iOS

Apple'i iOS operatsioonisüsteemile platvormipõhiste rakenduste loomiseks on võimalik kasutada kahte erinevat arenduskeelt: Objective-C ja Swift. Swift on 2014. aastal avalikkuse ette toodud Apple'i poolt loodud arenduskeel, mis on 2015. aasta detsembrist alates ka avatud lähtekoodiga [12], [13]. Stack Overflow arendajate küsitluse andmete põhjal oli Swift teine armastatuim arenduskeel ning lähtekoodi avamise kuust alates ka populaarsem kui Objective-C [10]. iOS rakenduste ametlikuks arendusvahendiks on Apple'i Xcode, mida saab kasutada vaid Mac seadmetel.



Joonis 2. iOS rakenduste disainimuster

iOS rakendused järgivad tavalist MVC ning delegeerimise disainimustrit [14]:

- UIApplication objekt vastutab sündmuste ringi (*Event Loop*) eest ning teavitab rakenduse üleminekuid delegaadile.
- Rakenduse delegaat (*Application Delegate*) on arendaja poolt loodud kood ning rakenduse tuum. Delegaadi ülesanneteks on rakenduse initsialiseerimine, liikuda seisundite vahel ning paljude muude kõrgetasemeliste sündmustega tegelemine.
- Andmeobjektid (*Data Objects*) ja dokumendid (*Document*) hoiavad rakendusega seotud andmeid ning ressursse, näiteks rakendusele vajalikku andmebaasi või pilte.
- Vaatekontrollerid (*View Controller*) vastutavad rakenduse sisu kuvamise eest seadme ekraanile. Kui kutsutakse välja vaatekontroller, siis muudab nimetatud objekt nähtavaks endaga seotud vaated (*View*) ning alamvaated. Vaatekontroller pakub vaikimisi funktsionaalsust vaadete laadimiseks, nende kuvamiseks ning

pööramiseks vastavalt seadme orientatsioonile ja muid standardseid süsteemi käitumisi.

- UIWindow objekt haldab vaadete kuvamist ühele või mitmele ekraanile. Tüüpiliselt on seadmetel üks aken, millele kuvatakse rakenduse sisu, kuid võib olla ka näiteks teine aken, mis paikneb mõnel muul ekraanil.
- Vaated on rakenduse sisu visuaalne esitlus. Vaade on objekt, mis joonistab sisu nelinurksele pinnale ning vastab sündmustele sellel alal.

Sarnaselt Androidile kompileeritakse iOS rakenduse lähtekood IPA arhiiviks, milles on rakenduse käivitamiseks vajalikud failid ja ressursid. Näiteks sisaldab IPA arhiiv Info.plist faili, milles on kogu rakenduse seadistuse andmed. Selle abil saab süsteem kindlaks teha, kuidas rakendusega suhelda. Samuti on arhiivis ikoonid ja pildid, mida kasutatakse Apple'i rakenduste veebipoes [15].

3.2 Veebirakendused

Veebirakendused on HTMLi, CSSi ja JavaScripti kasutavad veebilehed, mis on optimeeritud mobiilide jaoks ning jätavad platvormipõhise rakenduse mulje. Veebirakenduste puhul on võimalik kasutada seadmespetsiifilist funktsionaalsust, näiteks GPSi ja kaamerat, kuid võimalusi on märgatavalt vähem kui platvormipõhistel või hübriidrakendustel. Veebirakendused võimaldavad vähendada arenduskulutusi ning -aega, kuna on kasutatavad kõigis populaarsemates tänapäeva brauserites. Kuna töö eesmärgiks on võrrelda just hübriid- ja platvormipõhiseid rakendus, siis veebirakenduste jõudlust ei analüüsita.

3.3 Hübriidrakendused

Sarnaselt veebirakendustele on hübriidrakenduste eesmärgiks vähendada mobiilirakenduste loomisega soetud aega ning kulutusi ja lihtsustada tarkvara loomist mitmele platvormile, samal ajal pakkudes platvormipõhiste rakendustega võrreldavat jõudlust ning funktsionaalsust. Erinevatel hübriidrakenduste raamistikel võivad olla ka võrdlemisi teistsugused arendusideoloogiad.

3.3.1 React Native

React Native'i eestvedajaks ja loojaks on Facebook, millele kuulub ka samanimeline sotsiaalmeedia võrgustik. React Native toodi avalikkuse ette 2015. aasta märtsis ning esialgu toetas vaid iOSi. Sama aasta septembris lisati Android tugi [16]. React Native on ehitatud JavaScripti teegi React põhjal, mis oli Stack Overflow arendajate küsitluse andmetel 2015. aastal suure eduga kõige trendikam tehnoloogia [10]. Raamistik valiti testimiseks, kuna töö kirjutamise hetkel jõudlust analüüsivaid töid selle raamistiku kohta tehtud pole ning tegemist on uue ja populaarse tehnoloogiaga, mis pakub huvitavat alternatiivi platvormipõhiste rakenduste kirjutamisele.

React Native kasutab funktsionaalsuse loomiseks JavaScripti ning Reacti teeki. Reacti kasutavad rakendused koosnevad komponentidest (*Component*), mis omavad sisendandmeid (*props*), seisundit (*state*) ning *render* funktsiooni. *Render* funktsioon kasutab XMLiga sarnase süntaksiga JSXi ja tagastab HTMLi. Kasutajaliidese täpsemaks kujundamiseks kasutatakse CSSi sarnast StyleSheet süsteemi [17].

```
class RNative extends Component {
	render() {
		return (
			<NavigatorIOS
				style={styles.wrapper}
				initialRoute={{
					title: 'Tests',
					component: Buttons
				}} />
		);
	}
}
```

Joonis 3. React Native komponent

```
const styles = StyleSheet.create({
	wrapper: {
		flex: 1
	},
		title: {
		fontSize: 20,
		marginBottom: 8,
		textAlign: 'center'
	}
});
```

Joonis 4. React Native rakenduse kasutajaliidese kujundamine

Erinevate platvormide jaoks ei ole üldjuhul vaja kasutajaliideste jaoks eraldi koodi kirjutada, kuid hetkel tuleb seadmespetsiifilise funktsionaalsuse kasutamiseks teha erandeid. Näiteks on iOSi puhul võimalik kasutada etteantud NavigatorIOS komponenti, mis on välimuselt ja funktsionaalsuselt identne iOSi platvormipõhise navigatsiooniribaga [18], kuid Androidi puhul tuleb rohkem vaeva näha ja täiendada Navigator komponendi välimust ja funktsionaalsust [19].

React Native on täielikult asünkroonne, mis tähendab, et kõik tegevused JavaScripti koodi ja platvormi vahel ei jää üksteise taha ootama – kasutajaliidese kuvamine toimub ühel lõimul (*thread*) ja andmete kettale salvestamine või keerukad arvutused toimuvad mõnel teisel lõimul. Tulemuseks on sujuv ja kiirelt reageeriv rakendus [17].

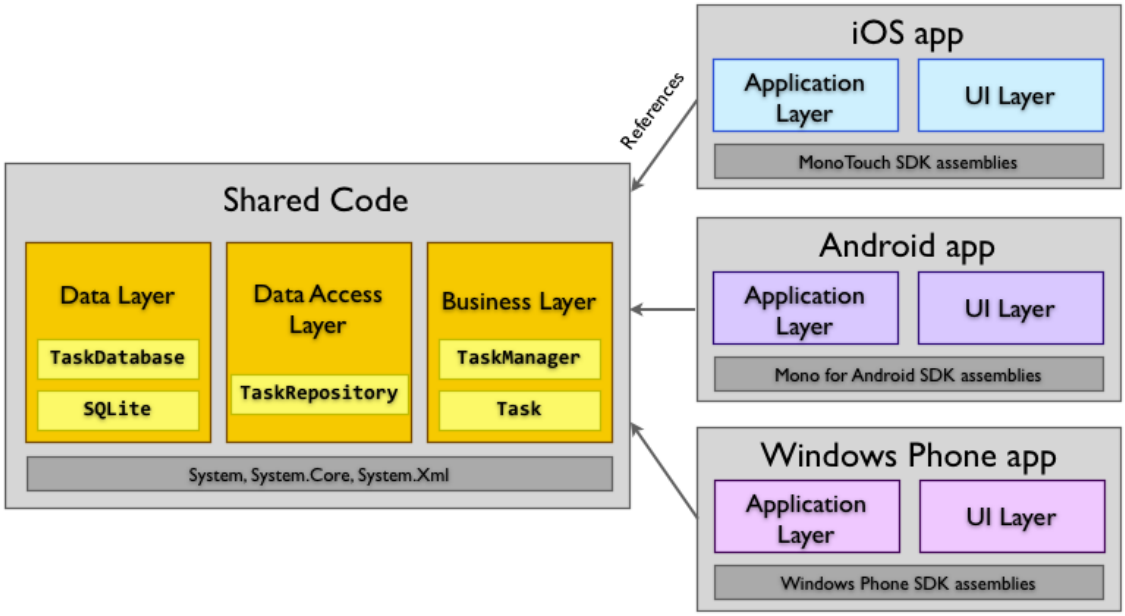
Raamistiku kasutamiseks ametlikku arendusvahendit hetkel ei pakuta, kuid silumiseks (*debugging*) on võimalik kasutada Google Chrome'i arendajavahendeid (*developer tools*). Rakenduse arendamiseks ning silumiseks teeb lihtsamaks ja mugavamaks programmikoodi kiirvärskendamine (*hot reload*). Erinevalt traditsionaalsetest mobiilirakendustest (näiteks Swift ja Java), mida on vaja koodimuudatuste sisseviimise järgselt uuest kompileerida, saab React Native rakendusi vaid osaliselt värskendada. Nii on võimalik hoida alles rakenduse andmed ning seisund, kuid muuta funktsionaalsust [20].

3.3.2 Xamarin

Xamarin on 2016. aastal Microsofti poolt omastatud hübriidrakenduste loomise platvorm. Seda kasutab üle 1,4 miljoni arendaja üle maailma ning on äärmiselt populaarne tegija

oma valdkonnas. Xamarini arenduskeeleks on C# ning arendusvahenditeks on Apple'i Maci puhul Xamarin Studio ning Microsofti Windowsi puhul Visual Studio. Xamarin toetab rakenduste loomist Androidile, iOSile ning Windows Phonele [21]. Xamarini valimise põhjuseks oli populaarsus hübriidrakenduste raamistike seas ning platvormipõhise jõudluse ja kasutajaliidese lubadus loojate poolt [3].

Erinevalt React Nativest tuleb Xamarini puhul luua iga platvormi kasutajaliides eraldi. Platvormide vahel saab jagada loogikat, mis ei kasuta seadmespetsiifilist funktsionaalsust – näiteks suhtlus andmebaasiga või algoritmide realiseerimiseks loodud programmikood.



Joonis 5. Xamarin rakenduse struktuur

Jagatud programmikood koosneb tüüpilise Xamarin rakenduse struktuuri kohaselt kolmest osast:

- Andmekiht (*Data Layer*) hoiab endas koodi, mis tegeleb andmete füüsilise salvestamisega näiteks andmebaasi või failidesse. Androidil ja iOSil on mõlemal sisseehitatud SQLite andmebaasimootor, mis muudab selle põhiliseks valikuks andmete hoidmisel [22].
- Andmete ligipääsukiht (*Data Access Layer*) pakub programmiliidese, mille kaudu saab rakendus teha päringud valitud andmete salvestusmehhanismi või väliste programmiliideste vastu. Ligipääsukihi kaudu saab andmeid luua, salvestada, muuta ja kustutada [23].

- Äriloogika kiht (*Business Layer*) implementeerib mudelid (*Model*), fassaadi (*Façade*) nende haldamiseks ning muud vajalikku üldist rakenduse äriloogikat. Mudelite eesmärgiks on organiseerida andmeid ning pakkuda neid strukturaalselt [23].
- Rakenduse kiht (*Application Layer*) seob jagatud koodi platvormipõhiste klassidega, et oleks võimalik andmeid kasutajale läbi kasutajaliidese kihi (*User Interface Layer*) kuvada [23].
- Kasutajaliidese kiht koosneb platvormipõhistest kasutajaliidese elementidest, mille kaudu rakendused saavad kasutajale andmeid visualiseerida. Siia kuuluvad kõik vaated, pildid ning muud platvormile spetsiifilised elemendid [23].

Programmikoodi valmimisel saab genereerida vastavad APK ja IPA failid ning Android ja iOS hübriidrakendused üles panna platvormidele vastavatesse veebipoodidesse, kust kasutajad saavad rakendusi alla laadida.

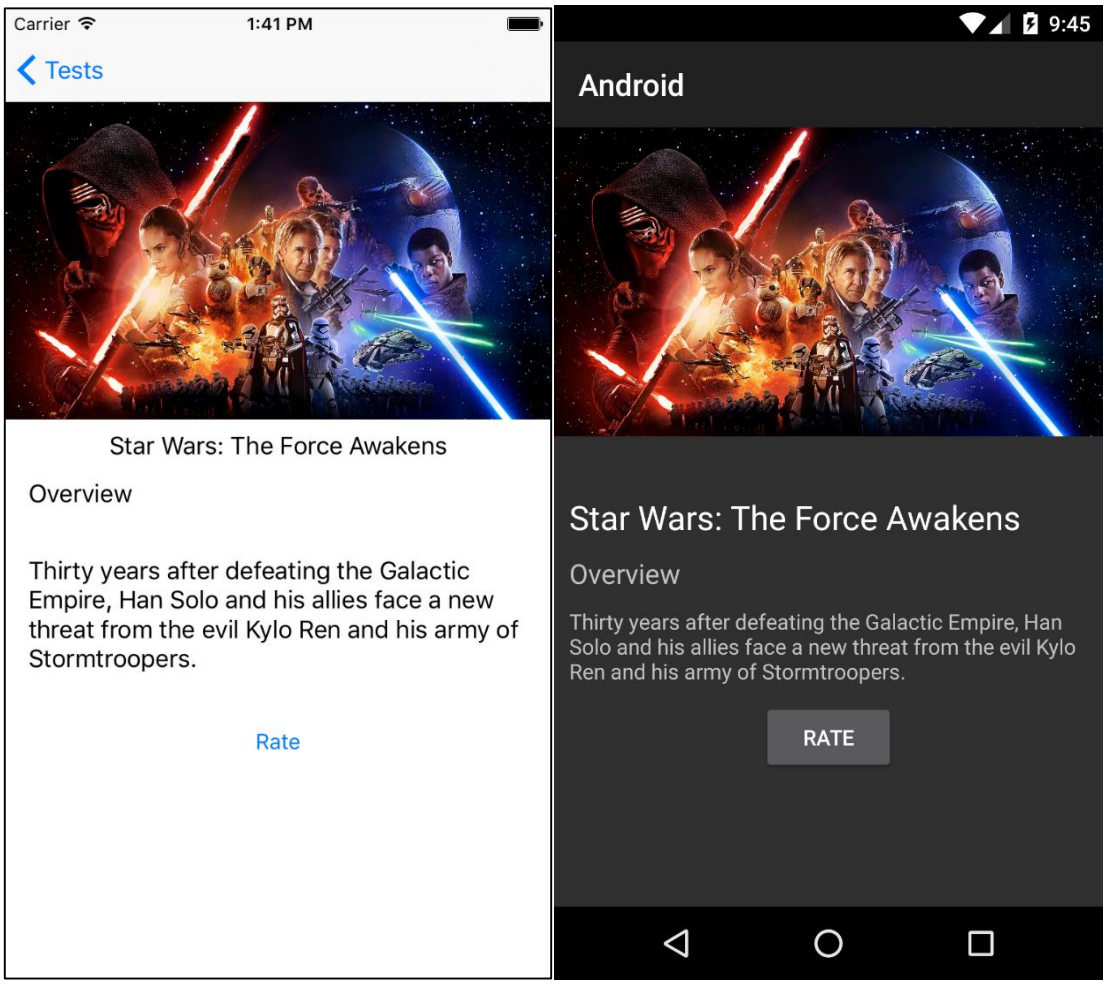
4 Testimine

4.1 Testide kirjeldus

Platvormide võrdlemiseks kasutati nelja erinevat testi – kaks kasutajaliidese ning kaks jõudluse võrdlemiseks. Samuti võrreldi rakenduste poolt kasutatavat sisemälu. Testrakenduse avamisel kuvatakse kasutajale kasutajaliides, millel on neli nuppu. Iga nupp alustab erinevat testfunktsiooni. Kõiki teste käivitati 10 korda ning arvesse võeti kõikide katsete tulemuste aritmeetiline keskmine.

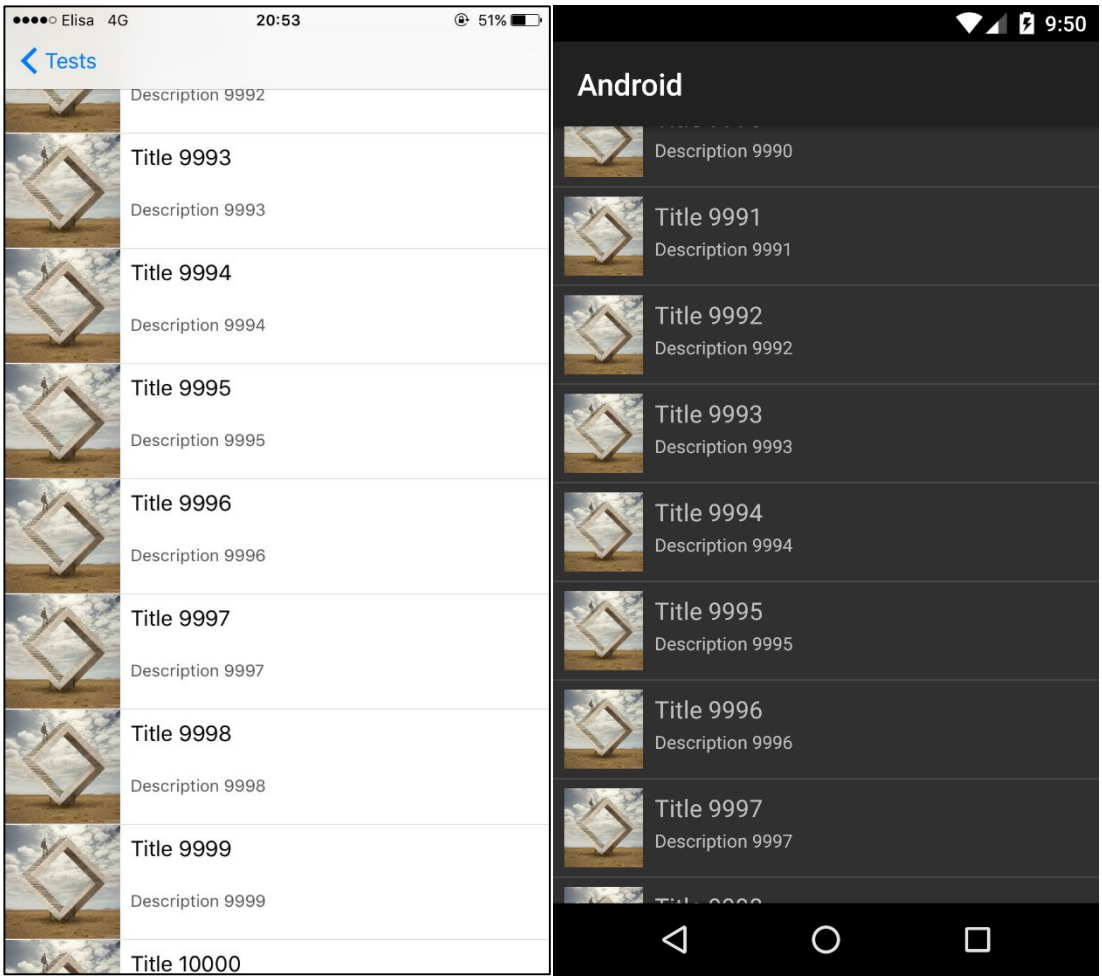
4.1.1 Kasutajaliidese testid

Esimeseks kasutajaliidese testiks oli kahe vaate vahel liikumise võrdlemine. Kasutaja vajutab põhivaatel nuppu, mis viib ta järgmisele vaatele. Seal kuvatakse talle paljudele filmiandmebaasidega seotud rakendustele sarnaselt ühe filmi plakat, pealkiri, kirjeldus ning hinnangu andmise nupp. Aega ja muutmälu kasutust hakatakse mõõtma tegevuse alustamiseks mõeldud nupule vajutamise hetkest ning mõõtmine lõpetatakse vaate täielikul ilmumisel (kõik animatsioonid on lõppenud). Testi eesmärgiks oli võrrelda, kui kiiresti suudavad erinevate vahendite abil loodud rakendused vaadete vahel liikuda ning ressursse ekraanile kuvada.



Joonis 6. iOS ja Android navigatsiooni testi kujundus

Teiseks kasutajaliidese testiks oli paljude andmetega täidetud tabelivaate kuvamine ning tabeli lõppu liikumine. Tüüpiliselt on selline funktsionaalsus foorumi- ja uudisteagregaatorite põhiliseks tegevuseks. Tabel täidetakse kümne tuhande reaga ning kuvatakse kasutajale, kasutades kõiki võimalikke optimeerimise meetodeid. Aega ja muutmälu kasutust hakatakse mõõtma tegevuse alustamiseks mõeldud nupule vajutamise hetkest ning mõõtmine lõpetatakse tabeli lõppu jõudmisel (kõik animatsioonid on lõppenud). Testi eesmärgiks oli näha, kui hästi suudavad erinevad rakendused tabelite elemente taaskasutada ning kas rohkete andmete kuvamine ekraanile aeglustab seadet või suurendab ressursside kasutamist.



Joonis 7. iOS ja Android tabelivaate testi kujundus

4.1.2 Jõudluse testid

Esimeseks jõudluse testiks oli maatriksite korrutamine. Hetkel, mil kasutaja vajutab maatriksite korrutamise testi alustamiseks mõeldud nupule, genereeritakse kaks 500x500 mõõtmetega maatriksit, mis täidetakse suvaliste täisarvudega vahemikus 0 kuni 100. Seejärel korrutatakse maatriksid omavahel. Kulunud aega ja muutmälu kasutust hakatakse mõõtma nupu vajutamise hetkel ning mõõtmine lõpetatakse maatriksite korrutamise tulemuse saavutamisel. Maatriksite suurus valiti maksimumina, millest suuremate arvude puhul hakkasid rakendused muutuma ebastabiilseteks. Tulemuste kinnitamiseks prooviti ka väiksemate maatriksite korrutamist. Testi eesmärgiks oli näha, kui hästi peavad rakendused vastu intensiivsele arvutamisele ning kuidas kannatavad sellise protsessi all seadme ressursid. Selliseid tegevusi võivad sooritada rakendused, mis tegelevad algoritmidega või keeruka matemaatilise loogikaga.

```
private double[][] randomMatrix(int n) {
	double[][] randomMatrix = new double [n][n];
	Random rand = new Random();
	rand.setSeed(System.currentTimeMillis());
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			Integer r = rand.nextInt() % 100;
			randomMatrix[i][j] = Math.abs(r);
		}
	}
	return randomMatrix;
}
```

Joonis 8. Maatriksi genereerimise funktsioon keeles Java

```
private double[][] multiply(double[][] m1, double[][] m2) {
	int matrix1ColumnLength = m1[0].length;
	int matrix2RowLength = m2.length;
	if (matrix1ColumnLength != matrix2RowLength) {
		return null;
	}
	int resultRowLength = m1.length;
	int resultColumnLength = m2[0].length;

	double[][] mResult = new double[resultRowLength][resultColumnLength];
	for (int i = 0; i < resultRowLength; i++) {
		for (int j = 0; j < resultColumnLength; j++) {
			for (int k = 0; k < matrix1ColumnLength; k++) {
				mResult[i][j] += m1[i][k] * m2[k][j];
			}
		}
	}
	return mResult;
}
```

Joonis 9. Maatriksite korrutamise funktsioon keeles Java

Teiseks jõudluse testiks oli binaarsest otsingupuust teatud väärtusega elemendi leidmine. Tegevuse alustamiseks mõeldud nupu vajutamisel luuakse binaarse otsingupuu objekt,

mis täidetakse 5000 järjestikuse väärtusega elemendiga. Seejärel otsitakse puust välja element väärtusega 3674. Valitud elementide arv oli maksimum, mille puhul kõik seadmed funktsioneerisid testi lõpuni. Tulemuste kinnitamiseks prooviti ka väiksemaid elementide arvusid. Testi sooritamiseks kulunud aega ja muutmälu kasutust hakatakse mõõtma eelnimetatud nupu vajutamise hetkel ning mõõtmine lõpetatakse otsingupuust vastava elemendi edukal leidmisel. Testi eesmärgiks oli näha, kui hästi tulevad rakendused toime suurte objektide koguste korral.

```
public Node find(Node node, int value) {
	if (node == null) return null;
	if (value == node.getValue()) {
		return node;
	} else if (value < node.getValue()) {
		if (node.getLeft() == null) return null;
		return find(node.getLeft(), value);
	}
	else if (value > node.getValue()) {
		if (node.getRight() == null) return null;
		return find(node.getRight(), value);
	}
	return null;
}
```

Joonis 10. Elemendi leidmine binaarsest otsingupuust keeles Java

```
public Node add(int value) {
	if (this.getRoot() == null) {
		Node node = new Node(value);
		this.setRoot(node);
		return node;
	}
	return insert(this.getRoot(), value);
}
```

Joonis 11. Elemendi lisamine binaarsesse otsingupuusse keeles Java

```
private Node insert(Node node, int value) {
	if (node == null) return null;
	Node newNode = new Node(value);

	if (value < node.getValue()) {
		if (node.getLeft() == null) {
			node.setLeft(newNode);
			newNode.setParent(node);
			return newNode;
		} else {
			return insert(node.getLeft(), value);
		}
	} else if (value > node.getValue()) {
		if (node.getRight() == null) {
			node.setRight(newNode);
			newNode.setParent(node);
			return newNode;
		} else {
			return insert(node.getRight(), value);
		}
	}
	return null;
}
```

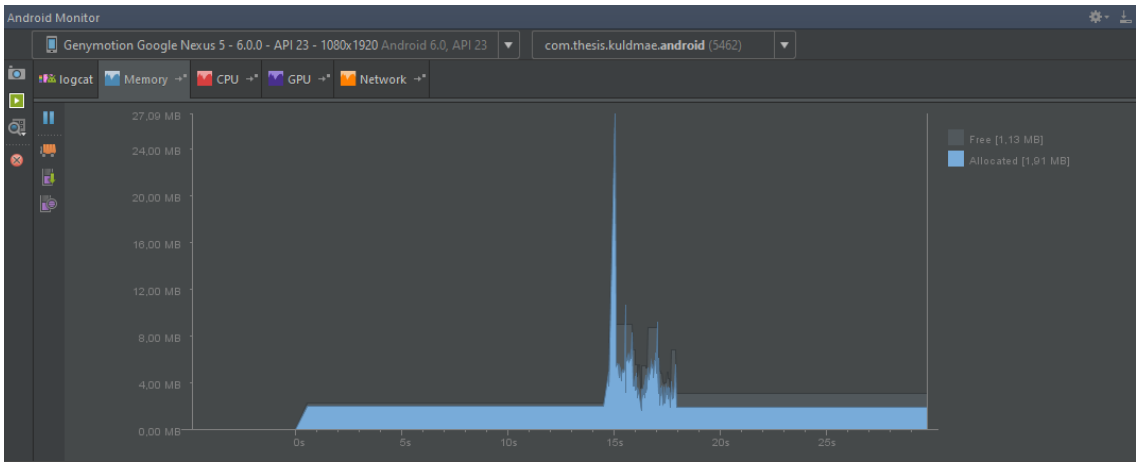
Joonis 12. Rekursiivne elemendi lisamine binaarsesse otsingupuusse keeles Java

4.2 Vahendid

Testide tulemuste mõõtmiseks oli kasutusel operatsioonisüsteemispetsiifilised profileerimisvahendid ning kasutati füüsilisi seadmeid, mitte emulaatoreid. Järgnevalt kirjeldan lähemalt testimisvahendeid.

4.2.1 Android Studio

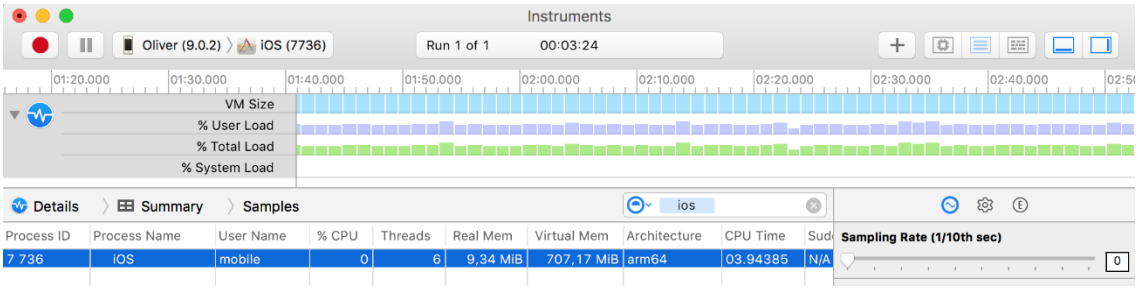
Android Studiol on sissehitatud vahendid, mille abil on võimalik jälgida Android seadmele paigaldatud rakenduste protsesse, nende logi ning ressursside kasutamisega seotud statistikat.



Joonis 13. Android Studio seadme mälukasutuse vaade

4.2.2 Instruments

Instruments on Apple'i poolt loodud jõudluse analüüsimiseks mõeldud tarkvara. Sarnaselt Android Studiole võimaldab Instruments jälgida rakenduste ressursside kasutamist, kuid erinevaid statistikaid on märgatavalt rohkem kui Androidi profileerimisvahendil.



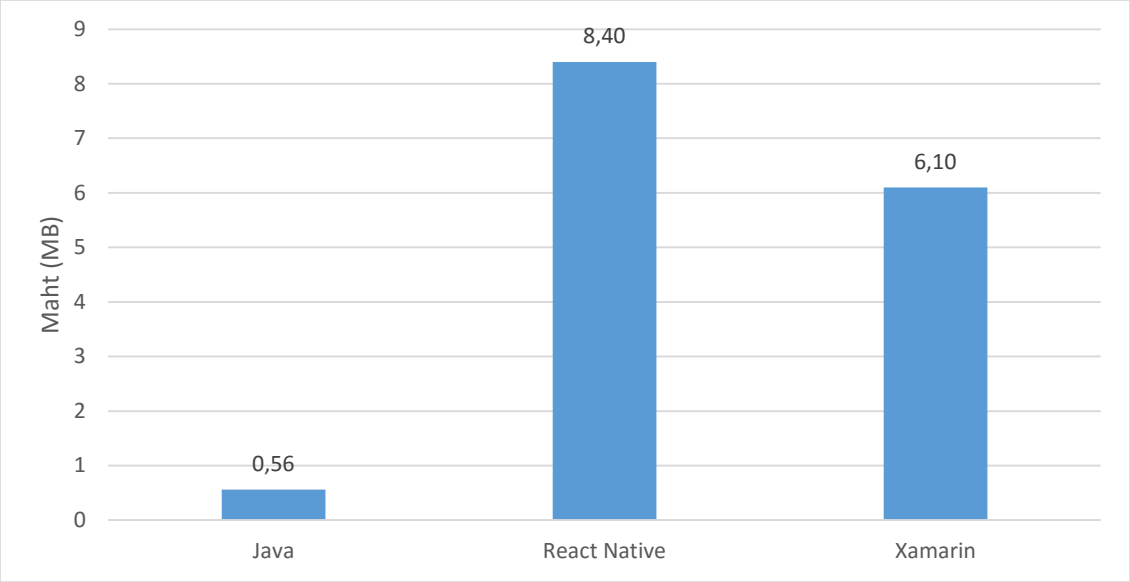
Joonis 14. Instrumentsi protsessi jälgimise vaade

4.2.3 Seadmed

Testimiseks kasutati Apple iPhone 6 16GB, millele oli paigaldatud iOS operatsioonisüsteemi versioon 9.0.2 ning LG Nexus 5, millel Android operatsioonisüsteemi versioon 6.0.1. Rakenduste profileerimiseks kasutati 2013. aasta MacBook Pro 13-tollise Retina ekraaniga sülearvutit.

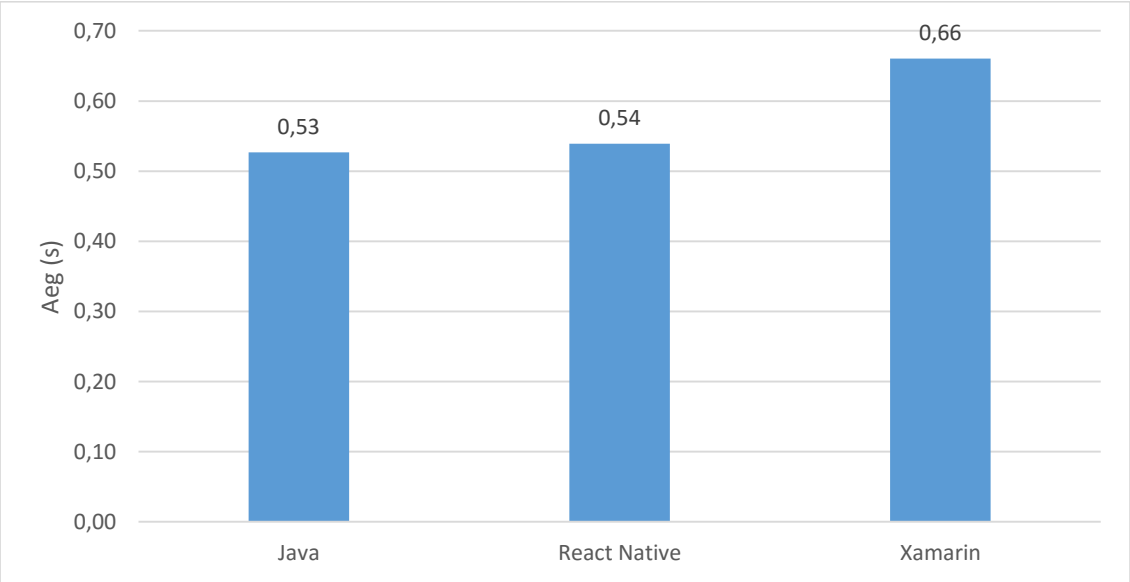
4.3 Tulemused

4.3.1 Android



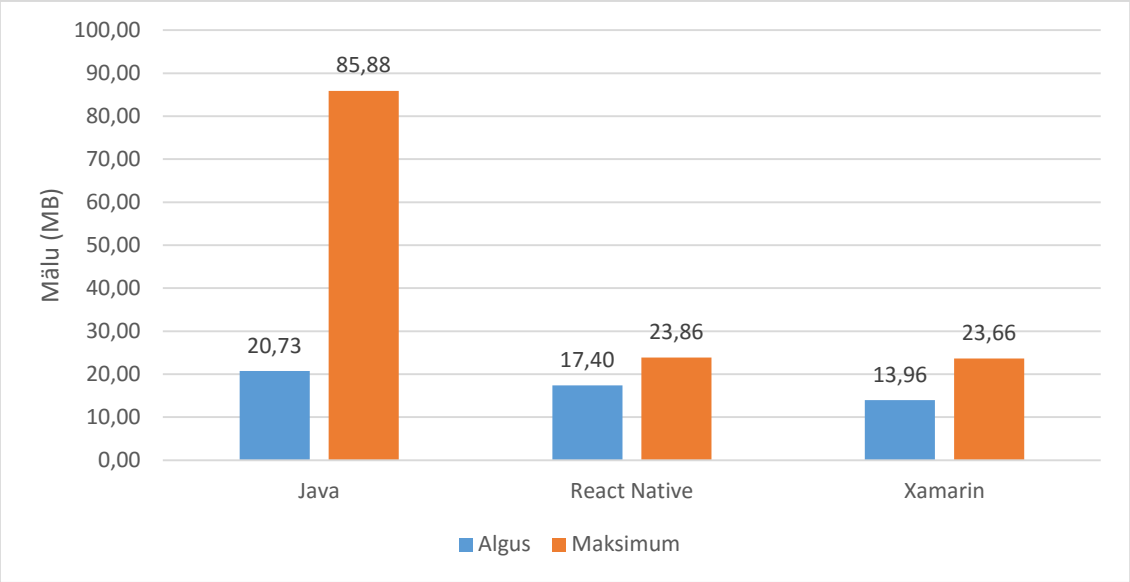
Joonis 15. Sisemälu kasutamine rakenduste poolt Android seadmel

Android seadme sisemälu kasutas ära kõige efektiivsemalt platvormipõhine, mille programmikoodist genereeritud APK arhiiv oli vaid 0,56 MB suurune. React Native ja Xamarin rakendused olid märgatavalt suuremad. Üheks rakenduste suuruse vaheks on see, et hübriidrakendused pakivad kaasa toimimiseks vajalikud teegid, mida platvormipõhiste Java rakenduste puhul pole vaja.



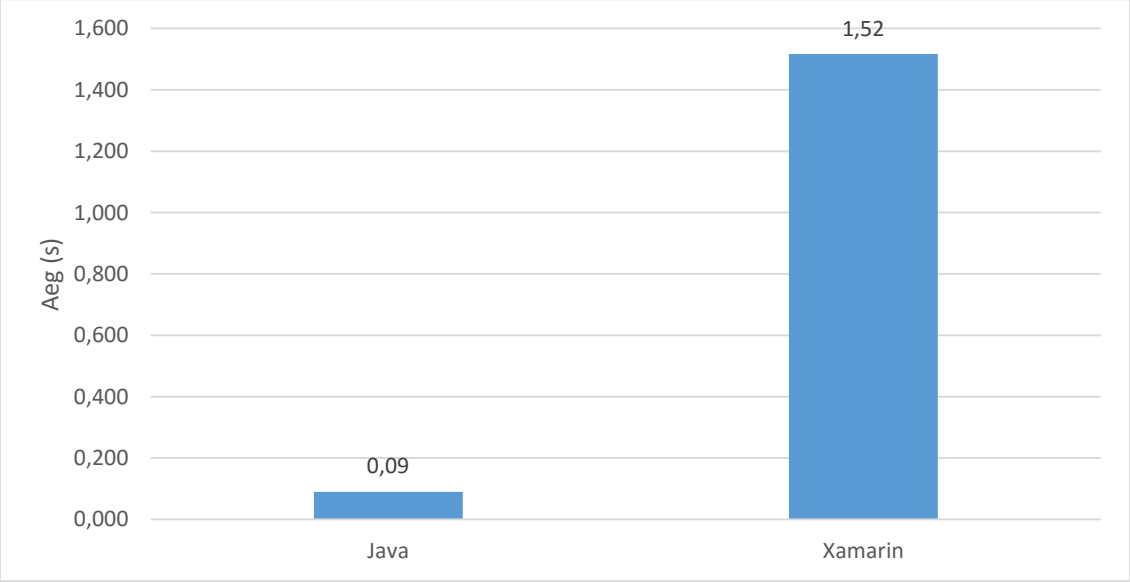
Joonis 16. Navigatsiooni testiks kulunud aeg Android seadmel

Navigeerimisel olid Java ja React Native võrdlemisi sama kiired – platvormipõhine rakendus keskmiselt vaid sajandiku võrra kiirem. Xamarin rakendus oli veidi aeglasem, kuid üldpildis olid kõik rakendused üsna sarnase kiirusega.



Joonis 17. Navigatsiooni testi mälukasutus Android seadmel

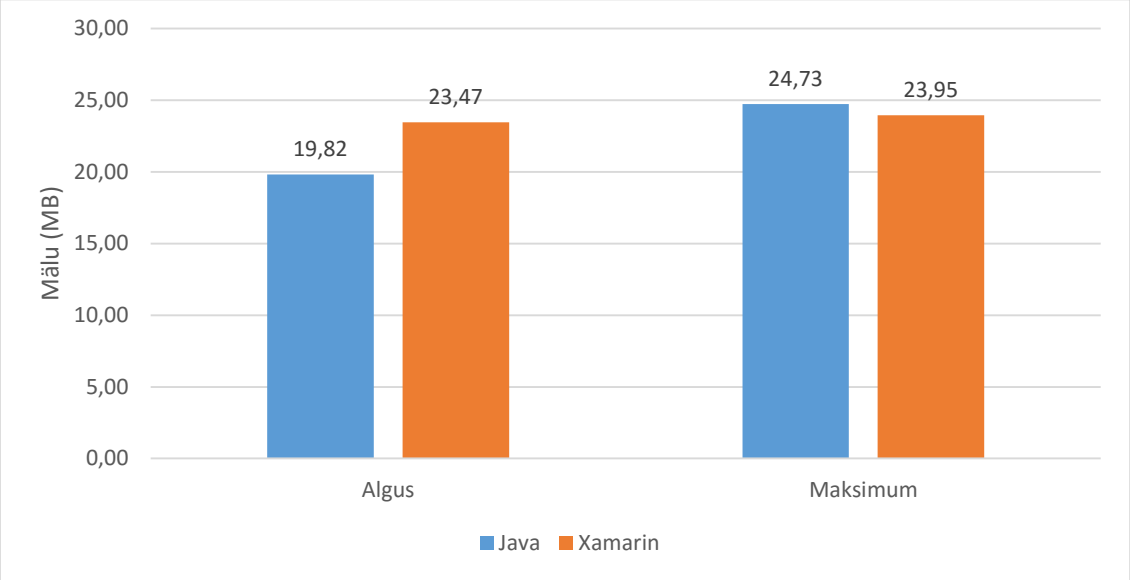
Mälukasutuse puhul on pilt märgatavalt teistsugune. Java platvormipõhine rakendus kasutas rakenduse avamise hetkel kõige rohkem mälu, kuid testi käivitamisel tõusis see näitaja kordades rohkem kui teistel rakendustel.



Joonis 18. Tabelivaate testiks kulunud aeg Android seadmel

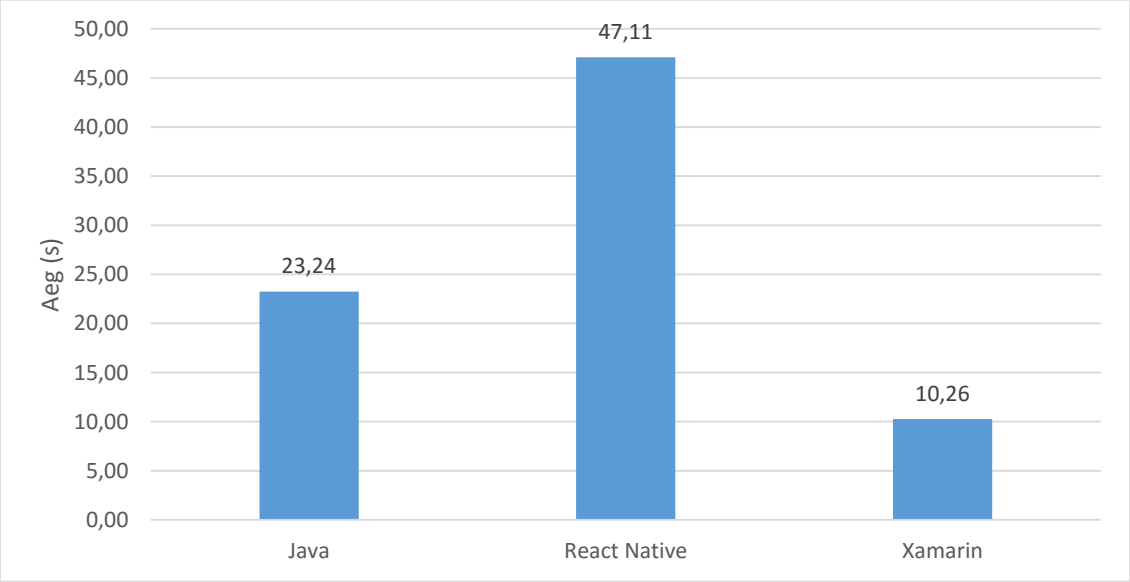
Tabelivaate testi puhul oli ainult kaks võrdlusalust. Arenduse käigus tuli välja, et React Native ei võimalda valida tabelivaatest spetsiifilisel positsioonil olevat elementi ning selleni liikuda. Ainus võimalus oli kuvada välja kõik kümme tuhat elementi ning seejärel

mõõtude järgi liikuda elemendini, mille peale rakendus muutus äärmiselt aeglaseks ning lõpuks sulgus. Androidi puhul on näha, et Java platvormipõhine rakendus on väga kiire tabelivaadete kuvamisel – testi sooritamiseks kulus keskmisel vaid üheksa sajandikku. Xamarini puhul läks märgatavalt kauem, ligikaudu poolteist sekundit. Siin tuleb välja platvormipõhiste rakenduste parem optimeeritus ning tabeli elementide taaskasutamine.



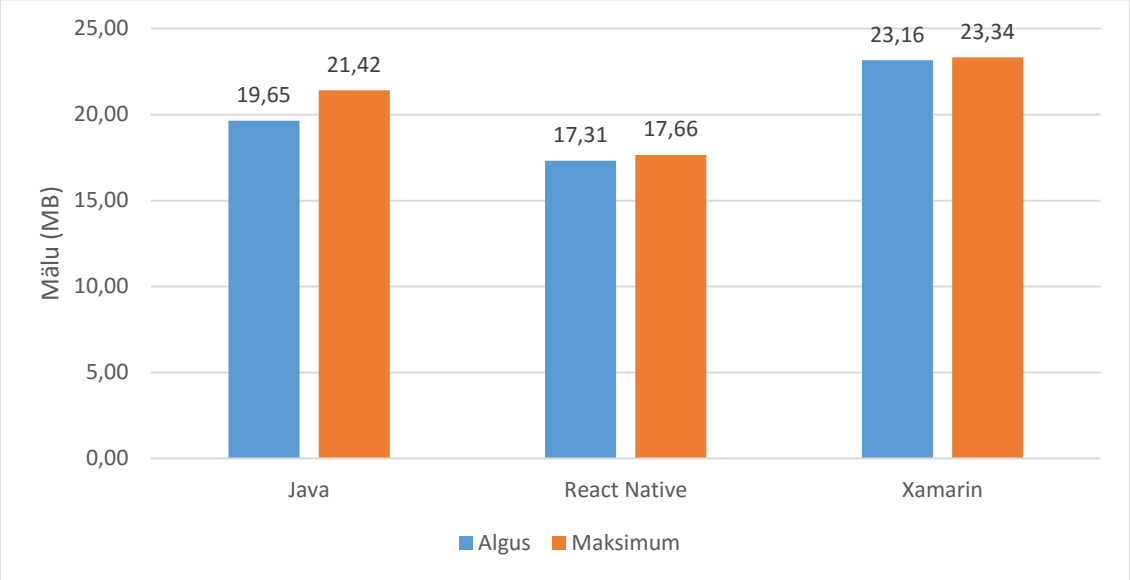
Joonis 19. Tabelivaate testi mälukasutus Android seadmel

Mälukasutuse puhul on rakenduste tulemused sarnasemad. Java puhul on nii rakenduse avamise hetkel kui ka testi lõpuks mälukasutus madalam kui Xamarinil, kuid üllatusena vähendas testi käivitamine Xamarini mälukasutust.



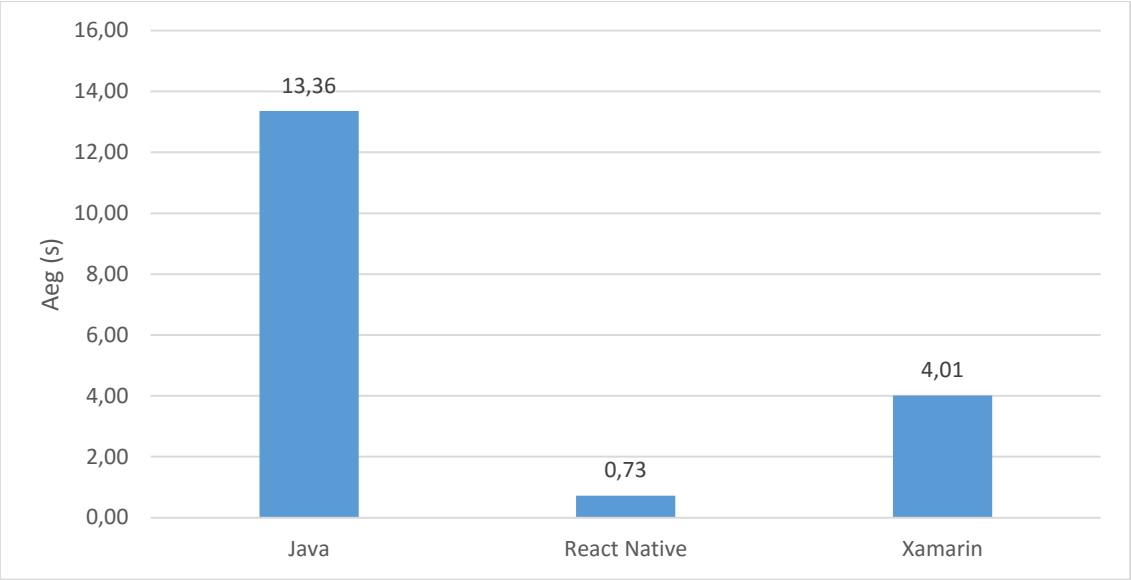
Joonis 20. Maatriksite korrutamise testiks kulunud aeg Android seadmel

Maatriksite korrutamisel oli kiireimaks Xamarin, kulutades testi sooritamiseks 10,26 sekundit. Teistel rakendustel läks margatavalt kauem: platvormipõhisel Java rakendusel kulus 23,24 sekundit ning React Native rakendusel koguni 47,11 sekundit. Trendid säilisid ka 100x100 ja 300x300 maatriksite korrutamisel. Tulemustest võib järeldada, et JavaScripti kasutav React Native ei ole efektiivne intensiivsel arvutamisel.



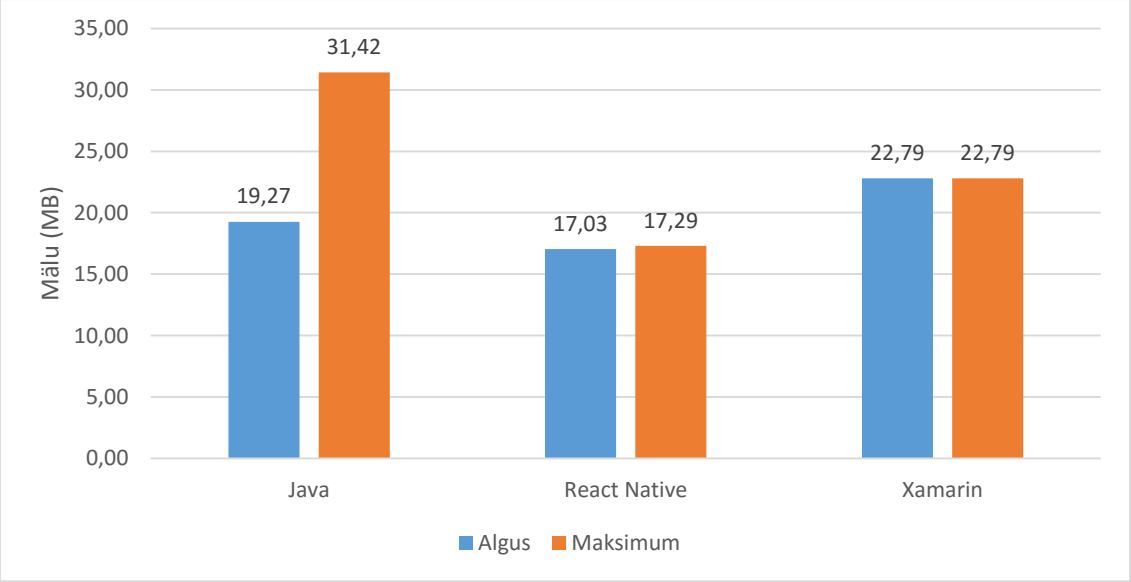
Joonis 21. Maatriksite korrutamise testi mälukasutus Android seadmel

Samas oli aeglaseim rakendus mälukasutuselt kõige ökonoomsem, kasutades rakenduse avamise hetkel 17,31 MB ja maksimaalselt 17,66 MB muutmälu. Kiireim rakendus oli ebaökonoomseim, kasutades esialgu 23,16 MB ja maksimaalselt 23,34 MB muutmälu. Java rakendus kasutas algul 19,65 MB muutmälu ning maksimaalselt 21,42 MB muutmälu. Graafikud jäid sarnasteks ka väiksemate maatriksite korrutamisel.



Joonis 22. Binaarse otsingupuu testiks kulunud aeg Android seadmel

Androidi puhul oli kiireim binaarsest otsingupuust elemendi leidmise testi sooritaja React Native, kulutades keskmiselt vaid 0,73 sekundit. Xamarinil kulus testi sooritamiseks 4,01 sekundit, kuid platvormipõhisel Java rakendusel kulus teistega võrreldes märgatavalt rohkem aega – 13,36 sekundit. Käivitades testi tuhande ja kolme tuhande elemendiga ilmnesid samad trendid. See võib tähendada, et Java ei tule rekursiivsete funktsioonide või suurte objektide hulgaga sama hästi toime kui konkurendid.

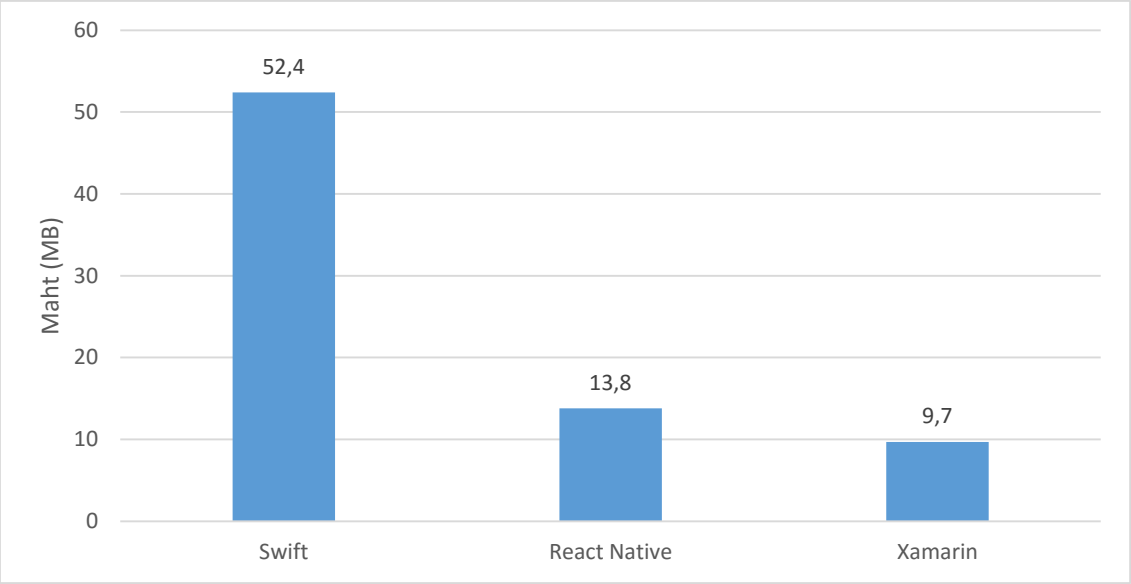


Joonis 23. Binaarse otsingupuu testi mälukasutus Android seadmel

Mälukasutuselt oli Java ebaökonoomseim, kasutades esialgu 19,27 MB ning maksimaalselt 31,42 MB seadme muutmälu. Xamarin on ka siin kategoorias teine, kasutades rakenduse avamise hetkel ning ka maksimaalselt 22,79 MB muutmälu.

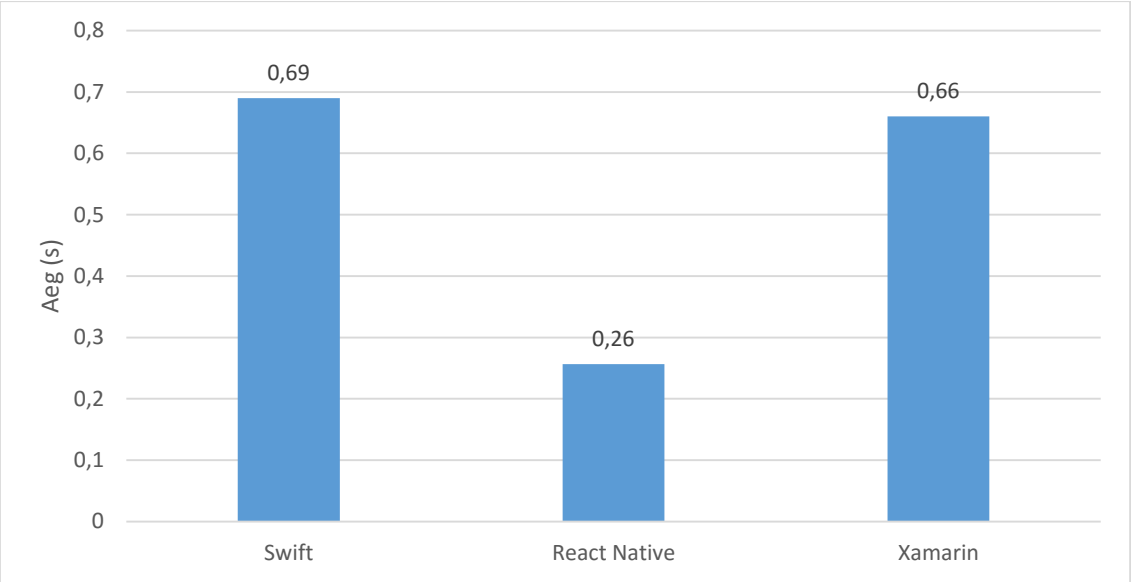
Xamarini puhul testi käivitamine mälukasutust ei muutnud. React Native oli kiireim ning ka mälukasutuse poolest parim, kasutades algul 17,03 MB ning maksimaalselt 17,29 MB muutmälu. Java puhul tõusis testi käivitamisel rakenduse mälukasutus 63% võrra, kui Xamarini puhul sama näitaja ei muutunud ning React Native kasutas ainult 0,26 MB rohkem mälu. Väiksemate elementide arvuga testi käivitades jäid graafikud sarnasteks. Mälukasutuse suur hüpe ning ka märgatavalt aeglasem testi soorituse aeg viitavad probleemile objektide hoidmisel ning lahtilaskmisel mälust.

4.3.2 iOS



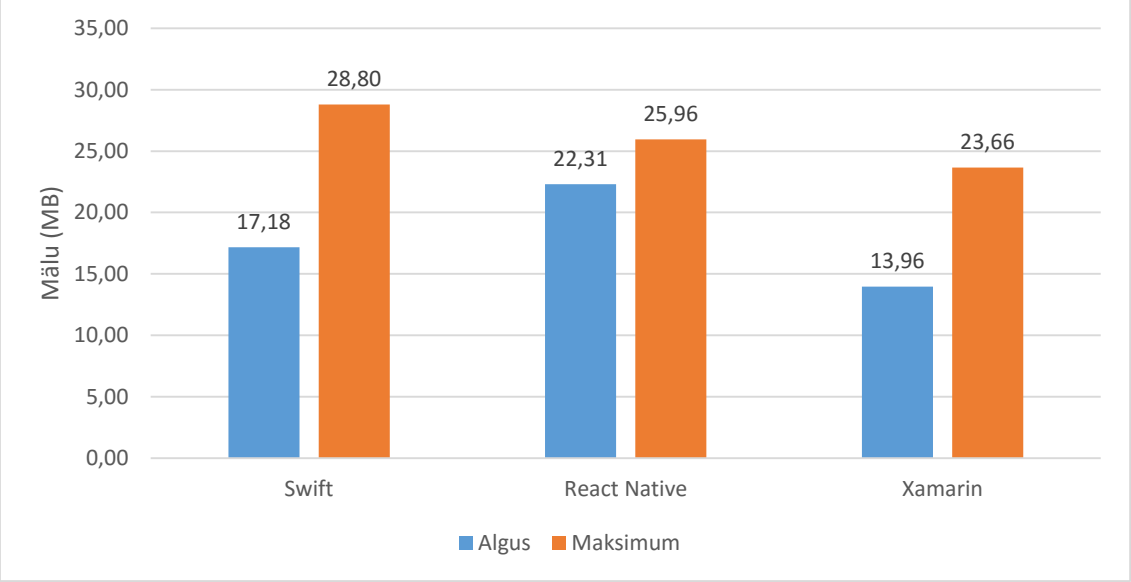
Joonis 24. Sisemälu kasutamine rakenduste poolt iOS seadmel

iOSi puhul oli sisemälu kasutuse poolest kõige parem React Native, kasutades ära 8,2 MB seadme sisemälust. Xamarin oli vaid veidi kehvem, kasutades 9,7 MB sisemälu. Platvormipõhine rakendus keeles Swift oli kõige mahukam, kasutades ära 52,4 MB mälu, mis on kordades rohkem teistest rakendustest. Rakenduse sisu uurides tuli välja, et 43,1 MB mahust võtab fail libswiftCore.dylib. iOS arendajate juhendi järgi on Xcode versioonist 7.0.1 alates võimalik bittkoodi suurust App Store'i üleslaadimise järel vähendada [24], mistõttu veebipoest allalaetav rakendus võib muutuda märgatavalt väiksemaks.



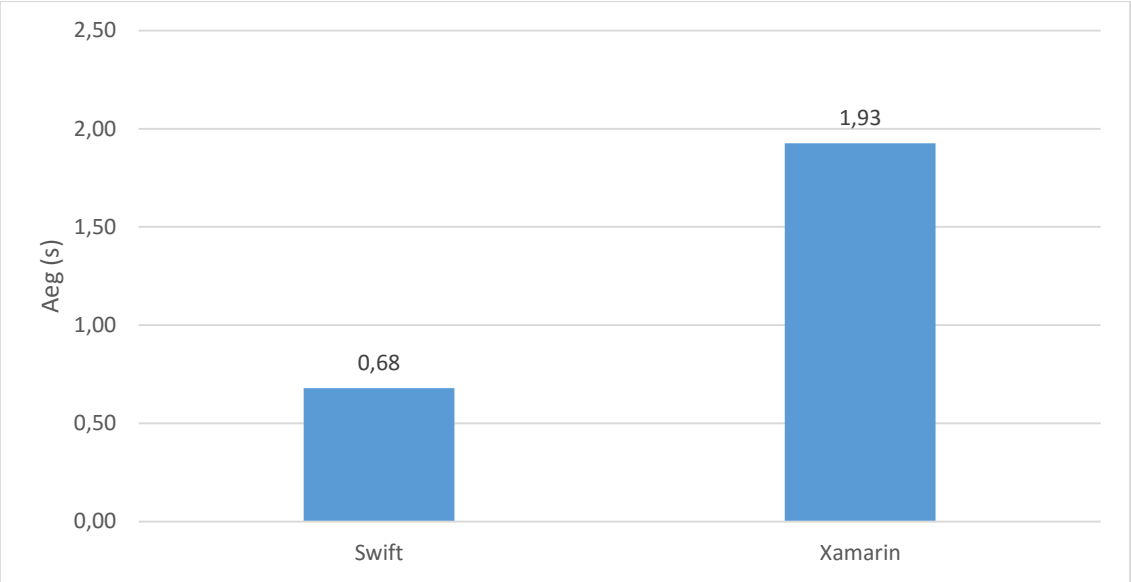
Joonis 25. Navigatsiooni testiks kulunud aeg iOS seadmel

Navigeerimise testis oli kõige kiiremaks rakenduseks React Native hübriidrakendus. Ühelt vaatelt teisele liikumiseks kulus vaid 0,26 sekundit. Xamarin ja Swift rakendused olid peaaegu sama kiired, Xamarin oli kolme sajandiku võrra kiirem. React Native'i puhul oli tuntav animatsioonide kiiruse vahe, mistõttu oligi näitaja sedavõrd madal.



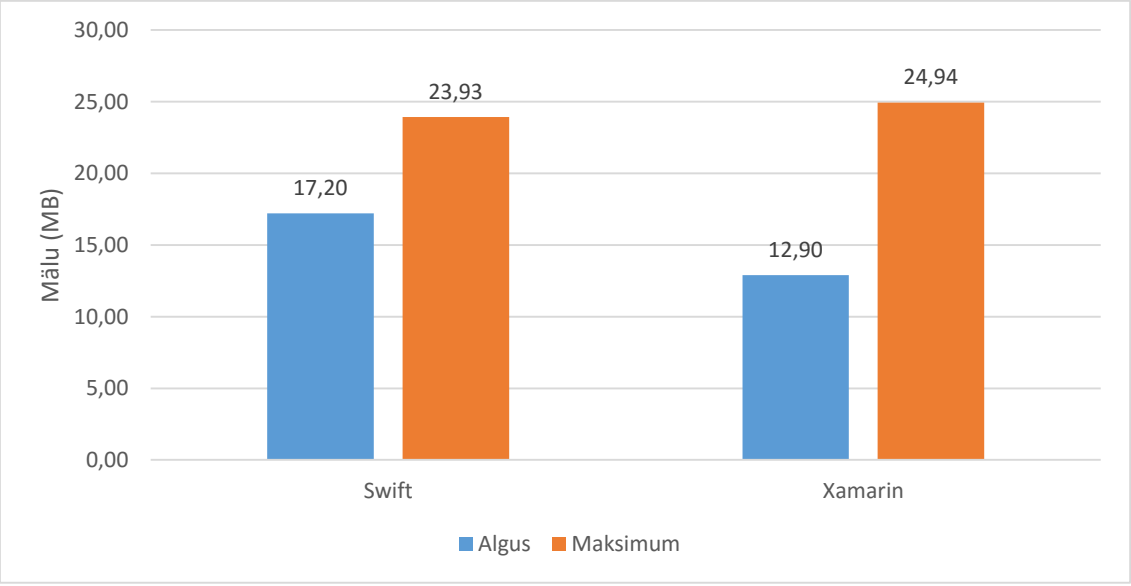
Joonis 26. Navigatsiooni testi mälukasutus iOS seadmel

Maksimaalse mälukasutuse poolest oli parim Xamarin ning Swift kõige kehvem, kasutades ära 28,80 MB muutmälu. Kõige väiksem muutus mälukasutuses oli React Nativel, mis võib tuleneda erinevatest ressursside haldamise loogikast.



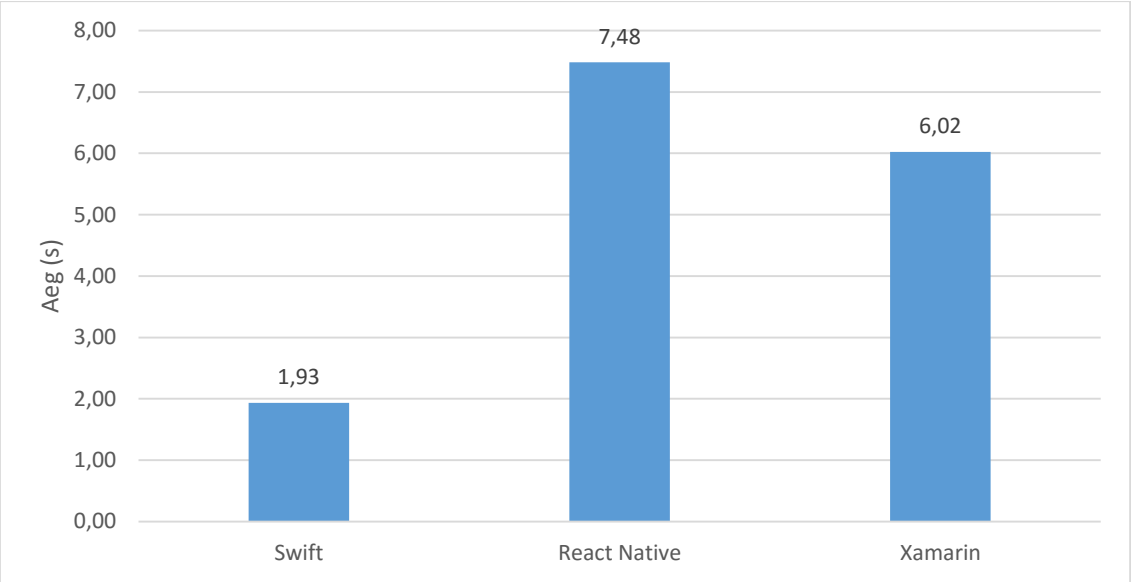
Joonis 27. Tabelivaate testiks kulunud aeg iOS seadmel

Sarnaselt Androidile ei olnud selle testi puhul võimalik mõõta React Native'i jõudlust. Jällegi tuleb Swifti ja Xamarini vahel välja platvormipõhise rakenduse optimeeritus elementide taaskasutamisel. Swiftis kirjutatud rakendus suutis tabeli viimase elemendi kuvada välja 0,68 sekundiga, kui Xamarinil läks sama tegevuse peale 1,93 sekundit.



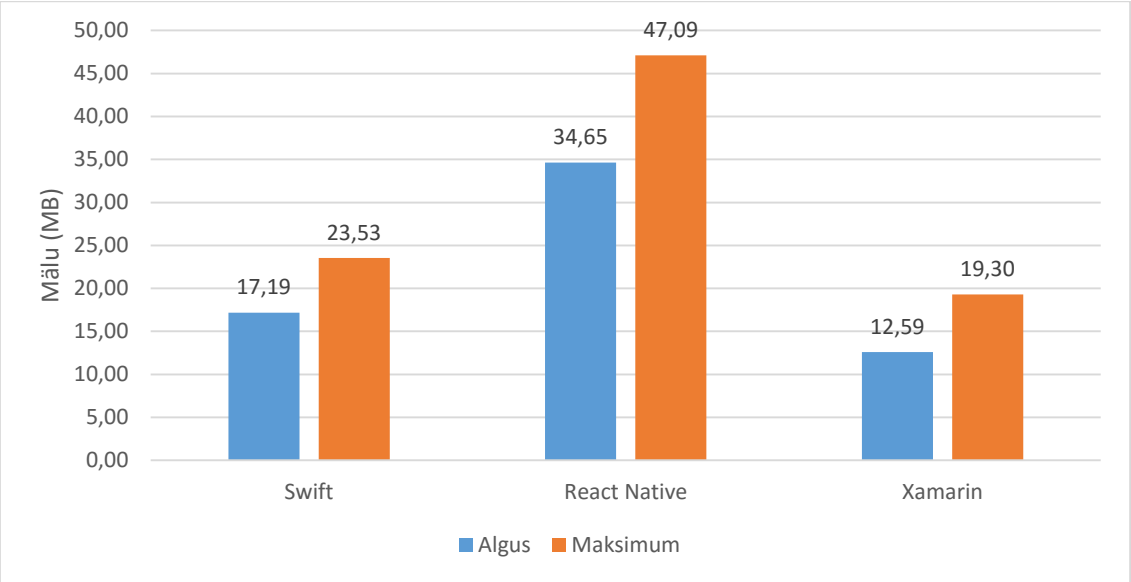
Joonis 28. Tabelivaate testi mälukasutus iOS seadmel

Algne mälukasutus oli Swiftil kõrgem, kuid maksimaalne mälukasutus madalam. Samuti oli suhteline mälukasutuse muutus oli Swifti puhul väiksem. Seega oli tabelivaate kuvamine Xamarini rakenduse puhul ressursside poolest kulukam, kui Swifti puhul.



Joonis 29. Maatriksite korrutamiseks kulunud aeg iOS seadmel

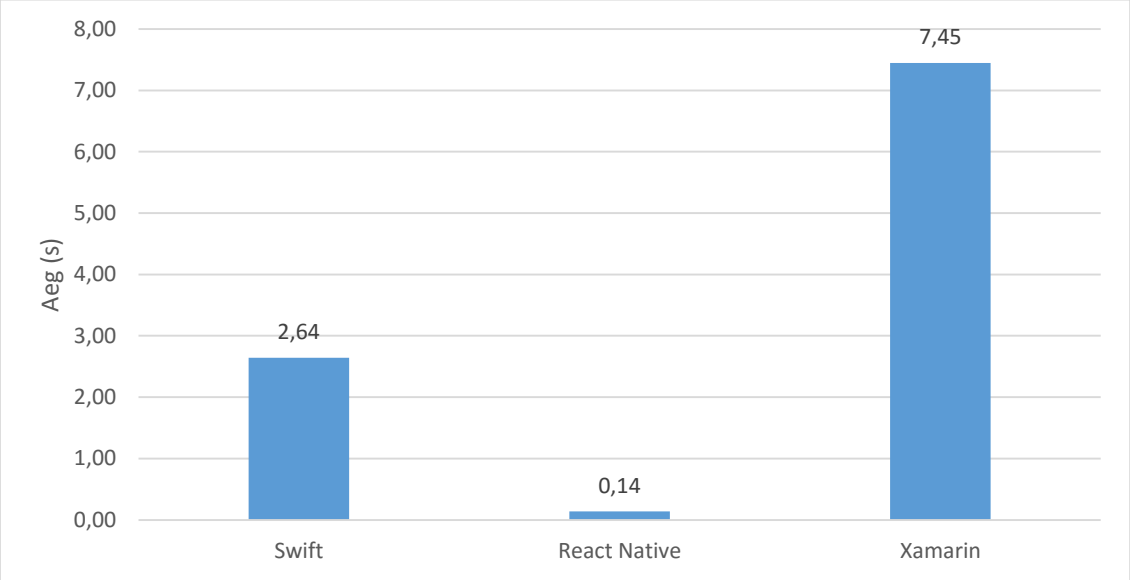
Maatriksite korrutamisel oli Apple'i operatsioonisüsteemil kõige kiiremaks platvormipõhine rakendus keeles Swift, kulutades testi sooritamiseks vaid 1,93 sekundit. Xamarini ja React Nativet kasutavatel rakendustel läks märgatavalt kauem aega, kulutades vastavalt 6,02 ja 7,48 sekundit testi sooritamiseks. 100x100 maatriksite korral oli trend samasugune, kuid üllatusena oli React Native 300x300 maatriksite korrutamisel ligi poole kiirem kui Xamarin.



Joonis 30. Maatriksite korrutamise testi mälukasutus iOS seadmel

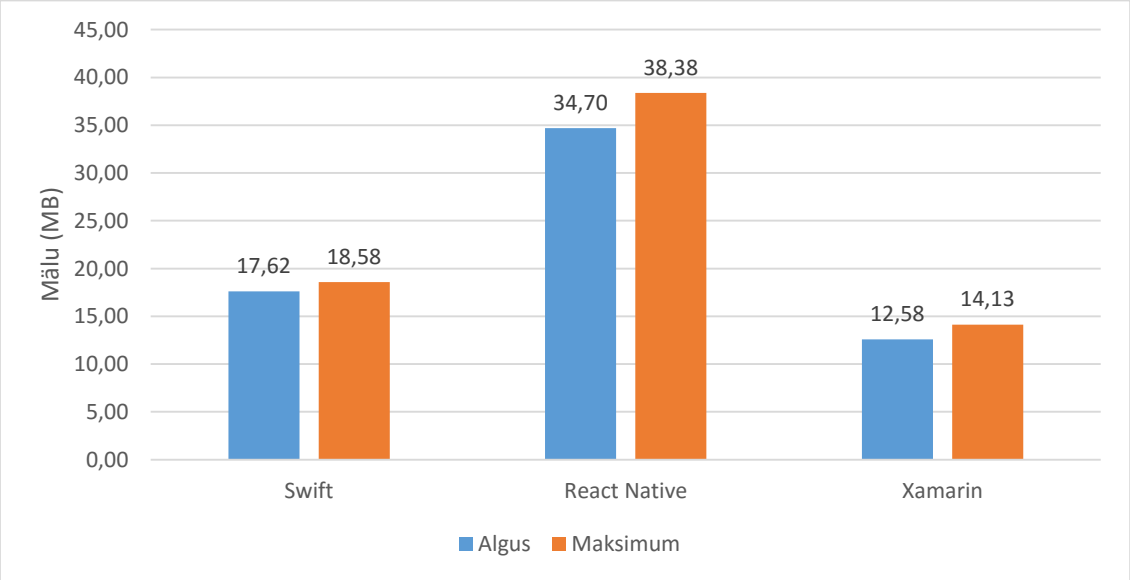
Kõige madalam muutmälu kasutus oli Xamarinil, kasutades rakenduse avamise hetkel 12,59 MB ning maksimaalselt 19,30 MB muutmälu. Swiftis kirjutatud rakendus kasutas esialgu 17,19 MB ning maksimaalselt 23,53 MB mälu. React Native raamistikku kasutav

rakendus oli kõige ebaökonoomsem, kasutades algul 34,65 MB ning maksimaalselt 47,09 MB muutmälu. Parameetrite muutmisel jäid mälukasutuse trendid samasugusteks.



Joonis 31. Binaarse otsingupuu testiks kulunud aeg iOS seadmel

Binaarse otsingupuu testi sooritas kõige kiiremini React Native ning tegi seda väga kiiresti – keskmiselt kulus aega vaid 0,14 sekundit. Swiftis kirjutatud rakendusel kulus 2,64 sekundit ning Xamarini kasutaval rakendusel kulus 7,45 sekundit. Käivitades testi tuhande elemendi sisestamisega, oli React Native siiski kiireim, kuid vahe Xamarini ja Swifti vahel oli märgatavalt väiksem. Kolme tuhande elemendiga tegeledes oli Xamarini ja Swifti ajavahe juba ligi kahekordne.



Joonis 32. Binaarse otsingupuu testi mälukasutus iOS seadmel

Muutmälu kasutamise poolest oli kõige ökonoomsem jälle Xamarin, kasutades algul 12,58 MB ning maksimaalselt 14,13 MB seadme muutmälu. Platvormipõhisel rakendusel kulus rakenduse avamise hetkel 17,62 MB muutmälu ning maksimaalne mälukasutus oli 18,58 MB. React Native kasutas teistest rakendustest märgatavalt rohkem mälu, algul 34,70 MB ja maksimaalselt 38,38 MB. Tuhande ja kolme tuhande elemendiga testi käivitamisel jäid trendid samasuguseks.

Kokkuvõte

Käesoleva töö eesmärgiks oli analüüsida ja võrrelda hübriid- ja platvormipõhiste mobiilirakenduste jõudlust.

Töö esimeses pooles tutvustati mobiilseid operatsioonisüsteeme, mobiilirakendusi ja nende erinevaid tüüpe. Töö teises pooles käsitleti jõudluse testimist ning nimetatud testide tulemusi.

Testide tulemustest oli näha, et hübriidrakendustel on sobivaid ja mittesobivaid otstarbeid. Ühtne koodibaas võimaldab kirjutada mobiilirakendusi kiiremini, kuid ei pruugi võimaldada sama funktsionaalsust, mis on platvormipõhistel rakendustel.

React Native, olles kõige uuem ja unikaalsem raamistik testimisaluste seas, näitas puudujääke intensiivsel arvutamisel ning funktsionaalsuse puudulikkust võrreldes konkurentidega. Samas oli React Native kiireim suurte objektidega hulgaga tegelemisel. Väide, et hübriidrakendused suudavad pakkuda platvormipõhistele rakendustele sarnast jõudlust peab kindlasti paika. Teatud juhtudel on hübriidrakendused märgatavalt efektiivsemad platvormipõhistest konkurentidest.

Antud tööd on kindlasti võimalik täiendada ning edasi arendada. Teste saaks optimeerida ning muuta vastavalt seadmete ja programmeerimiskeelte omapäradele. Samuti saaks sügavamalt analüüsida testide tulemusi ja nende tagamaid. Lähtudes Micro to Mainframe testimismetoodikast, saaks luua täiendavaid teste, mis võimaldaksid analüüsida hübriid- ja platvormipõhiste rakenduste teisi külgi. Testida saaks näiteks andmebaasiga suhtlemist, seadmespetsiifilise riistvara kasutamist ja 3D-animatsioonide kujutamise jõudlust.

Summary

The goal of this thesis was to analyze and compare the performance of hybrid and native mobile applications.

The first part of this thesis introduced mobile operating systems, mobile applications and different types of said applications. The second half described the process and results of performance testing.

From the results of the performance tests, it is apparent that hybrid mobile applications have their advantages and shortcomings. A common code base allows developers to write applications faster, but this may come at the expense of less functionality than native applications.

The newest and most unique framework among the test subjects, React native, showed functional shortcomings and poor performance when running intense computation tasks. However, it was the fastest application when it came to dealing with large amounts of objects. The statement “Hybrid applications can offer performance similar to native applications” definitely holds true. In some cases, hybrid applications are actually much more efficient than their native counterparts.

This thesis is definitely possible to improve upon and develop even further. The performance tests used could be optimized and altered to take advantage of device and programming language specifics. The performance testing results and the reasons behind them could also be further analyzed. Taking into consideration the testing methodology created by Micro to Mainframe software testing company, it would be possible to create additional tests, which would help analyze other aspects of hybrid and native mobile applications. For example, database communication, device-specific hardware usage and 3D animation performance could be tested.

Kasutatud kirjandus

[1] Gartner, Inc., „Gartner Says Emerging Markets Drove Worldwide Smartphone Sales to 15.5 Percent Growth in Third Quarter of 2015," Gartner, Inc., 18 November 2015. [Võrgumaterjal]. Available: https://www.gartner.com/newsroom/id/3169417. [Kasutatud 15 Mai 2016].

[2] Ericsson, Inc., „Ericsson Mobility Report," November 2015. [Võrgumaterjal]. Available: http://www.ericsson.com/res/docs/2015/mobility-report/ericsson-mobility-report-nov-2015.pdf. [Kasutatud 20 Mai 2016].

[3] Xamarin, Inc., „Xamarin Platform," 2016. [Võrgumaterjal]. Available: https://www.xamarin.com/platform. [Kasutatud 20 Mai 2016].

[4] AppDynamics, Inc., „The App Attention Span," 2014. [Võrgumaterjal]. Available: http://info.appdynamics.com/rs/appdynamics/images/App%20Attention%20Span%20research%20report%20-%20final.pdf. [Kasutatud 10 Mai 2016].

[5] J. d. Plessis, „Performance Testing Methodology," 2008. [Võrgumaterjal]. Available: https://www.cmcrossroads.com/sites/default/files/article/file/2013/XUS206223565file1_0.pdf. [Kasutatud 10 Mai 2016].

[6] International Data Corporation, „Smartphone OS Market Share, 2015 Q2," [Võrgumaterjal]. Available: https://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Kasutatud 15 Mai 2016].

[7] B. Elgin, „News Analysis," 17 August 2005. [Võrgumaterjal]. Available: http://www.webcitation.org/5wk7sIvVb. [Kasutatud 15 Mai 2015].

[8] R. Williams, „Apple iOS: a brief history," 17 September 2015. [Võrgumaterjal]. Available: http://www.telegraph.co.uk/technology/apple/11068420/Apple-iOS-a-brief-history.html. [Kasutatud 15 Mai 2016].

[9] Google, Inc., „Android SDK," [Võrgumaterjal]. Available: https://developer.android.com/sdk/index.html. [Kasutatud 15 Mai 2016].

[10] Stack Overflow, „Developer Survey Results 2016," [Võrgumaterjal]. Available: https://stackoverflow.com/research/developer-survey-2016. [Kasutatud 15 Mai 2016].

[11] Google, Inc., „Application Fundamentals," [Võrgumaterjal]. Available: https://developer.android.com/guide/components/fundamentals.html. [Kasutatud 15 Mai 2016].

[12] Apple, Inc., „Swift Has Reached 1.0," 9 September 2014. [Võrgumaterjal]. Available: https://developer.apple.com/swift/blog/?id=14. [Kasutatud 15 Mai 2016].

[13] Swift, „Swift.org and Open Source," [Võrgumaterjal]. Available: https://swift.org/about/#swiftorg-and-open-source. [Kasutatud 15 Mai 2016].

[14] Apple, Inc., „The App Life Cycle,“ [Võrgumaterjal]. Available: https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/TheAppLifeCycle/TheAppLifeCycle.html#//apple_ref/doc/uid/TP40007072-CH2-SW1. [Kasutatud 15 Mai 2016].

[15] Apple, Inc., „Expected App Behaviors,“ [Võrgumaterjal]. Available: https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/ExpectedAppBehaviors/ExpectedAppBehaviors.html#//apple_ref/doc/uid/TP40007072-CH3-SW2. [Kasutatud 15 Mai 2016].

[16] D. Witte ja P. von Weitershausen, „React Native for Android: How we built the first cross-platform React Native app,“ 14 September 2015. [Võrgumaterjal]. Available: https://code.facebook.com/posts/1189117404435352/react-native-for-android-how-we-built-the-first-cross-platform-react-native-app/. [Kasutatud 15 Mai 2016].

[17] Facebook, Inc., „React Native - A Framework For Building Native Apps Using React,“ 2016. [Võrgumaterjal]. Available: https://facebook.github.io/react-native/. [Kasutatud 17 Mai 2016].

[18] Facebook, Inc., „NavigatorIOS,“ 2016. [Võrgumaterjal]. Available: https://facebook.github.io/react-native/docs/navigatorios.html. [Kasutatud 17 Mai 2016].

[19] Facebook, Inc., „Navigator,“ 2016. [Võrgumaterjal]. Available: https://facebook.github.io/react-native/docs/navigator.html. [Kasutatud 17 Mai 2016].

[20] M. Bigio, „Introducing Hot Reloading,“ 24 Märts 2016. [Võrgumaterjal]. Available: https://facebook.github.io/react-native/blog/2016/03/24/introducing-hot-reloading.html. [Kasutatud 17 Mai 2016].

[21] Xamarin, Inc., „The Xamarin story.,“ [Võrgumaterjal]. Available: https://www.xamarin.com/about. [Kasutatud 15 Mai 2016].

[22] Xamarin, Inc., „Cross-Platform Data Access,“ 2016. [Võrgumaterjal]. Available: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/data. [Kasutatud 17 Mai 2016].

[23] Xamarin, Inc., „Case Study: Tasky,“ 2016. [Võrgumaterjal]. Available: https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/case_study-tasky/. [Kasutatud 17 Mai 2016].

[24] Apple, Inc., „App Thinning (iOS, tvOS, watchOS),“ 29 Aprill 2016. [Võrgumaterjal]. Available: https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/AppThinning/AppThinning.html. [Kasutatud 17 Mai 2016].