Универзитет у Београду
Математички факултет

Милош Милаковић

Упоредна анализа радних оквира за израду мобилних апликација за Андроид

Мастер рад

Ментор: **др Саша Малков**

Београд,
2017.

Ментор:

др Саша Малков
Математички факултет
Универзитет у Београду

Чланови комисије:

др Владимир Филиповић
Математички факултет
Универзитет у Београду

мр Јелена Хаци Пурић
Математички факултет
Универзитет у Београду

Датум одбране:

Садржај

Глава 1

Увод

1.1 Основни појмови

Паметни телефони су мобилни телефони са интегрисаним функционалностима које оригинално нису биле својствене обичним телефонима као што су поседовање оперативног система, претраживање интернета и извршавање апликација специјално развијених за њих.

Мобилне апликације су софтверске апликације специјално развијене за извршавање на уређајима као што су паметни телефони и таблет рачунари. Мобилне апликације се развијају уз разматрање захтева и ограничења уређаја, али и тако да се на најбољи могући начин искористе њихове могућности и предности.

Мобилне апликације се грубо могу поделити у три врсте:

- Стандардне (енг. *native*) апликације
- Мобилне веб (енг. *Web-based*) апликације
- Хибридне (енг. hybrid) апликације

Стандардне мобилне апликације представљају апликацијске програме који су развијени за употребу на одређеној платформи или уређају. Углавном се развијају

тако да на најбољи начин искористе предности оперативних система, као и хардверских компоненти доступних на уређајима за које се развијају. Карактеристика им је и да је углавном јако тешко апликацију написану за једну платформу пребацити на другу а да се програмски код не промени. Најзаступљеније платформе за израду нативних мобилних апликација су оперативни системи Андроид (енг. *Android*) и *iOS*).

Мобилне веб апликације представљају апликацијске програме који се налазе на удаљеном серверу, а на саме уређаје се достављају преко корисничког интерфејса претраживача који су доступни на уређајима. Предности развоја веб апликација у односу на стандардне су у томе што не постоји зависност од карактеристика оперативних система, односно једном развијена мобилна веб апликација може се покретати на различитим платформама. Једна од главних мана је немогућност приступа свим функционалностима уређаја на којима се покрећу.

Хибридне мобилне апликације представљају апликацијске програме који комбинују претходна два приступа. Код хибридних апликација углавном се развија једна апликација коју је уз мало или чак без било каквих измена могуће покретати на различитим платформама. Са друге стране за разлику од веб апликација углавном омогућавају приступ свим функционалностима уређаја [1].

1.2 Циљ рада

Циљ рада је критички сагледати и упоредити онснове карактеристике неке од радних оквира коју нуде могућност развоја апликација за оперативни систем Андроид. Биће обрађени радни оквири *Xamarin*, *Apache Cordova* и *React Native* и на сваком од њих је направљена по једна тест апликација како би се посматране карактеристике практично провериле. Свака од апликација се састоји од неколико функционалности које одражавају неке од основних ствари које се јако често јављају у развоју мобилних апликација. То су:

- коришћење камере за прављење слике;
- чување слика у меморији телефона;
- учитавање локације корисника;

- дохватање слика по кључној речи са сервиса *Flickr*; [1]

Приликом пријаве теме разматрани су и радни оквири *Sencha Touch* и *Appcelerator Titanium* који нису детаљније обрађени у раду, док је у рад увршћен радни оквир *React Native*, јер тренутно представља један од радних оквира на коме ради највише програмера када је у питању развој хибридних апликација за мобилне уређаје и који по неким изворима представља будућност развоја хибридних апликација [2][3].

Радни оквир *Sencha Touch* омогућава програмерима да уз помоћ уобичајенеих веб технологија *CSS3*, *HTML5* и *JavaScript-a* израде графичко корисничко окружење за апликације за мобилне уређаје. Последња верзија која је објављена 2015. године носила је ознаку 2.4. Убрзо након тога је овај радни оквир спојен са радним оквиром *ExtJS*, којег је развијала иста компанија. Након тога објављена је једна верзија радног оквира *ExtJS*, током 2016. године. Предности овог радног оквира су богатство елемената доступних за коришћење на корисничком окружењу, робусан пакет података који омогућује једноставно дохватање података са било ког извора и могућност развијања апликација за све најкоришћеније оперативне системе за мобилне уређаје. Највећа мана је немогућство приступа функционалностима уређаја, попут камере или контаката, те се у тим ситуацијама користи заједно са радним оквиром *Cordova* [4]. Због тог разлога, као и због тога што овај радни оквир користи исте технологије за развој као и радни оквир *Cordova* одлучено је да се не ради његова детаљнија анализа.

Радни оквир *Appcelerator Titanium* користи само *JavaScript* за развој апликација. Чак се и елементи графичко корисничког окружења програмирају кроз *JavaScript* код. Предност му је што се ти елементи исцртавају попут урођених елемената, али је мана што подржава само графичко корисничке елементе заједничке за све платформе. Поред тога, мане су му и што програмери морају учити *Titanium* АПИ, што захтева лиценцирање за употребу и што се доступне библиотеке углавном наплаћују. Поред тога, главни разлог што овај радни оквир није детаљније обрађен је и реалтивно мали број програмера који развијају на њему у односу на неке друге радне оквире [5].

[1] *Flickr* је сервис за складиштење слика и видео материјала

1.3 Оперативни системи за мобилне уређаје

1.3.1 Историјат

Данашњи живот је скоро немогуће замислити без употребе паметних телефона јер данас телефони нуде много више могућности него првобитни телефон. Преко телефона данас учимо, плаћамо, забављамо се, зарађујемо итд, за разлику од првобитних телефона који су били доста оскуднији са функционалностима.

Први мобилни телефон продат је 1983. и то је био *DynaTAC 8000x*. Власници првих телефона су могли да позову свега неколико људи у својој околини током дана због ограничења саме мреже. Такође, батерија је могла да издржи свега 30 минута разговора а за њено пуњење је било потребно и до 10 сати, а сами телефони су били тешки око 2 килограма. Ипак, чак и тај телефон је имао једноставну апликацију *Контакти* која је била доступна корисницима [6] [7].

За разлику од данашњег развоја апликација за мобилне телефоне ситуација у почетку је била потпуно другачија. Сваки произвођач је софтвер за апликације развијао у оквиру своје компаније и то углавном у тајности како би могли надмашити конкуренцију и својим корисницима понудити више могућности. То практично значи да је развој апликација био онемогућен свим програмерима који нису радили у компанијама које су развијале мобилне телефоне.

Једна од првих врста апликација која је постала популарна на телефонима су биле игрице. Једна од најпопуларнијих је свакако Нокијина (енг. *Nokia*) игра Змијица (енг. *Snake*), али су се убрзо појавиле и игрице попут Понга, Тетриса и слично.

Током деведесетих нарастао је притисак на произвођаче за новим игрицима и новим функционалностима на телефонима од стране корисника. У то време веб презентације су биле доста развијеније, са много више текста, слика и других медијалних садржаја. Такве веб презентације је било скоро немогуће приказати на мобилним телефонима јер су били ограничени резолуцијом екрана, меморијом и процесорском снагом. Решење се појваило у облику новог протокола који је назван *WAP* (енг. *Wireless Application Protocol*) који је био верзија постојећег *HTTP* протокола. *WAP* претраживачи су били направљени тако да могу да раде у хардверски ограниченим условима какви су постојали на тадашњим телефонима. WAP презентације су тако

корисницима одједном омогућиле приступ вестима, спортским резултатима, берзама и разним другим садржајима.

Прве комерцијалне *WAP* презентације су били каталози слика за позадине телефона и мелодија за звукове примања позива и порука. То је по први пут омогућило корисницима да персонализују своје телефоне.

Ипак *WAP* у многоме није испунио очекивања. Презентације су се учитавале јако дуго, а корисници су морали да плате за све податке који су скинули приликом учитавања страница. То је поприлично уништило угођај корисницима, који су свакако хтели још више могућности на телефонима. Млађи корисници у то време су користили конзоле за играње игрица, музичке плејере те је само тржиште показало пут произвођачима телефона у ком правцу треба да се крећу, јер развој традиционалних телефона није представљао сигурну будућност.

То је довело до појаве различитих власничких платформи (енг. *proprietary platforms*) за развој мобилних апликација. Један од првих је био *Palm OS* чија је прва верзија објављена 1996. године који је у себи имао апликације за контакте, листу задатака (енг. *ToDo list*) и калкулатор. Карактеристично је што су апликације инсталиране и извршаване директно у РАМ меморији. Годину дана касније се појавио *Symbian* за кога су платформе развијали произвођачи телефона *Nokia*, *Sony Ericcson*, *Motorola* и *Samsung*. Две године касније појавила се и прва верзија *BlackBerry OS* која је објављена за пејџер *BlackBerry 850*. Компанија *Microsoft* је свој први оперативни систем за мобилне телефоне објавила 2000. године под именом *PocketPC* који је 2003. године преименован у *Windows Mobile*.

Октобра 2003. године у Пало Алту, Калифорнија, основана је компанија Андроид (енг. *Android Inc*). Према опису једног од оснивача Андроид пројекат је представљао огроман потенцијал у развоју паметнијих мобилних уређаја који би били свеснији географске локације и преференција власника. Првобитна намера је била да се развија напреднији систем за дигиталне фотоапарате, али се од тога убрзо одустало јер су утврдили да то тржиште није довољно велико за њихове циљеве. [8] Током 2004. године мењају се приоритети те Андроид бива представљен као оперативни систем за мобилне телефоне који би био конкурент системима *Symbian* и *Windows Mobile OS* који су у то време били најраспрострањенији. Прекретница у историји овог система се десила јула 2005. године када је компанија *Google* купила компанију Андроид и представила га произвођачима телефона и телекомуникационим компанијама као флексибилан и надоградив систем.

Новембар 2007. године је битан месец у историји Андроид система јер је тада основана организација *Open Handset Alliance*[2]. Истог месеца је та организација објавила циљ да развије прву отворену и свеобухватну платформу за мобилне уређаје. То је резултирало већ следеће године када је објављен први комерцијални телефон који је покретао Андроид ОС, а то је био *HTC Dream* познатији и као *T-Mobile G1*.

Компанија *Apple Inc.* је 2007. године објавила свој *iPhone OS* намењен искључиво уређајима које произведе та компанија. Врло брзо након објаве система, већ 2008. године је објављен и сет алата за развој апликација (енг. *Software Development Kit*) као и продавница са иницијално 500 доступних апликација за кориснике што је врло брзо привукло програмере развоју апликација за тај систем. До данас је објављено десет верзија система који од своје верзије број 4, која је објављена 2010. године, носи име *iOS*.

1.3.2 Данас

Временом су се многи од система угасили под притиском конкуренције, тако да практично постоје два најраспрострањенија оперативна система - Андроид и *iOS*, који заједно заузимају преко 95% тржишта по већини истраживања [9]. Сви остали заједно имају врло мали утицај и базу корисника.

Подаци за последње тромесечје 2016. године потврђују доминацију ова два система. Наиме, од свих нових телефона који су продати у овом периоду чак 81,7% је покретао систем Андроид, а 17,9% систем *iOS-y*. Сви остали су имали удео мањи од 0,5% укључујући и *Windows Phone* са својих 0,3% тржишта [10].

Наравно постоји разлика у процентима у зависности од земље до земље. Табела 1.1 1.1 показује удео мобилних оперативних система по земљама и регијама.

Доминација Андроид оперативног система на многим тржиштима се може објаснити због неколико фактора:

- систем је бесплатан од самог појављивања те је због тога привукао пажњу великог броја произвођача телефона;

[2]Конзорцијум копманија чији је циљ развој отворених стандарда за мобилне уређаје. Неке од чланица конзорцијума су *Google*, *T-Mobile*, *LG*, *Samsung* итд.

Регија	Андроид	*iOS*	*Windows*	*BlackBerry*
Глобално	71,87%	19,88%	0,96%	0,32%
Африка	70,86%	5,38%	1,74%	1,75%
Азија	76,91%	12,92%	0,68%	-
Европа	68,55%	28,29%	1,96%	0,24%
Северна Америка	49,95%	48,48%	0,54%	0,32%
Јужна Америка	85,69%	8,91%	2,75%	0,26%
Србија	84,24%	10,36%	3,48%	0,19%
САД	45,08%	53,87%	0,37%	0,11%
УК	46,33%	49,63%	1,87%	0,97%
Индија	77,8%	2,72%	1,01%	-
Русија	67,07%	29%	2,1%	-
Кина	55,54%	43,82%	0,12%	0,01%
Француска	62,54%	34,58%	1,96%	0,16%
Аустралија	39,58%	57,25%	0,64%	0,04%

Табела 1.1: Удео оперативних система за мобилне уређаје по регијама за април 2017. Извор *statcounter* [11]

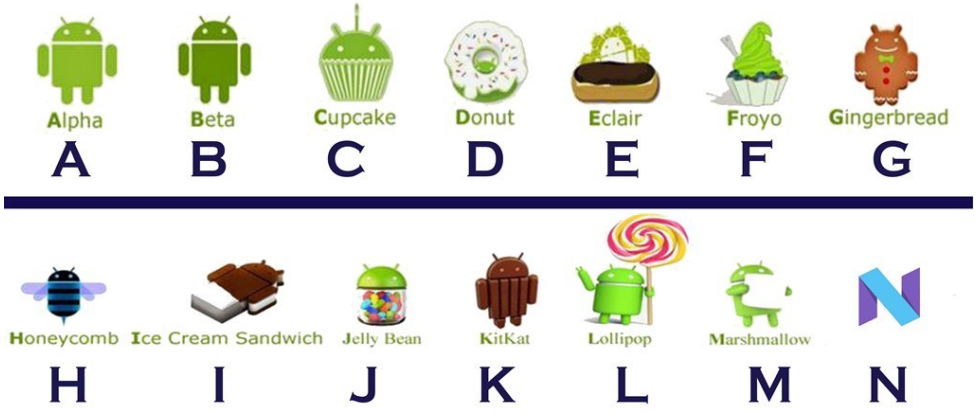
- за разлику од већине осталих система, Андроид је под лиценцом отвореног кода (енг. *Open Source*) што је привукло велики број програмера који су временом допринели развоју система;
- омогућено је произвођачима телефона да прилагођавају систем по својој жељи како би својим корисницима омогућили угоднije коришћење телефона;
- велики број доступних апликација на продавници од којих су већина бесплатни за крајње кориснике;
- велики избор уређаја које покреће систем Андроид, чије цене су доступне већини становништва на глобалном нивоу.

Глава 2

Оперативни систем Андроид

Андроид је платформа заснована на принципу отвореног кода. То значи да ни програмери ни произвођачи телефона не плаћају ништа да би развијали на Андроид платформи. Андроид је лиценциран под *GNU* [1] лиценцом која подразумева да свако ко доноси и приказује нова побољшања система мора свој рад објавити под истом лиценцом.

У претходном поглављу је описана кратка историја оперативног система Андроид, а слика 2.1 приказује историју главних верзија оперативног система Андроид. Свака од приказаних верзија је донела одређене новине и побољшања а од верзије 1.5 свака верзија је добијала име по некој од посластица, јер како кажу у самој компанији њихов оперативни систем чини живот корисника лакшим и слађим баш као и посластице.



Слика 2.1: Верзије оперативног система Андроид

[1] GNU, (енг. *General Public License*) је лиценца за слободни софтвер.

2.1 Архитектура

Андроид стек се обично грубо дели у 5 слојева као што је приказано на Слици 2:

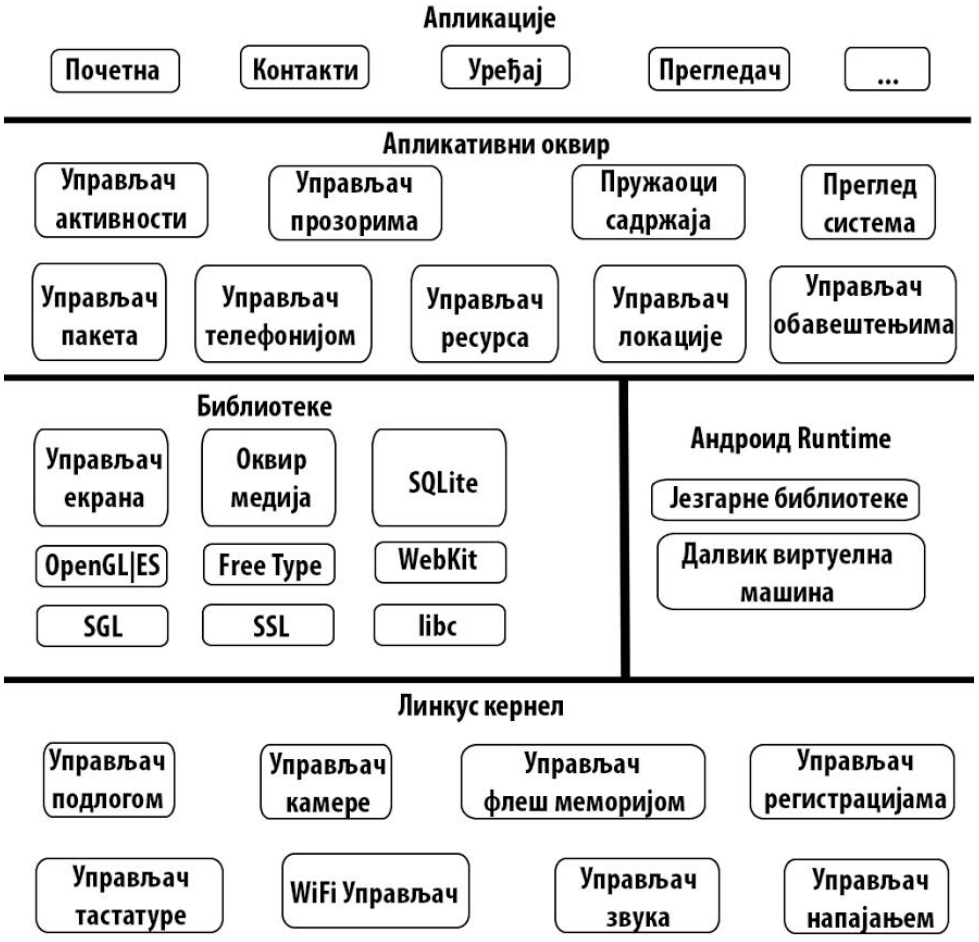
- Линукс језгро
- Библиотеке
- *Android Runtime*
- Апликацијски радни слој
- Апликације

2.1.1 Језгро Линукса

Језгро Линукса је задужено за комуникацију са хардвером и садржи све неопходне драјвере за камеру, тастатуру, екран итд. Такође језгро Линукса води рачуна о повезивању на мрежу, управљању процесима и меморијом. Само језгро је написано на програмском језику *C*. До верзије 4.0 Андроид система коришћено је језгро базирано на језгру Линукса 2.6.x а од тада се користи језгро базирано на верзији 3.x.

2.1.2 Библиотеке

Изнад језгра се налазе системске библиотеке које су углавном писане у програмским језицима *C++* и Јава. Неке од библиотека које су смештене у овом слоју су покретач за веб претраживаче *WebKit*, библиотека *libc*, база података *SQLite*, библиотеке задужене за пуштање и снимање аудио и видео датотека, библиотеке *SSL* задужене за енкрипцију, библиотеке за рад са графиком итд. Библиотека *libc* је развијена специјално да би се оптимизовало коришћење меморије, што је врло битно на уређајима који користе систем Андроид јер су то углавном уређаји са врло мало главне меморије и слабијим перформансама.



Слика 2.2: Графички приказ архитектуре Андроид система

2.1.3 Android Runtime

Android Runtime је радно окружење које се налази у склопу слоја библиотека и садржи основне библиотеке и виртуелну машину *Dalvik*.

Основне библиотеке омогућавају програмерима коришћење стадардних функционалности програмског језика Јава.

Виртуелна машина *Dalvik* је развијена од стране компаније *Google* као замена за Јава 2 микро издање (енг. *J2ME*) која представља стандардни механизам за покретање апликација писаних у програмском језику Јава на мобилним уређајима. Основна разлика између ове две виртуелне машине је у томе што је Јавина виртуелна машина базирана на принципу стека, а *Dalvik* је регистарски базирана виртуелна машина. Превођење кода из Јавиних *.class* датотека у нове *.dex (Dalvik Executable)* датотеке омогућава бољу прилагођеност за рад на слабијим процесорима и боље искоришћење

меморије на уређајима. Такође омогућено је вишеструко инстанцирање виртуелне машине, што значи да се свака Андроид апликација покреће као засебан процес и са својом инстанцом виртуелне машине *Dalvik*.

2.1.4 Апликацијски радни слој

Радни оквири на овом слоју представљају апстрактни слој ка стандардним библиотекама и функционалностима виртуелне машине *Dalvik*. Ови сервиси су доступни програмерима за коришћење у облику Јава класа. Неки од оквира који се налазе на овом слоју су:

- Управљач акцијама - задужен за управљање свим аспектима животног циклуса апликација;
- Испоручиоци садржаја - задужени за омогућавање дељења података са другим апликацијама;
- Управљач ресурсима - омогућује приступ угњежденим ресурсима као што су ниске, боје и изгледи корисничког сучеља;
- Управљач обавештењима - омогућује апликацијама приказивање упозорења и обавештења.

На самом врху архитектуре се налазе саме апликације које се развијају за оперативни систем Андроид од стране програмера.

2.2 Компоненте апликација

Андроид апликације су јединствене јединице које могу да се инсталирају, покрећу и користе независно. Апликације се састоје од конфигурационих фајлова, Јава изворног кода и датотека ресурса. Након компајлирања кода са програмског језика Јава добијени код се архивира у Андроид пакет заједно са свим подацима и свим осталим неопходним датотекама. Добијени пакет се дистрибуира и инсталира на уређајима.

Свака апликација се извршава у сопственом процесу који се покреће када треба да се изврши било који део апликације, а гаси се када више није неопходан. Свака

апликација се извршава независно од осталих јер се свака извршава у сопственој инстанци виртуелне машине. Могуће је и променити поставке тако да подаци из једне аплиакције буду видљиви другој апликацији.

Андроид апликације се састоје од основних компоненти које се могу инстанцирати и употребљавати по потреби. То значи да свака од компоненти може представљати улазну тачку преко које систем приступа апликацији. Постоје 4 врсте компноненти и то су:

- Активности (енг. *Activities*);
- Сервиси (енг. *Services*);
- Испоручиоци садржаја (енг. *Content providers*);
- *Broadcast receivers*.

2.2.1 Активности

Активност представља појединачни екран са графичким корисничким окружењем. Активности користе погледе (енг. *views*) и фрагменте (енг. *fragments*) за прављење графичког корисничког окружења и за интеракцију са корисницима. Свака апликација може имати више активности и обично су повезане да би правиле логичку целину, иако је свака активност независна од осталих. Битно својство је да једна апликација може да покрене активност из неке друге апликације иако та друга апликација није покренута. На пример из апликације за плаћање паркинга је могуће отворити активност за слање СМС порука из одговарајуће апликације. Свака активност се имплементира као подкласа класе *Activity*.

```
public class MainActivity extends Activity {
}
```

Врло је битно разумети и концепт животног циклуса активности. Свака активност има четири основна стања:

- Трајање (енг. *Running*);
- Мировање (енг. *Paused*);

- Заустављено извршавање (енг. *Stopped*);
- Уништење (енг. *Destroyed*);

Фазама животног циклуса управљају методе *onCreate()*, *onStart()*, *onResume()*, *onPause()*, *onStop()*, и *onDestroy()*. Слика 2.3 приказује дијаграм животног циклуса активности. [12]

Слика 2.3: Животни циклус активности

Метода *onCreate()* се позива када систем први пут прави активност. Пракса је да се у тој методи извршава код који представља основну логику која би требала да се изврши само једном за цео животни век активности. То је на пример поставка података у листе, иницијализација нити које се извршавају у позадини, инстанцирање класних променљивих и слично.

Метода *onStart()* омогућује исцртавање графичког корисничког окружења на екран. У овој методи апликација припрема активност да буде приказана у првом плану. Примери задатака који се извршавају у овом статусу су иницијализација кода одговорног за исцртавање и одржавање видљивих елемената, или регистровање *BroadcastReceiver* догађаја који прате промене које могу утицати на промену графичког корисничког окружења.

Метода *onResume()* се позива сваки пут (укључујући и први пут приликом прављења активности) када активно доспе у први план. У овом стању апликација је у интеракцији са корисником и остаје у том стању све док се не деси нешто што склони фокус са активности. Такви догађају су на пример примање позива, прелазак на нову активност и слично. Када се деси неки такав догађај активност прелази у стање паузе и позива се метода *onPause()*. Погодне ствари за извршавање у овој методи су започињање анимација и иницијализација компоненти које активности користи само када има фокус корисника.

Метода *onPause()* се позива када се појави прва индиција да корисник напушта активност. У овој методи би требало да се прекине извршавање свих операција чије извршавање није неопходно док је активност у стању мировања. Такве операције су на пример извршавање анимација или пуштање музике. Такође у овом стању се могу отпустити системски ресурси као што су *broadcast receivers*, *GPS* систем и слични који могу утицати на трајање батерије уређаја.

Матода *onStop()* се позива када активност више није видљива крајњем кориснику. Пример када се ова метода позива је када се позове нова активност која долази у први план. И у овом стању се могу позвати методе за отпуштање *broadcast receivers* који су регистровани у методи *onStart()*. У овој методи се могу извршити и операције које захтевају више процесорске снаге као што је чување података у трајној меморији. Док је у стању заустављеног извршавања активност се и даље чува у меморији. Из овог стања активност је могуће поново поренути или је у потпуности уништити уколико је извршавање завршено.

Метода *onDestroy()* се позива пре него што се активност уништи. До позива ове методе долази или због тога што се позове метода *finish()* или због тога што систем привремено уништава процесе како би ослободио меморијски простор. Ова два сценарија је могуће разликовати преко *isFinishing()* методе. Изузетак је када систем позове ову методу када се промени оријентација телефона. Тада се одмах након ове

методе позива *onCreate()* метода како би се поново креирали сви процеси и компоненте у новој оријентацији.

2.2.2 Сервиси

Сервиси извршавају у позадини задатке за који није потребно графичко корисничко окружење. Они могу да комуницирају са осталим компонентама и да шаљу обавештења (енг. *notifications*) корисницима. Често се употребљавају за задатке који могу да захтевају дуготрајно извршавање као што је на пример дохватање података са мреже без блокирања корисникове интеракције са активношћу.

Сваки сервис се имплементира као подкласа класе *Service* и углавном бивају покренути од стране других компоненти система, а најчешће од стране активности.

```
public class MyService extends Service {
}
```

2.2.3 Испоручиоци садржаја

Испоручиоци дефинишу структуиран приступ ка апликацијским подацима. Испоручиоци се углавном користе за приступ подацима једне апликације, али могу бити искоришћени и за дељење података са другим апликацијама.

Андроид садржи базу података *SQLite* која се често користи у спони са испоручиоцима садржаја. Обично се подаци смештају у базу података, а преко испоручиоца се приступа тим подацима. Поред база података, подаци преко којих се приступа уз помоћ испоручиоца могу бити складиштени и у систему датотека, на мрежи или на било којој другој трајној локацији којој и сама апликација може приступити.

Испоручиоци садржаја се имплементирају као подкласе класе *ContentProvider* и морају имплементирати стандардни скуп метода како би апликацијама било омогућено да извршавају трансакције.

```
public class MyContentProvider extends  ContentProvider {
   public void onCreate(){}
}
```

2.2.4 Пријемници

Пријемници могу бити регистровани да ослушкују како системске поруке и намере (енг. *intents*) тако и поруке од стране других апликација. Када се деси одређени системски догађај, пријемник се аутоматски обавештава о њему. Помоћу ове компоненте могуће је иницирати емитовање података ка другим апликацијама. Ове компоненте такође немају графичко корисничко окружење али могу покренути активност као одговор на податке које су примили или могу користити неки од сервиса у позадини како би дали обавештење корисницима о пристиглим подацима. Обично се за то користе треперење светла, вибрација или репродукција одређеног звука.

Примери системских порука су обавештење да је батерија слаба, да је слика направљена у случају да се користи апликација Камера, да је промењена временска зона итд.

Пријемници су имплементирани као подкласе класе *BroadcastReceiver*, а свака порука се емитује као објекат типа *Intent*.

```
public class MyReceiver  extends  BroadcastReceiver {
   public void onReceive(Context context, Intent intent){}
}
```

2.2.5 Намере

Намере (енг. *intents*) представљају механизам за прослеђивање порука унутар апликације али и између различитих апликација. Најчешће се користе за покретање активности или сервиса, као и за објављивање системских догађаја као што су надолазећи позив или пристизање СМС поруке.

```
Intent myIntent = new Intent(CurrentActivity.this, NextActivity.class);
myIntent.putExtra("key", value);
CurrentActivity.this.startActivity(myIntent);
```

Пример кода 2.1: Покретање нове активности коришћењем намере

Глава 3

Xamarin

Xamarin је платформа, или радни оквир, који омогућава развој стандардних апликација за оперативне систем Андроид, *iOS* и *Windows*. За развој се у потпуности користи програмски језик *C#*, што омогућује програмерима који познају овај програмски језик, развијање апликација за мобилне уређаје. Заснован је на платформи отвореног кода за радни оквир *.NET*, чија је сврха била извршавање програма писаних на програмском језику *C#*, на оперативним системима који нису *Windows*, као што су системи засновани на *Unix-y*. Састоји се од сопственог *C#* преводиоца, окружења за извршаваање као и од основних *.NET* библиотека које омогућују све основне функционалности као што су спајање на мрежу, разне структуре података, приступ датотекама, бази података, нитима и слично. Битна карактеристика платформе је доступност целокупног *API-ja* за *iOS* и Андроид оперативне системе.

3.0.1 Историјат

Компанија *Xamarin* је основана 2011. године. Претече ове платформе сежу доста раније, до 2003. године када почиње рад на платформама *MonoTouch* и *Mono for Android* у оквиру компаније *Novell*. Након што је *Novell* продат, цео тим који је радио на пројекту *Mono* прелази у новоосновану компанију *Xamarin*. За разлику од својих претеча сам *Xamarin* је био комерцијални производ од самог почетка. Године 2016. компанију је купио *Microsoft*. Последица ове куповине јесте да је *Xamarin SDK* постао доступан у оквиру алата *Visual Studio* али је истовремено и објављен под лиценцом отвореног кода *MIT*. Такође укинута је наплата лиценце што је омогућило

даљи и бржи развој платформе. Иако се сматра младом платформом компанија истиче да тренутно има преко 15000 клијената и преко 1,4 милиона програмера који развијају апликације на овој платформи [13].

3.0.2 Структура

Платформа има два главна производа:

- Xamarin.iOS - библиотека класа које омогућавају приступ оперативном систему *iOS*;
- Xamarin.Android - библиотека класа које омогућавају приступ оперативном систему Андроид.

Разлика између њих је у начину превођења кода (енг. *compile*). За *iOS* изворни код се директно преводи у стандардни *ARM* код пре почетка извршавања истог, док се за Андроид код прво преводи у међукод који је читљив виртуелној машини *Mono* која се такође додаје у апликациони пакет. Потом се међукод преводи у природни код и то у време извршавања (енг. *Just-in-Time* превођење). Током извршавања Андроид апликација обе виртуелне машине (*Dalvik* и *Mono*) раде у исто време једна поред друге и деле податке преко посебних канала комуникације.

Слика 3.1: Поједностављена архитектура *Xamarin* система

Процена компаније је да је у просеку могуће делити око 75% изворног кода приликом развоја за различите платформе. У део кода који могу користити различите

платформе спадају пословна логика, слојеви за дохватање и складиштење података, сервисни слој, модели података и слично. Остатак кода је углавном везан за графичко окружење које се разликује од платформе до платформе.

3.0.3 Алат Xamarin.Forms

Једна од битних одлика ове платформе је постојање алата *Xamarin.Forms* који конвертује елементе графичког окружења у елементе специфичне за сваку платформу. То превођење се одвија током извршавање апликације те углавном апликације које су развијене овим путем имају нешто слабије перформансе баш због тог додатног слоја апстракције графичког окружења. Ипак велика је предност приликом развијања апликација јер се тај процес значајно убрзава.

Код се може писати као *C#* код или *XAML* код. *XAML (Extensible Application Markup Language)* је програмски језик заснован на језику *XML* развијен од стране *Microsoft-a*. *XAML* је језик који стоји иза видљиве репрезентације апликације која се развија. Једна страница у *Xamarin.Forms* представља један екран у апликацији. Странице подржавају подршку за додире, распоред елемената (енг. *layout*), дугмиће, ознаке (енг. *labels*), листе и остале уобичајене елементе који се користе.

Коришћење *Xamarin.iOS* и *Xamarin.Android* се препоручује код апликација где интеракције захтевају природно понашање, које користе много специфичних *API* метода везаних за одређену платформу и код апликација где је изглед корисничког окружења много битнији од дељења кода.

3.1 Развој апликације

3.1.1 Подешавање окружења

За развој Андроид апликација потребно је инсталирати програмско окружење *Microsoft Visual Studio*. У раду је коришћена верзија за заједницу програмера (енг. *community edition*) под ознаком 7.0.1. Током инсталације се бирају компоненте које ће бити инсталиране. Могуће је инсталирати појединачне компоненте везане за Андроид или

iOS или за веб апликације, а уколико потребе програмера то захтевају, могуће је инсталирати све компоненте, јер ово програмско окружење подржава развој свих ових технологија.

3.1.2 C#

Програмски језик *C#* је језик који подржава више програмских парадигми као што су објектно-оријентисана, функционална и компонентна. Заснован је на синтакси језика *C* коју у великој мери користи језик *C++*, са чијом синтаксом се такође доста поклапа. Развијен је од стране компаније *Microsoft* а до сада је објављено седам главних верзија овог језика од којих последња у марту 2017. године. Поред тога што је могуће развијати апликације за оперативне системе Андроид, *iOS* и *Windows Phone*, овај језик се користи и за развој конзолних, *Windows* и веб апликација.

3.1.3 Рад са ресурсима

Рад са нискама у овом радном оквиру је поприлично једноставан и сличан раду на стандардном окружењу. Ниске се чувају у датотекама *Strings.xml* у директоријуму *Resources/values*. У примеру кода 3.1.3 је приказан пример датотеке са нискама из апликације. Уколико је потребно додати преводе за додатне језике потребно је направити посебне директоријуме са именом *values-kodJezika*. За додавање ћириличне верзије апликације било би потребно да се направи директоријум *values-sr_RS* и у њега смести одговарајућа датотека са ћириличним нискама [14]. У самим датотекама свакој ниски која се користи у апликацији се додељује кључ по коме ће се ниска дохватати из кода. Ниске се из кода дохватају помоћу методе:

```
GetString(Resource.String.camera_and_memory)
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">MATFMaster</string>
  <string name="camera_and_memory">Kamera i Memorija</string>
  ...
</resources>
```

Пример кода 3.1: Део датотеке *Strings.xml* у којој су смештене ниске

Фотографије које се користе у апликацији се смештају у директоријум *Resources/drawable*. Да би се одржао идентичан изглед фотографија на различитим величинама екрана потребно је фотографију под истим именом а у различитим резолуцијама поставити у одговарајуће директоријуме у оквиру директоријума *Resources*. То су директоријуми *mipmap-ldpi*, *mipmap-mdpi*, *mipmap-hdpi*, *mipmap-xhdpi*, *mipmap-xxhdpi*, *mipmap-xxxhdpi* [15]. Коришћене фотографије из кода и постављање исте на *ImageView* се позива на следећи начин:

```
private ImageView _imageView;
...
_imageView.SetImageResource(Resource.Drawable.image_preview);
```

3.1.4 Рад са елементима графичко корисничког окружења

Графичко корисничко окружење могуће је направити на два начина:

- писањем *xml* кода;
- помоћу уграђеног одељка за превлачење објеката.

Xamarin.Android подржава рад са свим основним градивним блоковима Андроид апликација као што су линеарни распоред *Linear Layout*, распоред заснован релативним односима *Relative Layout*, табеларни распоред *Table Layout*, мрежни распоред *Grid View*, странични распоред *Tab Layout* и листа погледа *List View* [16].

3.1.5 Рад са камером и меморијом

У апликацији је имплементирана функционалност за прављење фотографија уз помоћ уграђене апликације под именом Камера. Фотографије се чувају у меморији у директоријуму направљеном из апликације под именом **MATFApp**. За ове две акције потребно је додати две дозволе (енг. *permissions*) *CAMERA* и *WRITE_EXTERNAL_STORAGE* у датотеку *AndroidManifest.xml*.

```
private void CreateDirectoryForImages()
{
  App._dir = new File(Environment.GetExternalStoragePublicDirectory(
        Environment.DirectoryPictures), "MATFApp");
  if (!App._dir.Exists())
  {
      App._dir.Mkdirs();
  }
}
```

Пример кода 3.2: Метода која прави директоријум у меморији уколико већ не постоји

На притисак дугмета за покретање камере покреће се *Intent* који од система тражи да пронађе активност која може да прави фотографије што је обично већ уграђена апликација Камера. Том *Intent*-у се прослеђује и локација на коју треба да сачува фотографију. У истом кораку потребно је одабрати и име фотографије како би путања до датотеке и сама локација били потпуни. За то је искоришћен *Guid* метода случајног одабира који се наставља на основу имена: **фотографија_**.

```
private void TakeAImage(object sender, EventArgs eventArgs)
{
    Intent intent = new Intent(MediaStore.ActionImageCapture);
    _imageName = String.Format("fotografija_{0}.jpg", Guid.NewGuid());
    App._file = new File(App._dir, _imageName);
    intent.PutExtra(MediaStore.ExtraOutput, Uri.FromFile(App._file));
    StartActivityForResult(intent, 0);
}
```

Пример кода 3.3: Метода која покреће *Intent* и конструише путању у меморији до датотеке

Након што апликација Камера заврши са прављењем фотографије позива се метода *OnActivityResult*. У тој методи се фотографија дохвата из меморије и поставља се да буде доступна у галерији фотографија на уређају за шта је задужен *Intent.ActionMediaScannerScanFile*. Након тога се фотографији смањује величина и тако умањена се поставља на већ припремељни *imageView*. Разлог зашто се фотографија умањује је тај што би постављање непромењене фотографије из меморије

у већини случајева довело до пуцања апликације због превелике величине оригинала. Такође у овој методи се поставља име датотеке на ознаку за име на графичко корисничком окружењу.

```
protected override void OnActivityResult(int requestCode, Result
    resultCode, Intent data)
{
  base.OnActivityResult(requestCode, resultCode, data);

  Intent mediaScanIntent= new Intent(Intent.ActionMediaScannerScanFile);
  Uri contentUri = Uri.FromFile(App._file);
  mediaScanIntent.SetData(contentUri);
  SendBroadcast(mediaScanIntent);

  int height = Resources.DisplayMetrics.HeightPixels;
  int width = _imageView.Height;
    App.bitmap = App._file.Path.LoadAndResizeBitmap(width, height);
  if (App.bitmap != null)
  {
      _imageView.SetImageBitmap(App.bitmap);
    App.bitmap = null;
  }
    _imageNametextView.Text = _imageName;
}
```

3.1.6 Дохватање локације

Оперативни систем Андроид омогућује приступ различитим технологијама за дохватање локације као што што су локација базне станице, *Wi-Fi* и *GPS*. Детаљи сваке технологије су апстраховани кроз провајдере локације што омогућују апликацијама да дохвате локацију на исти начин без обзира који извор се користи. Да би се дохватила локација телефона апликацији је потребан приступ хардверским сензорима да би се примили подаци од *GPS-a*, *Wi-Fi*-а и мобилне мреже. Приступ се регулише кроз одговарајуће дозволе које се дефинишу у датотеци *Android Manifest*. Постоје две дозволе које је могуће користити:

- *ACCESS_FINE_LOCATION* - даје дозволу апликацији да приступи *GPS* технологији;

- *ACCESS_COARSE_LOCATION* - даје дозволу апликацији да приступи подацима мреже и *Wi-Fi* локацији.

Подаци о локацији се купе преко хардверских сензора и смештају у системски сервис коме се из апликација приступа преко класе *LocationManager* и имплементације интерфејса *ILocationListener*. Инстанца класе *LocationManager* се добија позивањем методе *GetSystemService*:

```
LocationManager _locationManager;
...
_locationManager = GetSystemService(Context.LocationService) as
  LocationManager;
```

Позивом методе *RequestionLocationUpdates* на класи *LocationManager* апликација се претплаћује на добијање података о локацији. Ова метода омогућује да се одабере извор података, као и време и померај како би се управљало ажурирањем података.

```
_locationManager.RequestLocationUpdates(LocationManager.NetworkProvider,
    2000, 1, this);
```

Пример позива методе приказан изнад користи мрежне податке као извор, и захтева се ажурирање података о локацији на сваких 2000 милисекунди и само онда када се локација променила за више од једног метра.

Интерфејс *ILocationListener* обезбеђује методе које прате промене на системском сервису за локацију.

```
public class LocationActivity : Activity, ILocationListener
{
  ...

  public void OnProviderEnabled (string provider)
  {
    ...
  }
  public void OnProviderDisabled (string provider)
  {
    ...
  }
  public void OnStatusChanged (string provider, Availability status,
   Bundle extras)
```

```
  {
    ...
  }
  public void OnLocationChanged (Android.Locations.Location location)
  {
    ...
  }
}
```

Пример кода 3.4: Имплементација интерфејса *ILocationListener*

Интерфејс омогућује да се апликација претплати на четири системска догађаја:

- *OnProviderEnabled* и *OnProviderDisabled* - методе које обавештавају апликацију када је корисник омогућио или онемогућио извор података;
- *OnStatusChanged* - метода која обавештава апликацију када се променио статус извора података (нпр када корисник уђе у затворен простор могуће је да се изгуби *GPS* сигнал);
- *OnLocationChanged* - метода која обавештава апликацију да се локација корисника променила по задатим критеријумима приликом подешавања захтева за дохватање локације.

3.1.7 Рад са подацима и мрежом

Повезивање апликације и веб сервиса је уобичајена ствар у развоју апликација за мобилне уређаје. Пример кода 4.5 приказује моделе у које се мапира добијени одговор са сервера. За десеријализацију серверског одговора је коришћена библиотека *Newtonsoft* која је под лиценцом отвореног кода.

Пример кода 4.6 приказује методу преко које се извршава *GET* позив за дохватање слика. Уколико је статус одговора са сервера успешан, садржај одговора се десеријализује у одговарајући модел података преко *JsonConvert.DeserializeObject* методе. Класа *HttpClient* је задужена за слање захтева преко *HTTP* протокола и за дохватање *HTTP* одговора. Карактеристика је да се сваки захтев шаље као асинхрона операција.

```
using System;
using System.Collections.Generic;
namespace MATFMaster
{
    public class RootObject
    {
        public ImageRootObject photos { get; set; }
    }

    public class ImageRootObject
    {
        public int page { get; set; }
        public int pages { get; set; }
        public int perpage { get; set; }
        public string total { get; set; }
        public List<Image> photo { get; set; }
    }

    public class Image
    {
        public string id { get; set; }
        public string owner { get; set; }
        public string secret { get; set; }
        public string server { get; set; }
        public int farm { get; set; }
        public string title { get; set; }
        public int ispublic { get; set; }
        public int isfriend { get; set; }
        public string url_o { get; set; }
        public string height_o { get; set; }
        public string width_o { get; set; }
    }
}
```

Пример кода 3.5: Модели за дохватање слика са *Flickr-a*

```
using System.Net.Http;
using System.Threading.Tasks;
using Newtonsoft.Json;
...
public async Task<RootObject> RefreshDataAsync(String tag)
    {
        Photos = new RootObject();
```

```
        HttpClient client = new HttpClient();
      var uri = new Uri(string.Format(Constants.RestUrl, tag));

      try
      {
        var response = await client.GetAsync(uri);
        if (response.IsSuccessStatusCode)
        {
          var content = await response.Content.ReadAsStringAsync();
          Photos = JsonConvert.DeserializeObject<RootObject>(content);
        }
      }
      catch (Exception ex)
      {
        Debug.WriteLine(@" ERROR {0}", ex.Message);
      }

          return Photos;
    }
```

Пример кода 3.6: Метода која извршава позив за дохватање слика са *Flickr-a*

3.2 Закључак

Предности развоја на овом радном оквиру су бројне. Пре свега, као што је наведено могуће је делити 75% кода приликом развоја на различитим платформама. Такође, по различитим тестовима перформансе апликација написаних на овом радном оквиру су приближне перформансама стандардних апликација. Такође поред поменутог алата *Xamarin.Forms*, у понуди су и алати *Xamarin Test Cloud* и *Xamarin Insights*. Први може значајно смањити време у ком апликација може да стигне до већег броја корисника тако што омогућује тестирање апликације на преко 2000 уређаја доступних преко овог алата. Други омогућује програмерима приступ информацијама о престанку рада апликације (енг. *app crash*), понашању корисника у оквиру апликације као и уређајима на којим се апликација користи.

Једна од највећих негативних страна овог радног оквира је цена лиценце коју морају купити пословни корисници, а која износи 999 америлких долара. Такође, на

подршку за последњу верзију платформи за које се развија апликација, се може чекати неко време. Ту је и недостатак библиотека издатих под лиценцом отвореног кода, каквих има приличан број када је у питању стандардни развој апликација. Иста ситуација је и са бројем програмера који активно развијају на овој платформи. Апликације написане на овом радном оквиру су обично знатно веће од стандардних што се такође може посматрати као мана.

Глава 4

Apache Cordova

Apache Cordova је радни оквир за израду апликација за оперативне системе на мобилним уређајима. За развој се користе уобичајене веб технологије *CSS3*, *HTML5* и *JavaScript*. Резултат развоја на овом радном оквиру су хибридне апликације које исцртавају елементе графичко корисничког окружења преко веб погледа (енг. *Web view*) уместо стандардних елемената. Добијене апликације се не могу сврстати у чисто веб апликације јер имају приступ функционалностима уређаја путем основних *API*-ја. Коришћење овог радног оквира се препоручује програмерима за мобилне уређаје који желе да прошире апликацију на више од једне платформе, без посебне имплементације за сваку платформу, као и веб програмерима који желе да објаве апликацију на различитим продавницама апликација за оперативне системе за мобилне уређаје.

4.0.1 Историјат

Радни оквир је првобитно развијан од 2009. године од стране компаније *Nitobi* и то под именом *PhoneGap*. Компанија *Adobe Systems* је купила 2011. поменуту компанију а потом неколико месеци касније објавила верзију *PhoneGap* радног оквира под лиценцом отвореног кода, која носи име *Apache Cordova*. До сада је објављено седам главних верзија радног оквира а последње ажурирање је било у мају 2017. године [17].

PhoneGap и *Apache Cordova* су у суштини изграђени на једној основи а током година су њихови развоји кренули мало другачијим путем. У суштини ова два радна оквира

служе истој сврси и скоро у потпуности су исти. Мала разлика је настала током времена јер се *PhoneGap* састоји и од додатних алата који су у складу са другим *Adobe* сервисима, а који нису одговарајући за *Apache* пројекат. Такав алат је на пример *PhoneGap Build* који омогућава корисницима да подигну свој код на *Adobe* сервере који потом праве стандардне апликације [18].

Поред поменуте дистрибуције, постоји још много дистрибуција изграђених над оквиром *Apache Cordova*. Неки од најпознатијих су *Ionic*, *Monaca* и *Intel XDK*.

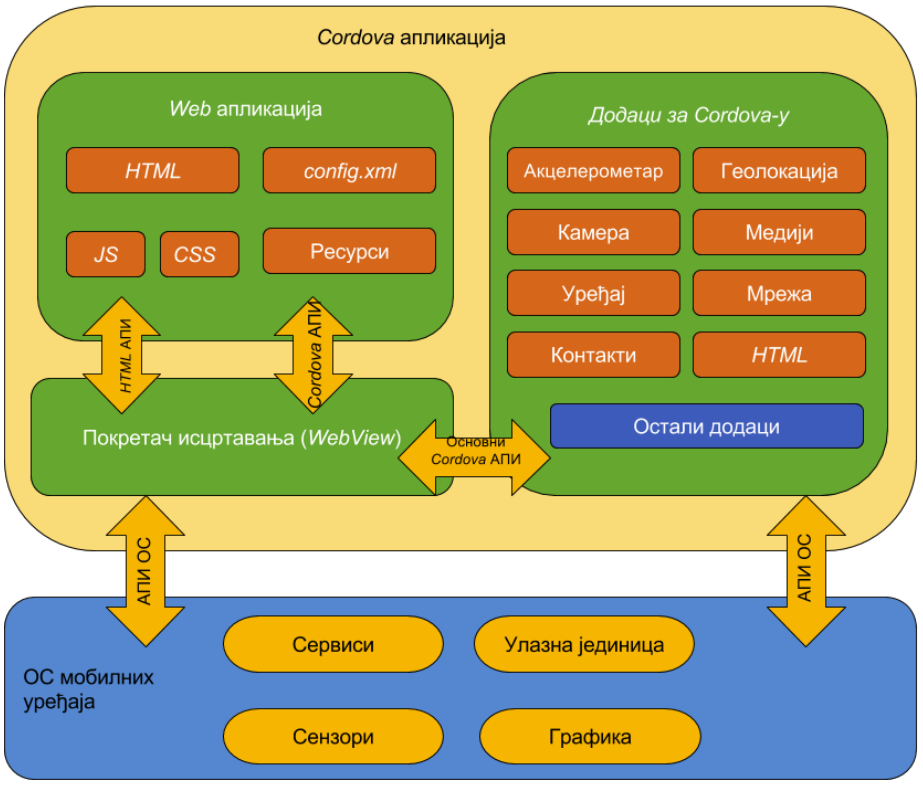
4.0.2 Архитектура

Архитектура радног оквира се састоји од неколико компоненти. Слика 4.1 илуструје архитектуру радног оквира.

- **Веб поглед** испоручује апликацији целокупно графичко корисничко окружење.
- **Веб апликација** је део где се смешта програмерски код. Сама апликација се имплементира као веб страница, обично у датотеци *index.html* из које се референцирају *CSS* и *JavaScript* код, слике, медијалне датотеке и остали ресурси неопходни да би се апликација несметано извршавала. У оквиру ове компоненте се налази и датотека *config.xml* која представља глобалну конфигурациону датотеку. У њој се дефинишу многи аспекти понашања апликације као и име апликације, аутора, опис и слично.
- **Додаци (енг. *Plugins*)** су интегрални део система *Cordova*. Они испоручују интерфејс за *Cordova*-у и стандардне компоненте како би могли међусобно да комуницирају. То омогућава програмерима да позивају стандардни код из *JavaScript* кода.

  Пројекат *Apache Cordova* одржава скуп додатака названих *додаци језгра* (енг. *Core Plugins*). Ови додаци обезбеђују апликацији приступ функционалностима уређаја као што су камера, контакти, компас, статус батерије и слично [19].

  Постоје и други додаци написани од стране трећих лица који обезбеђују везе до додатних функционалности уређаја, који нису увек доступни на свим платформама, а програмерима је на располагању и могућност да сами напишу додатак уколико за то постоји потреба.



Слика 4.1: Приказ архитектуре радног оквира *Cordova*

4.1 Развој апликације

4.1.1 Подешавање окружења

За развој апликација на радном оквиру *Apache Cordova* претходно је потребно инсталирати *Node*[1]. Помоћу њега се инсталира окружење за развој покретањем команде **sudo npm install -g cordova** на оперативним системима Линукс и *Mac OS*, или команде **npm install -g cordova** на оперативном систему *Windows*.

Прављење апликације се извршава командом **cordova create imeAplikacije com.example.imeAplikacije**. Након тога се добија апликација са структуром приказаној на слици 4.2. На радном оквиру *Apache Cordova* је могуће развијати апликацију која ће се извршавати на различитим платформама. На списку доступних платформи су: Андроид, *iOS*, *iBlackBerry*, *osx* и

[1] *Node* представља окружење за развој *JavaScript* апликација на серверској страни.

webos. Сваку платформу за коју се жели развијати апликација потребно је засебно додати у апликацију командом **cordova platform add imePlatforme**.

```
myapp/
|-- config.xml
|-- hooks/
|-- merges/
| | |-- android/
| | |-- windows/
| | |-- ios/
|-- www/
|-- platforms/
| |-- android/
| |-- windows/
| |-- ios/
|-- plugins/
  |--cordova-plugin-camera/
```

Слика 4.2: Структура *Cordova* апликације

У датотеци **config.xml** се наводе опште информације о апликацији попут имена, описа, аутора, платформи за које се апликација развија, коришћени додаци приликом развоја и слично. У директоријум **hooks** се смештају скрипте које служе за прилагођавање команда окружења командне линије. Специфични ресурси и датотеке (*HTML*, *CSS* и *JavaScript*) који су везани за одређену платформу се смештају у директоријум *merges*.

Директоријум **www** представља директоријум у коме се налазе датотеке над којима програмери највише раде током развоја *Cordova* апликација. У њему се налази датотека *index.html* која представља улазну тачку за апликацију, као и поддиректоријуми *js*, *css* и *img* у који се смештају датотеке наведених формата и слике које се смештају у последње наведени поддиректоријум.

Директоријум **platforms** садржи изворни код и скрипте за генерисање апликације за сваку платформу која је додата у апликацију. У директоријуму **plugins** се налазе сви додаци који се током развоја користе у оквиру апликације.

Тестирање апликације на емулатору или уређају се такође покреће из окружења командне линије. За то постоје команде **cordova emulate android** и **cordova run android**, којима је могуће мењати трећи параметар, у зависности од тога за који оперативни систем се апликација покреће.

4.1.2 Рад са ресурсима

За рад са нискама није пронађен адекватан додатак који би испунио очекивања. Испробан је додатак **cordova-plugin-localization-strings** који захтева прављење *json* датотека са нискама за сваки језик. Командом **cordova prepare android** аутоматски се креира датотетка *strings.xml* на путањи */values-kodJezola/strings.xml*. Међутим, из документације остаје нејасно како се дефинисане ниске користе у апликацији и да ли се користе у *JavaScript* датотекама, у *HTML* датотекма или их је могуће користити са оба места.

```
{
  "app" : {
    "app_name": "MATFApp  - React",
    "camera_and_memory": "Kamera i memorija"
    ...
  }
}
```

Пример кода 4.1: Изглед *rs.json* датотеке са нискама

Други приступ који се може пронаћи је да се за сваки језик праве посебне *HTML* датотеке па да се у зависности од изабраног језика отварају одговарајуће датотеке. На пример, уколико апликација треба да подржи српски и енглески језик онда треба направити датотеке */rs/index.html* и */en/index.html* које се отварају у зависности који језик је одабран. Ипак, овај приступ је доста компликованији поготово уколико постоји доста екрана у апликацији и велики број језика које треба подржати. Из тог разлога није узет као релевантан и није имплементиран у тест апликацији.

Рад са сликама је прилично једноставан захваљујући веб технологијама које се користе за развој. Препорука је да се слике чувају у директоријуму *www/img* али то не мора нужно бити испоштовано јер се слике користе тако што се користи њихова путања. У *HTML* датотекама користи се елемент *img* коме се као извор прослеђује путања до слике:

```
<img id="cameraImage" src="img/image_preview.png"></img>
```

У *JavaScript* датотекама потребно је дохватити елемент *img* помоћу методе *getElementById* којој се прослеђује идентификација елемента којег треба дохватити, а потом се том елементу проследи путања до слике.

```
var image = document.getElementById('cameraImage');
      image.src = imageData;
```

4.1.3 Рад са камером

За рад са камером потребно је инсталирати одговарајући додатак покретањем команде **cordova plugin add cordova-plugincamera** из окружења командне линије. Функција *takePicture* је задужена за покретање камере јер се у оквиру ње позива глобални објекат *navigator.camera*, који је обезбеђен од стране инсталираног додатка. Уколико сликање буде успешно, подаци о фотографији ће бити прослеђени функцији *onSuccess* у којој се фотографија поставља на одговарајући *HTML* елемент на графичко корисничком окружењу. Уколико је сликање неуспешно, биће позвана функција *onFail* у којој треба приказати одговарајућу грешку. Приликом покретања камере могуће је подесити неколико параметара. У тест апликацији је подешен квалитет слике (параметар *quality*) на вредност 50 (на скали од 1 до 100), параметар *cameraDirection* који одређује која камера ће бити коришћена (0 за предњу камеру или 1 за задњу), као и *destinationType* чија вредност је подешена на *FILE_ URI* што значи да ће као параметар функције *onSuccess* бити враћена путања до слике из меморије.

```
takePicture: function () {
    navigator.camera.getPicture(onSuccess, onFail, {
      quality: 50,
      cameraDirection: 1,
      destinationType: Camera.DestinationType.FILE_URI
    });

    function onSuccess(imageData) {
      var image = document.getElementById('cameraImage');
      image.src = "" + imageData;
    }

    function onFail(message) {
      alert(message);
    }
}
```

Пример кода 4.2: Коришћење функционалности камере

4.1.4 Дохватање локације

За дохватање локације корисника потребно је инсталирати одговарајући додатак покретањем команде **cordova plugin add cordova-plugin-geolocation** из окружења командне линије. Функција *getPosition* је задужена за дохватање географксе дужине и ширине који представљају локацију корисника. Помоћу глобално доступног објекта *navigator.geolocation* покреће се метода *getCurrentPosition* којој се прослеђују опције и повратне методе *onSuccess* и *onError*. Додатне опције које се прослеђују се подешавају за случај да се захтев не изврши у предвиђеном времену како би се користила последња позната локација. Уколико се захтев изврши на време позива се метода *onSuccess* са параметром *position* који у себи носи податке о тренутној локацији корисника.

```
getPosition: function () {
    var options = {
      enableHighAccuracy: true,
      maximumAge: 3600000
   }
   var watchID = navigator.geolocation.getCurrentPosition(onSuccess,
   onError, options);

   function onSuccess(position) {
      alert('Geografska sirina: ' + position.coords.latitude + '\n' +
         'Geografska duzina: ' + position.coords.longitude + '\n' +
         'Tacnost: ' + position.coords.accuracy + '\n');
   };

   function onError(error) {
      alert('Kod greske: ' + error.code + '\n' + ' Poruka: ' + error.
   message);
   }
}
```

Пример кода 4.3: Дохватање локације корисника

4.1.5 Рад са подацима и мрежом

Дохватање података са сервера се може урадити на више начина користећи стандардне веб технологије. У тест апликацији је за рад са мрежом искоришћен објекат *XMLHttpRequest* помоћу којег је извршен позив за дохватање слика са сервера. Над објектом *XMLHttpRequest* се позива метода *send* која шаље захтев серверу. Уколико је сервер прихватио захтев и вратио одговор, повратни подаци су доступни у оквиру *onload* методе у оквиру параметра *request.responseText*. Уколико је ипак дошло до одређене грешке приликом слања захтева серверу позива се метода *onerror* у којој се могу информисати корисници да је дошло до одређене грешке.

```
...
var request = new XMLHttpRequest();
request.open('GET', url, true);

request.onload = function () {
    if (request.status >= 200 && request.status < 400) {
        var data = JSON.parse(request.responseText);
        getAllFlickrImages(data.photos.photo.slice(1,10));
    } else {
        alert('Response text: ' + request.responseText);
    }
};

request.onerror = function () {
    alert('Error');
};

request.send();
```

Пример кода 4.4: Дохватање података са сервера преко објекта *XMLHttpRequest*

Функција *appendImage* која је приказана у примеру кода 4.5 се позива за сваку слику добијену са сервера. Њена улога је да на постојећи *div* елеменат са идентификацијом *flickrImages* надовеже сваку од слика користећи *img* елеменат који је изгенерисан на основу путање и идентификације слике.

```
appendImage: function(imagePath, id) {
    var imageTag = "<img id = 'flickrimage_'" + id + " src='" +
   imagePath + "' width='100' height='100'/>";
```

```
    $("#flickrImages").append(imageTag);
},
```

Пример кода 4.5: Приказ слика у оквиру *HTML* странице

4.2 Закључак

Apache Cordova је један од најстаријих и најкоришћенијих радних оквира за израду апликација за оперативне системе за мобилне уређаје. Представља једноставан начин да веб програмери, али и сви они који су упознати са технологијама *CSS3*, *HTML5* и *JavaScript*, развијају апликације за мобилне уређаје. Приликом развоја може се користити било који од мноштва доступних додатака који покривају скоро све функционалности које може да пружи и стандардни развој. Ипак нису сви додаци ажурирани на последњу верзију те се може десити да апликација престане да ради у неким случајевима коришћења. Из тог разлога није тако редак случај да програмери морају да дорађују неки од доступних додатака како би га или поправили или прилагодили неким својим потребама.
Препорука је да се овај радни оквир користи када је потребно брзо направити прототип апликације на различитим платформама. Предност у односу на стандардни развој представља лакше руковање сликама јер је једноставније руковање користећи *CSS* него прављењем различитих резолуција слика и њиховим смештањем по различитим директоријумима, што је својствено стандардном развоју за оперативни систем Андроид.

Коришћење овог радног оквира се не препоручује код комплекснијих апликација и апликације које поседују захтевну графику јер би се перформансе истих значајно смањиле у односу на стандардне апликације. Због самих карактеристика језика *JavaScript*, много је теже модуларизовати апликацију у односу на писање апликација на стандардним платформама. На екрану се не исцртавају стандардни елементи графичко корисничког окружења већ се све исцртава у оквиру веб погледа. Такође, *JavaScript* не подржава вишенитно извршавање, па цела апликација извршава на главној нити што није у духу развоја апликација за мобилне уређаје.
Документација некада није у потпуности комплетна што може отежавати развој, а такође не постоје никаква упутства или добре праксе по којима би се програмери водили приликом развоја апликација.

Глава 5

React Native

React Native је радни оквир заснован на програмском језику *Javascript*, који служи за развој апликација за оперативне системе *iOS* и Андроид. Такође, представља надоградњу *JavaScript* библиотеке *React*[1] развијене од стране компанија *Facebook* и *Instagram* [20]. *Facebook* је библиотеку *React Native* објавио 2015. године под лиценцом отвореног кода, а пре тога је већ била коришћена у оквиру компаније чак и у продукционим верзијама. Приликом објаве су објаснили да нису ставили нагласак на принцип *Пиши једном, извршавај било где*, који је чест код других сличних радних оквира, већ су желели да омогуће програмерима да праве апликације за различите платформе без потребе да уче различит скуп техонологија за сваку од платформи. Овај принцип су дефинисали као *Научи једном, пиши било где* [21].

За разлику од других радних оквира, попут *Cordova*-е или неког изграђеног над њом попут *Ionic*-а и *Sencha Touch*, где већи део апликације представља *HTML* код у оквиру веб погледа, *React Native* је сличнији стандардном развоју. То се постиже јер се дефинисане компоненте исцртавају као стандардни елементи. Тако се компонента *<View>* на платформи *iOS* исцртава као *UIView*, док се на платформи Андроид исцртава као *android.view*.

Правило је да се нове верзије радног оквира обајвљују сваког месеца те је тренутно актуелна верзија 0.47 што говори да још увек није објављена прва главна верзија. Такође, дешава се да у новим верзијама постоје значајне измене оквира које нису у сагласности са претходним верзијама, што је и очекивано јер је у питању млад радни оквир који се у значајној мери развија од стране заједнице програмера [22].

[1] *React* је *JavaScript* библиотека отвореног кода за израду графичко корисничких окружења.

5.0.1 Карактеристике радног оквира

Као што је наведено раније, *React Native* почива на библиотеци *React*. Главна разлика између њих је у томе што *React Native* користи стандардне компоненте као градивне елементе уместо веб елемената које користи *React*. Основни концепти које је битно разумети су компоненте, *JSX*, реквизити (енг. *props*) и стања (енг. *state*).

Компоненте представљају основни градивни елемент *React Native* апликација. Све што се исцртава на екрану је неки тип компоненте. Компоненте могу бити врло једноставне тако да се састоје од једног елемента, а могу бити и компликованије са већим бројем елемената. Једина ствар која је обавезна код прављења копоненти је постојање функције *render* која враћа *JSX* код који треба да се исцрта.

```
class HomeScreen extends React.Component {
    render() {
        return (
            JSX code
        )
    }
}
```

Пример кода 5.1: Дефинисање компоненте

JSX је додатак на језик *JavaScript* који омогућује коришћење *HTML* синтаксе како би се исцртали елементи графичко корисничког окружења.

```
<View>
    <Button
        onPress={() => navigate('CameraAndMemory')}
        title="Open Camera"
    />
</View>
```

Пример кода 5.2: Део *JSX* кода коришћеног у апликацији

Реквизити су параметри који служе да би се компоненте прилагодиле одређеном изгледу и особинама. У примеру кода 5.2 се приликом дефинисања дугмета (енг. *Button*) користи реквизит *title* који дефинише који ће текст бити исписан на дугмету. Сваки елемент који може бити дефинисан у оквиру *JSX* кода има дефинисан

скуп реквизита који могу бити коришћени за њега. Поред тих унапред дефинисаних реквизита, програмерима је омогућено дефинисање и коришћење нових реквизита. Њима се може приступити у функцији *render* преко команде *this.props*.

```
Location.propTypes = {
    provider: MapView.ProviderPropType
};
...
render() {
    <MapView
        provider={this.props.provider}
    >
}
```

Пример кода 5.3: Пример прилагођеног реквизита

Поред реквизита и **стања** могу контролисати компоненте. За разлику од реквизита који се постављају једном и неће се мењати кроз животни циклус компоненте, стања се користе за податке који ће се мењати током извршавања. Углавном се стања иницијализују у конструктору, а позивом методе *setState*, сваки пут када се подаци мењају, долази до поновног исцртавања компоненте.

У примеру код 5.4 дефинисани су параметри географске ширине и дужине као стање и њихова вредност је подешена у конструктору. Метода *onRegionChange* се позива када корисник промени своју локацију и у њој се позива метода *setState* након које ће се освежити графичко корисничко окружење. Разлика ће бити само у подацима, тј. у маркеру који је дефинисан у *JSX* коду, јер ће сваки следећи пут користити нове податке који су прослеђени приликом позива методе *setState*.

```
constructor(props) {
    this.state = {
        region: {
          latitude: LATITUDE,
          longitude: LONGITUDE
        },
    };
}
onRegionChange(region) {
  this.setState({ region });
}
```

```
render() {
    return (
        <MapView.Marker
            coordinate={{latitude: this.state.region.latitude,
            longitude: this.state.region.longitude}}
        />
    )
}
```

Пример кода 5.4: Пример коришћења стања

5.1 Развој апликације

5.1.1 Подешавање окружења

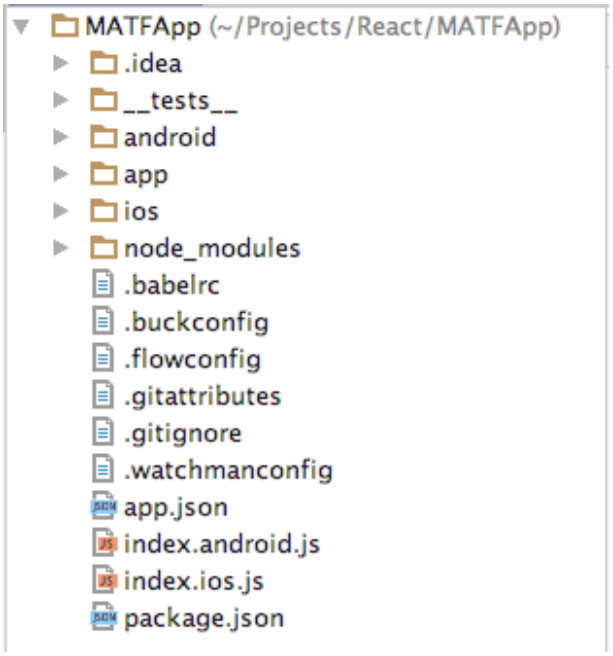
За потпуно подешавање окружења за развој *React Native* апликација за оперативни систем Андроид, потребно је претходно инсталирати *Node*, *Java Development Kit*[2], *Android Studio* и окружење командне линије за радни оквир *React Native*. За развој је потребно инсталирати, тренутно најновију, верзију 8 *JDK*-а.

Android Studio је програмско окружење за развој апликација за оперативни систем Андроид. Настао је на основама окружења *IntelliJ IDEA* компаније *JetBrains*, а званично је представљен на конференцији *Google I/O* у мају 2013. године [23]. Сама инсталација овог окружења са собом доноси и инсталацију последње верзије скупа алата за развој Андроид апликација (енг. *Android SDK*). Последња верзија *SDK*-а је издата у априлу 2017. године под ознаком 26.0.2. Ипак, за развој *React Native* апликација потребно је користити верзију 23.0.1, из октобра 2015. године, која се односи на *Android Marshmallow*.

Акције за прављење, покретање, изградњу (енг. *build*), брисање апликације се покрећу у оквиру командне линије. Иницијално прављење апликације се покреће командом: **react-native init ImeAplikacije**. За извршавање апликације на уређају који покреће Андроид оперативни систем потребно је покренути команду **react-native run-android**, која уједно и изграђује апликацију и затим је покреће на уређају или на емулатору.

[2] *Java Development Kit* је сет алата који се користе за превођење и извршавање Јава програма.

Након иницијаног прављења апликације добија се структура апликације приказана на слици 5.1. У директоријуму **android/** се налази основни Андроид код који се углавном не мења током развоја *React Native* апликација. Аналогно, **ios/** представља директоријум у коме је смештен основни *iOS* код. Датотеке **index.android.js** и **index.ios.js** представљају улазне тачке за Андроид и *iOS* апликације. Чест је случај да обе датотеке само преусмеравају улазак у апликације на неку другу датотеку која би била заједничка за обе платформе, како се исти код не би писао два пута.



Слика 5.1: Структура *React Native* апликације

Датотека **app.json** има у себи основне информације о апликацији попут имена апликације и имена које ће бити исписано на екрану уређаја и слично. Све датотеке које корисник додаје током развоја се додају у директоријум **app**. У њему програмери могу да праве поддиректоријуме како би организација датотека била смислена и прегледна.

5.1.2 Рад са ресурсима

Постоји више могућности за рад са нискама на овом радном оквиру. У тест апликацији је коришћен додатак *React Native Localization* који омогућује програмерима да једноставно користе преводе за различите језике у апликацији. Поред овог додатка, програмери се одлучују и за додатак *React Native I18N* који се користи на сличан начин као и претходно наведени.

Да би се додатак могао користити у апликацији потребно је га претходно убацити у пројекат командом **npm install react-native-localization - -save**. Саме ниске се чувају у једној датотеци чије име и локација у оквиру апликације могу бити произвољни. У тест апликацији ниске се чувају у датотеци **strings.js** у директоријуму са путањом ***appHome/app/config***. У самој датотеци се дефинишу преводи за све језике који ће бити омогућени корисницима апликације. Уколико није другачије наглашено кроз код, иницијално ће бити подешени преводи оног језика који је подешен на самом уређају, а уколико преводи за тај језик не постоје биће одабран језик чији су преводи наведени први у датотеци. Наравно, омогућен је и одабир језика кроз код позивом методе **setLanguage(languageCode)**.

```
import LocalizedStrings from 'react-native-localization';

export let strings = new LocalizedStrings({
 rs:{
    app_name:"MATFApp - React",
    camera_and_memory:"Kamera i memorija",
    ...
 },
});
```

Пример кода 5.5: Део датотеке *strings.js* са преводима

Ниске се у коду користе тако што се дохватају по кључу по коме су дефинисане:

```
<Button
    onPress={() => navigate('CameraAndMemory')}
    title={strings.camera_and_memory}
/>
```

За рад са фотографијама развијен је униформан систем. Фотографије се постављају на елемент *Image* тако што се прослеђује путања до њих:

```
<Image source={require('./image.png')} />
```

Пример кода 5.6: Пример коришћења фотографије

У примеру кода 5.6 фотографија ће бити тражена у истом директоријуму у коме се налази компонента којој је фотографија потребна. На програмерима је да одлуче да ли ће све фотографије чувати у једном директоријуму или ће фотографије чувати у истим директоријумима у којима се налазе компоненте. Додатна олакшавајућа

околност је што не мора да се води рачуна о резолуцији фотографија и о томе за коју платформу је фотографија припремљена. На пример, уколико постоје фотографије са именима *image.android.png* и *image.ios.png* или *image@2x.png* и *image@3x.png* и даље се фотографија користи на исти начин као у примеру кода 5.6, а систем сам препознаје резолуцију и систем уређаја и бира одговарајућу фотографију.

5.1.3 Рад са камером и меморијом

За коришћење функција камере у тест апликацији је коришћена компонента **react-native-camera**. Исталација компоненте је иста као и за друге компоненте и обавља се командом: **npm install react-native-camera –save**. У примеру кода 5.7 приказана је употреба компоненте *Camera* у тест апликацији.

```
<Camera
    ref={(cam) => {
        this.camera = cam;
    }}
    aspect={Camera.constants.Aspect.fill}>
    <Text onPress={this.takePicture.bind(this)}>{strings.take_picture}</
   Text>
</Camera>
```

Пример кода 5.7: Употреба компоненте *Camera*

Параметар *aspect* чија је вредност подешена на *fill*, одређује на који начин ће камера бити приказана на екрану у оквиру родитељског погледа (енг. *view*). Вредност *fill* означава да ће оригинална размера погледа камере бити сачувана и да ће слика бити приказана у тој размери.

Приликом позивања методе *capture* над компонентом *camera* могуће је проследити низ опција које прилагођавају процес фотографисања. У тест апликацији је подешена локација на којој ће се чувати фотографија подешавањем параметра *captureTarget*. Тај параметар је подешен на вредност *Camera.constants.CaptureTarget.disk* што је препоручена вредност како би се побољшале перформансе приликом фотографисања јер ће се тако најбрже сачувати фотографија у меморији. Поред ове опције могуће је сачувати фотографију у привременој меморији подешавањем вредности *Camera.constants.CaptureTarget.temp* или у галерији слика, подешавањем вредности

Camera.constants.CaptureTarget.cameraRoll. Поред овог параметра, подешен је и параметар *options.type* који одређује да ли се користи предња или задња камера. Након чувања фотографије на одабраној локацији, путањи се може приступити преко параметра *data*, који у себи има податке о фотографији међу којима су и путања, димензије фотографије, величина и слично.

```
takePicture() {
    const options = {};
    options.captureTarget = Camera.constants.CaptureTarget.disk;
    options.type = Camera.constants.Type.back;
    this.camera.capture({metadata: options})
      .then((data) => this.setState({ path: data.path }))
      .catch(err => console.error(err));
  }
```

Пример кода 5.8: Метода која извршава фотографисање

Највећа мана коришћене компоненте је та што се скуп опција и доступних функционалности значајно разликује за оперативне системе *iOS* и Андроид, при чему је скуп опција доста богатији за оперативни систем *iOS*.

5.1.4 Дохватање локације

За рад са геолокацијом је обезбеђен посебан скуп метода под именом *Geolocation API*. Као и код других радних оквира и овде је обавезно да се захтева дозвола од оперативног система Андроид за дохватање корисникове локације додавањем захтева *android.permission.ACCESS_FINE_LOCATION*. Метода *watchPosition* која припада поменутом скупу метода враћа податке о локацији сваки пут када се локација корисника промени. Као реквизит (енг. *prop*) на овој компоненти је коришћен објекат *region* са два параметра: *latitude* и *longitude* чије вредности се подешавају сваки пут када метода *watchPosition* врати нову локацију.

```
this.state = {
      region: {
        latitude: LATITUDE,
        longitude: LONGITUDE
```

```
    },
};
...
componentDidMount() {
    this.watchID = navigator.geolocation.watchPosition((position) => {
        let region = {
          latitude:       position.coords.latitude,
          longitude:      position.coords.longitude,
        }
        this.onRegionChange(region);
    });
}

onRegionChange(region) {
    this.setState({ region });
}
```

Пример кода 5.9: Дохватање локације корисника

За приказ корисникове локације на мапи је искоришћена компонента *react-native-maps*, која у себи има уграђену подршку за рад са мапама, као и са додацима на истим. У тест апликацији су искоришћене компоненте *MapView* и *MapView.Marker*. Компонента *MapView* је основни елелмент који приказује мапу на екрану и могуће га је подешавати параметрима. Параметар *provider* означава извор података за мапе, што је у случају тест апликације *GoogleMaps* који је уобичајени извор на систему Андроид. Поред тога подешен је иницијални регион који ће бити приказан на мапи док се не учита права локација корисника, као и позив методе која рукује подацима када се промени регион на мапи. Компонента *MapView.Marker* се користи за постављање ознака на мапи, а у тест апликацији је искоришћена за иницијални приказ локације корисника. За њега је довољно подесити координате на којима треба да буде приказан на мапи, што се може видети у примеру кода 5.10.

```
<MapView
    provider={this.props.provider}
    ref={ref => { this.map = ref; }}
    mapType={MAP_TYPES.TERRAIN}
    style={styles.map}
    initialRegion={this.state.region}
    onRegionChange={region => this.onRegionChange(region)}>
    <MapView.Marker
```

```
        coordinate={{latitude: this.state.region.latitude,
        longitude: this.state.region.longitude}}
    />
</MapView>
```

Пример кода 5.10: Коришћење компоненти *MapView* и *MapView.Marker*

5.1.5 Рад са подацима и мрежом

За рад са мрежом препоручено је коришћење библиотеке *Fetch API*, која програмерима даје могућност рада са *HTTP* захтевима, али и могућност обраде одговора са сервера. Користи се тако што се позива глобална метода *fetch()* која на једноставан начин асинхроно дохвата ресурсе са сервера. Асинхроност се постиже захваљујући систему обећања (енг. *promise*) чији објекти могу представљати неуспешан завршетак операције или резултирајућу вредност операције. Из добијеног одговора са сервера, над којим се позива метода *.json()*, једноставно је приступити подацима које је потребно приказати на компоненти коришћењем кључева који се враћају са сервера: **responseJson.photos.photo**.

```
return fetch('https://api.flickr.com/services/rest/?method=flickr.photos
    .search...')
    .then((response) => response.json())
    .then((responseJson) => {
        let ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1
    !== r2});
        this.setState({
            isLoading: false,
            dataSource: ds.cloneWithRows(responseJson.photos.photo),
        }, function() {
        });
    })
    .catch((error) => {
        console.error(error);
    });
}
```

Пример кода 5.11: Дохватање слика са портала *Flickr*

5.2 Закључак

React Native је свакако занимљив радни оквир који пре свега веб програмерима даје могућност развоја апликација за оперативне системе који се извршавају на мобилним уређајима. Највећа предност у односу на стандардни развој, му је свакако брз развој који омогућава програмерима ефикасно дељење кода између платформи. Предност у самом раду је свакако што је довољно само сачувати измене на датотеци и промене ће одмах бити видљиве на екрану уређаја или симулатора, што је прилично револуционарно у односу на стандардни развој где је после било какве промене на датотеци потребно наново компајлирати целу апликацију што може потрајати и по неколико минута.

Потенцијалне мане су му што је у питању још увек млад радни оквир који се и даље активно развија, те су могуће и значајније промене у оквиру самог радног оквира. Већ се дешавало да најновије верзије радног оквира нису компатибилне са неким мало старијим верзијама, те остаје на програмерима да ажурирају код како би могли да користе најновије могућности радног оквира. Такође сами додаци које развија заједница програмера нису увек у складу са последњим верзијама система Андроид и *iOS*, а врло често ни сами додаци не обезбеђују исти скуп функционалности за обе платформе, већ су често поједине функције доступне само на једној платформи.

Глава 6

Закључак

Радни оквири за израду апликација за оперативне системе за мобилне уређаје су углавном још увек младе технологије које сваком новом верзијом нуде више опција програмерима. Главни утисак је да и поред свих предности, које су побројане за сваки обрађени радни оквир, развој на радним оквирима ипак прилично заостаје за стандардним развојем када су у питању перформансе апликација и лакоћа приступа свим функционалностима уређаја. Такође, радни оквири углавном не прате довољном брзином најновије верзије оперативних система за мобилне уређаје и нове могућности које они доносе, што није случај са стандардним развојем.

Закључак је да развој на радним оквирима, уместо стандардног развоја може донети корист у случајевима када програмери довољно добро познају технологију која се користи приликом израде апликација на одређеном радном оквиру, као и када је битно да се апликација развије у кратком временском року за више платформи а притом захтеви у апликацији нису претерано велики када је у питању графичко окружење и комплексност апликације. У случају да је графичко окружење веома битно и притом захтевно, а поготово ако се развија пословна апликација, препорука је да се користи нативни развој због перформанси апликације на уређајима и стабилнисоти развоја на дужи период када су у питању подршка и могућност приступа најновијим функционалностима оперативних система.

6.1 Будући рад

Будући рад се своди на даље истраживање могућности сваког појединачног оквира и покушаја прављења целокупне апликације на одређеном радном оквиру. Лични утисак је да је највише пажње од обрађених радних оквира привукао *React Native*, пре свега због брзине развоја, приступачности додатних компоненти за развој, велике заједнице програмера и прилично добро написане документације. Иако циљ рада није био издвојити један радни оквир, *React Native* је издвојен јер ће знање стечено радом на овом раду представљати добру основу за будући развој апликација на њему.

Библиографија

[1] John Bristowe. What is a hybrid mobile app?, март 2015. URL http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/.

[2] Carey Wodehouse. 7 reasons why facebook's react native is the future of hybrid app development, 2017. URL https://www.upwork.com/hiring/mobile/react-native-hybrid-app-development/.

[3] TechInAsia Taylor Milliman. After building my first react native app, i'm now convinced it's the future, мај 2017. URL https://www.techinasia.com/talk/building-react-native-app-future.

[4] Sencha. Using device features in your touch app. URL https://docs.sencha.com/touch/2.4/tutorials/cordova_camera.html.

[5] Vasilina Bezuglaya. Comparing five popular frameworks for mobile development in 2017, јун 2016. URL https://www.graph.uk/blog/mobile-development-frameworks-in-2016.

[6] Shane Conder Lauren Darcey. *Android Wireless Application Development*. Addison-Wesley, Boston, 2009.

[7] Mehul Rajput. Tracing the history and evolution of mobile apps, новембар 2015. URL https://tech.co/mobile-app-history-evolution-2015-11.

[8] Jay Alabaster. Android founder: We aimed to make a camera os, април 2013. URL http://www.pcworld.com/article/2034723/android-founder-we-aimed-to-make-a-camera-os.html.

[9] Dave Chaffey. Mobile marketing statistics compilation, март 2017. URL http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/.

[10] James Vincent. 99.6 percent of new smartphones run android or ios, фебруар 2017. URL https://www.theverge.com/2017/2/16/14634656/android-ios-market-share-blackberry-2016.

[11] Statcounter. Mobile operating system market share worldwide, јун 2017. URL http://gs.statcounter.com/os-market-share/mobile/worldwide.

[12] Android. The activity lifecycle, јун 2017. URL https://developer.android.com/guide/components/activities/activity-lifecycle.html.

[13] Xamarin. The xamarin story, јун 2017. URL https://www.xamarin.com/about.

[14] Oracle. Jdk 6 and jre 6 supported locales, јун 2017. URL http://www.oracle.com/technetwork/java/javase/locales-137662.html.

[15] Android inc. Supporting multiple screens, јун 2017. URL https://developer.android.com/guide/practices/screens_support.html.

[16] Xamarin. User interface - creating the app ui, јун 2017. URL https://developer.xamarin.com/guides/android/user_interface/.

[17] Apache Cordova. Apache cordova 7.0.0, мај 2017. URL https://cordova.apache.org/news/2017/05/04/cordova-7.html.

[18] Adobe PhoneGap. Phonegap, cordova, and what's in a name?, март 2012. URL https://phonegap.com/blog/2012/03/19/phonegap-cordova-and-whate28099s-in-a-name/.

[19] Apache Cordova. Platform support, јул 2017. URL https://cordova.apache.org/docs/en/latest/guide/support/index.html#core-plugin-apis.

[20] Margi Murphy. What is react native? - facebook: 'if we work together in the open, we can advance the state of technology together', септембар 2015. URL http://www.techworld.com/apps-wearables/what-is-react-native-3625529/.

[21] Tom Occhino. React native: Bringing modern web techniques to mobile, март 2015. URL https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/.

[22] GitHub. React native. URL https://github.com/facebook/react-native.

[23] Android Developers Blog. Android studio: An ide built for android, мај 2013. URL https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html.