Математички факултет Универзитет у Београду

Никола Самарџија

Примена развојног алата *Eclipse* за развој апликације за наручивање и претраживање хране за рачунаре у покрету са оперативним системом *Android*

Мастер рад

Београд, 2014

Мастер рад

Студент:	Никола Самарџија 1010/2009
Наслов:	Примена развојног алата *Eclipse* за развој апликације за наручивање и претраживање хране за рачунаре у покрету са оперативним системом *Android*
Ментор:	др Саша Малков, доцент Универзитет у Београду – Математички факултет
Чланови комисије:	др Ненад Митић, ванредни професор Универзитет у Београду – Математички факултет, др Владимир Филиповић, ванредни професор Универзитет у Београду – Математички факултет
Датум:	xx.xx.201x

Садржај

1. Увод

У последњих тридесет година долази до великог развоја информационих технологија. Поред развитка хардвера, долази до великог развитка софтвера. За веома кратко време прешли смо пут од појаве првих персоналних рачунара, оперативних система, широко распрострањеног интернета и појаве мобилних телефона до тога да већина цивилизованог света поседује уређај који може стати који обједињује технологије које смо споменули. Наравно мислимо на паметне телефоне и таблете.

Према подацима из јануара 2014 године, 6,81 милијарди становника земље користи мобилни телефон [1]. Процењује се да има око 1,89 милијарди корисника паметних телефона, а да ће се већ следеће године тај број попети на 2 милијарде [2]. Оперативни систем *Android* [3] представља најраспрострањенији оперативни систем за паметне уређаје са уделом од 78,1 % [4]. Ови подаци говоре о једном великом тржишту и могућностима које пружа.

У овом раду, аутор ће представити процес развоја апликација за оперативни систем *Android*. Пре свега, биће речи о овом оперативном систему, његовом историјату, току развоја, као и о његовој садашњости. Аутор ће описати алате који су неопходни за креирање апликација. Један део рада обухватаће опис развојног алата *Eclipse* [5], тј. његовог додатка који омогућава израду апликације за поменути оперативни систем. Аутор ће се потрудити да приближи развојно окружење, његову структуру и организацију кода. Рад обухвата и израду апликације за наручивање хране за уређаје који садрже оперативни систем *Android*. Биће приказана методологија рада која је допринела изради једног оваквог пројекта.

Технологије које су коришћене приликом израде овог рада су развојни алат *Android SDK* [6], програмски језици *Java, PHP, MySql* и алат за пројектовање *Visual Paradigm.*

2. Технолошки оквир

Рачунарска платформа обухвата хардверску архитектуру и софтверску подршку који су потребни за покретање и извршавање апликација. У даљем тексту биће описана платформа *Android*, која обухвата мобилне уређаје и оперативни систем *Android*. За израду ће се користити развојни алат *Eclipse* и његови додаци који су неопходни за развој једне овакве апликације.

2.1 Оперативни систем *Android*

Оперативни систем *Android* (у даљем тексту *Android*) је тренутно најраспрострањенији оперативни систем за мобилне уређаје. Заснован је на *Linux* језгру [7] и прилагођен је тако да се може користити на већини мобилних уређаја, укључујући мобилне телефоне, таблет рачунаре, преносиве рачунаре, нетбук рачунаре, смартбук рачунаре, читаче електронских књига, па чак и ручне сатове и друге уређаје.

2.1.1 Кратак историјат

Android је првобитно развијан у компанији *Android Inc* [7] која је најпре била финансијски подржана и коначно купљена 2005. године од стране компаније *Google* [9]. У то време мало се знало о пословима које је обављао *Android Inc*, осим да су радили на програму за мобилне телефоне. Након куповине компаније, постало је јасно да *Google* планира да се фокусира на развој платформе за мобилне уређаје.

У *Google*-у је тим предвођен Андријем Рубином [10] развио оперативни систем за мобилне телефоне темељен на *Linux*-у. Произвођачима мобилних телефона представљен је као флексибилан и лако надоградив систем. Почеле су гласине стучне јавности да је *Google* већ склопио партнерство са низом хардверских и софтверских компанија у циљу промоције новог оперативног система. Дана 5. новембра 2007. основан је савез *Open Handset Alliance (OHA)* [11]. Укључивао је 34 компаније, а једни од њих су: *Google, HTC, Intel, Motorola, Qualcomm, TMobile, Sprint Nextel* и *NVIDIA*. Заједнички циљ је био развој отворених стандарда за мобилне уређаје. Већ при оснивању су представили и свој први производ: *Android*.

Први комерцијално доступан мобилни уређај који је користио *Android* је био телефон *HTC Dream* [12], који је пуштен у продају у октобру 2008. године. Телефон је користио *Android* верзију 1.6 названу *Donut* [13]. Пре ове верзије су постојале и радне верзије 1.1, 1.2, 1.5 названа *Cupcake* [14], а након ње су редом излазиле бројне верзије свака носећи са собом значајна побољшања како системска тако и визуална.

2.1.2 Пројекат отвореног кода

Android је развијан као пројекат отвореног кода (енг. *Open Source*) пројекат [15] што значи да је његов изворни код доступан свима. Као такав, пројекат је окупио велики број програмера и ИТ стручњака који свакодневно раде на његовом развоју. Сходно политици конзорцијума који ради на развоју *Android*-а и политици пројекта отвореног кода истичу се следеће карактеристике овог оперативног система [16]:

- Отвореност: *Android* је изграђен тако да омогућава програмерима стварање апликација које у потпуности користе све што уређај нуди. Апликација може користити функције језгра мобилног уређаје као што су позивање, слање текстуалних порука, камеру и остале функције. Допушта програмерима да створе богате и сложене корисничке програме. Почива на *Linux* језгру. Користи сопствену виртуалну машину [17] која је дизајнирана да оптимизује меморијске и хардверске ресурсе уређаја. *Android* се лако може проширивати и унапређивати. Платформа може да настави да расте све док програмерска заједница ради заједно и развија нове и иновативне апликације за мобилне уређаје.

- Све апликације су једнаке: *Android* не разликује апликације језгра и апликације независних произвођача, што се тиче приступа могућностима уређаја. И једне и друге апликације имају једнак приступ ресурсима што дозвољава корисницима употребу широког појаса апликација и услуга. Уређаје, који користе ову платформу, корисници у потпуности могу прилагодити својим захтевима.

- Рушење граница у стварању апликација: *Android* помера границе стварања нових и иновативних апликација. На пример, програмер може комбиновати информације са интернета са подацима на уређају појединог корисника - као што су контакти, календар или локација. За веома кратко време може се направити апликација која нпр. омогућава кориснику да види локацију својих пријатеља и да га обавести када су у близини.

- Брзо и једноставно развијање апликација: пружен је приступ широком распону библиотека и алата који се могу користити за израду разноврсних апликација. Огромну подршку пружа обимна и прецизна званична документација [18] као и *Android* заједница која обухвата форуме и дискутне групе и чини велики број програмера.

- Високо квалитетни графички приказ и звук: подржана је *2D* векторска [19] и *3D OpenGL* [20] графика и уграђени су кодеци често коришћених аудио и видео формата. Мултимедијски садржаји су широко подржани.

- Компатибилност са већином садашњег и будућег хардвера: *Android* омогућава преносивост *Android*-ових апликација на *ARM* [21], *x86* [22] и друге хардверске архитектуре, као и прилагођавање система улазним и излазним компонентама.

Android је био критикован зато што у почетку није био у потпуности отворен. Тачније, неки делови развојне библиотеке (енг. *SDK – Software Development Kit*) [23] су у почетку остали затворени. Претпостављало се да је то због тога што *Google* желео да контролише платформу и њен развој. Међутим, *Google* је након тога објавио вест да ће сви делови оперативног система бити отворени под лиценцом *Apache* [24] где је то могуће или под лиценцом *GPL* [25] на другим местима. У октобру 2008. године, *Google* је отворио сав изворни код, који пре тога није био доступан, под лиценцом *Apache*. Са овом лиценцом, програмери могу додавати властита проширења која не морају поделити са остатком заједнице. Очекује се да *Google*-ови доприноси платформи буду отворени, а са друге стране да се развијају друге гране под различитим лиценцама. Ово омогућава компанијама које се баве развојем мобилних уређаја да прилагоде *Android* својој визији графичког интерфејса и функционалности.

2.1.3 Архитектура оперативног система *Android*

Архитектура *Android*-а се састоји од више слојева, од којих сваки садржи одређене компоненте. Сваки слој архитектуре пружа различите сервисе слоју изнад њега и на тај начин омогућава функционисање читавог система. Шематски приказ архитектуре оперативног система *Android* приказана је на слици 1.

Слика 1. Архитектура *Android* оперативног система

2.1.3.1 Linux језгро

У основи оперативног система *Android* налази се модификовано *Linux* језгро које обезбеђује комуникацију са хардвером и управљање процесима и меморијом. Написано је на програмском језику *C* [26]. Модификације у односу на класично *Linux* језгро се највише огледају у прилагођавању раду са мобилним уређајима. Мобилни уређаји имају значајна ограничења у погледу меморије и процесора и самим тим је потребно прилагодити се тим ограничењима. Неке од најважнијих компоненти овог слоја, поред драјвера за коришћење различитих компоненти хардвера, су:

- управљање напајањем (енг. *Power management*) – оптимизује потрошњу струје. Уколико се нека апликација више не користи, систем је аутоматски суспендује да не користити ресурсе као што су процесорско време и електричну енергију, све док се апликација не активира поново;
- повезивач (енг. *Binder*) – омогућава међу процесну комуникацију између апликација и сервиса, који могу радити у одвојеним процесима и размењивати податке.

2.1.3.2 Библиотеке

Изнад језгра, налазе се системске библиотеке писане на програмском језику *C* и *C++*. Оне чине следећи слој у архитектури. На њих можемо гледати као на низ инструкција које помажу уређају да обради различите податке. На пример, библиотека која се бави мултимедијом подржава пуштање и снимање аудио или видео садржаја као и низ акција везаних за обраду слике. Неке од компоненти библиотеке су:

- *Libc* - имплементација стандардне системске библиотеке изведена из оперативног система BSD [27];
- *Surface manager* - библиотека одговорна за правилно исцртавање различитих компоненти апликације у простору и времену;
- *OpenGL* - пружа подршку за *2D/3D* графику;
- *SGL* (енг. *Scalable Graphics Library*) - представља *2D* библиотеку на којима је утемељена већина апликација;
- *Media Framework* - скуп кодека за снимање и репродукцију аудио формата, видео формата и слика;
- *FreeType* - библиотека која служи за растеризацију векторских фонтова
- *SSL* (енг. *Secure Socket Layer*) - омогућује безбедну комуникацију преко интернета;
- *SQLite* - библиотека која омогућава рад са локалном базом података;
- *WebKit* - библиотека која подржава *JavaScript* [28] и остале веб стандарде на мобилном уређају;

2.1.3.3 *Android runtime*

На истом нивоу као и библиотеке, *Android* окружење обезбеђује скуп основних компоненти које омогућавају програмерима да пишу *Android* апликације коришћењем програмског језика *Java*. *Android* окружење садржи виртуалну машину Далвик (*DVM,* енг. *Dalvik Virtual Machine*) [29]. Виртуална машина омогућава свакој *Android* апликацији да се извршава у сопственом процесу, са сопственом инстанцом виртуалне машине. *Android* апликације се преводе у датотеке са екстензијом *.dex* (енг. *Dalvik Executable*). Далвик је специјализована виртуална машина, пројектована специјално за *Android* и оптимизована за мобилне уређаје који користе батерију при раду и имају ограничене меморијске ресурсе и спорије процесоре.

Основне библиотеке (енг. *Core libraries*) [30] су писане на програмском језику *Java* и обухватају најосновније класе као што су класе за руковање колекцијама, класе за комуникацију с околином и слично.

2.1.3.4 Слој апликацијских оквира

Слој апликацијских оквира (енг. *Application Framwork*) написан је на програмском језику *Java* и садржи проширив скуп програмских компоненти. Омогућава коришћење различитих могућности оперативног система *Android*, тако да програмери могу да их користе у својим апликацијама. Неке од најважнијих компонената су:

- *Activity Manager* - управља животним циклусом апликације;
- *Package Manager* - управља програмским пакетима и садржи информације о инсталираним апликацијама;
- *Window Manager* - управља прозорима апликације;
- *Telephony Manager* - садржи програмске интерфејсе (енг. *API - Application Programming Interface*) [31] који се користе при изради апликација за управљање телефонским позивима;
- *Content Providers* - омогућава обједињено коришћење података који се налазе на телефону;
- *Resource Manager* - служи за управљање ресурсима као што су, на пример, слике;
- *View System* - садржи базу готових графичких приказа и алата (енг. *widget*) ;
- *Location Manager* – омогућава управљање услугама које се темеље на локацији корисника;
- *Notification Manager* - омогућава управљање обавештењима и догађајима (нпр. обавештење о пристиглим порукама, надолазећи састанци итд);

2.1.3.5 Апликативни слој

Апликативни слој је највиши слој у архитектури и чине га корисничке апликације уређаја. Представља слој видљив крајњем кориснику. Укључује неке од стандардних

системских апликација као што су програм за поруке, листа контакта, веб претраживач и многе друге.

2.2 Концепт апликације за *Android*

Све *Android* апликације писане су на програмском језику *Java*. Апликацију чини комплетан код упакован помоћу алата *AAPT* (енг. *Android Asset Packaging Tool*) [32] који као резултат даје фајл са екстензијом *.apk.* У овом формату се апликације дистрибуирају на покретни уређај. Већина креираних апликација припада једној од следећих категорија:

- активности у предњем плану (енг. *foreground activity*) – корисничке апликације без неке специфичне позадинске активности као што су игре или мапе. Када су ван фокуса, апликација се углавном привремено зауставља;
- позадинске услуге (енг. *background service*) – апликације са ограниченом интеракцијом са корисником, које се углавном одвијају у позадини. Апликација за управљање долазним позивима припада овој групацији;
- активност са прекидима – апликације које подразумевају одређени степен интеракције са корисником и које се углавном одвијају у позадини. Једна од таквих апликација је апликација за пуштање *mp3* музике;

За разлику од већине апликација на другим системима, апликације које се покрећу на *Android*-у немају само једну тачку покретања (не постоји само једна `main()` фунцкија). Разлог томе је карактеристика система која заступа идеју међусобне интеракције две или више различитих апликација. Да би то било остварив систем мора бити у могућности да на различите начине покрене процес нове апликације.

2.2.1 Основна структура

Апликацију чине четири основне компоненте:

- активност (енг. *Activity*) - представља компоненту апликације која се углавном може поистоветити са једним конкретним прозором апликације у којем корисник може извршити одређену радњу. Апликација може садржати једну или више активности, при чему је једна од активности дефинисана као примарна активност. Прелази између активности чине један компактни кориснички интерфејс, треба имати на уму да су оне међусобно независне. Свака активност имплементира се као посебна класа која наслеђује класу *Activity*, те је сама одговорна за чување свог стања у животном циклусу апликације;
- намера (енг. *Intent*) - представља намеру за обављањем одређене радње. У суштини, представља поруку којом желимо рећи да смо нешто урадили или желимо нешто урадити. У зависности од намере, апликација или оперативни систем може ослушкивати и реаговати уколико је то потребно;

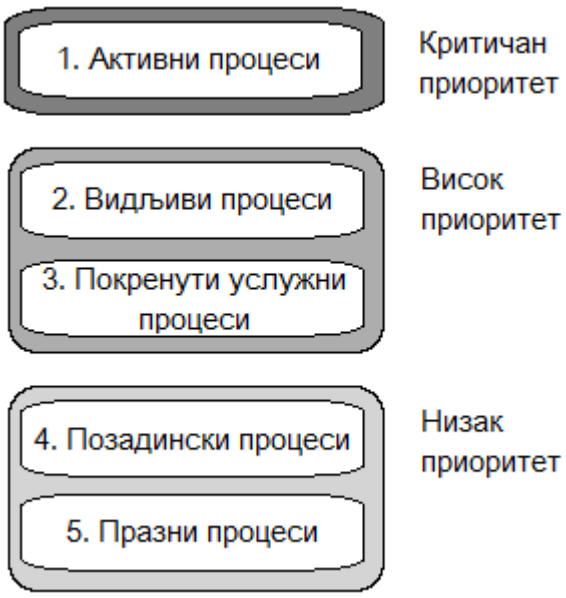
- услуга (енг. *Service*) - представља процес без видљиве корисничке интеракције. Углавном се извршава у позадини током неког предиода времена. Исто тако, користи се када желимо да прикажемо функционалност апликације другим апликацијама;
- испоручилац садржаја (енг. *content provider*) - омогућава међусобно размењивање података између различитих апликација и њихових процеса. Енкапсулира податке и пружа механизме за остваривање безбедности података;

Апликација не мора садржати све споменуте компоненте, а исто тако може садржати и неке друге.

2.2.2 Животни циклус апликације

У случају класичне радне површине оперативног система *Windows* [33] или *Linux*, обично постоји један активан прозор и низ неактивних прозора, а контролу животног циклуса врши корисник. За разлику од тога, *Android* сам брине о животном циклусу апликација, а контрола се врши по систему *LIFO* (енг. *Last In First Out*) [34].

Сваки приказ апликације остварен је помоћу објекта класе *Activity*. Апликацију чини једна или више активности. Врло је важно напоменути да апликација није исто што и процес, што битно утиче на понашање и животни циклус апликације. Како би се очувала ефикасност читавог система, уколико је то потребно, ради ослобађања меморијских ресурса за апликације вишег приоритета (углавном оне које корисник користи у том тренутку), неки процеси и њихове апликације ће бити угашене без упозорења. На следећој слици приказано је стабло приоритета процеса.



Слика 2. Приоритети процеса

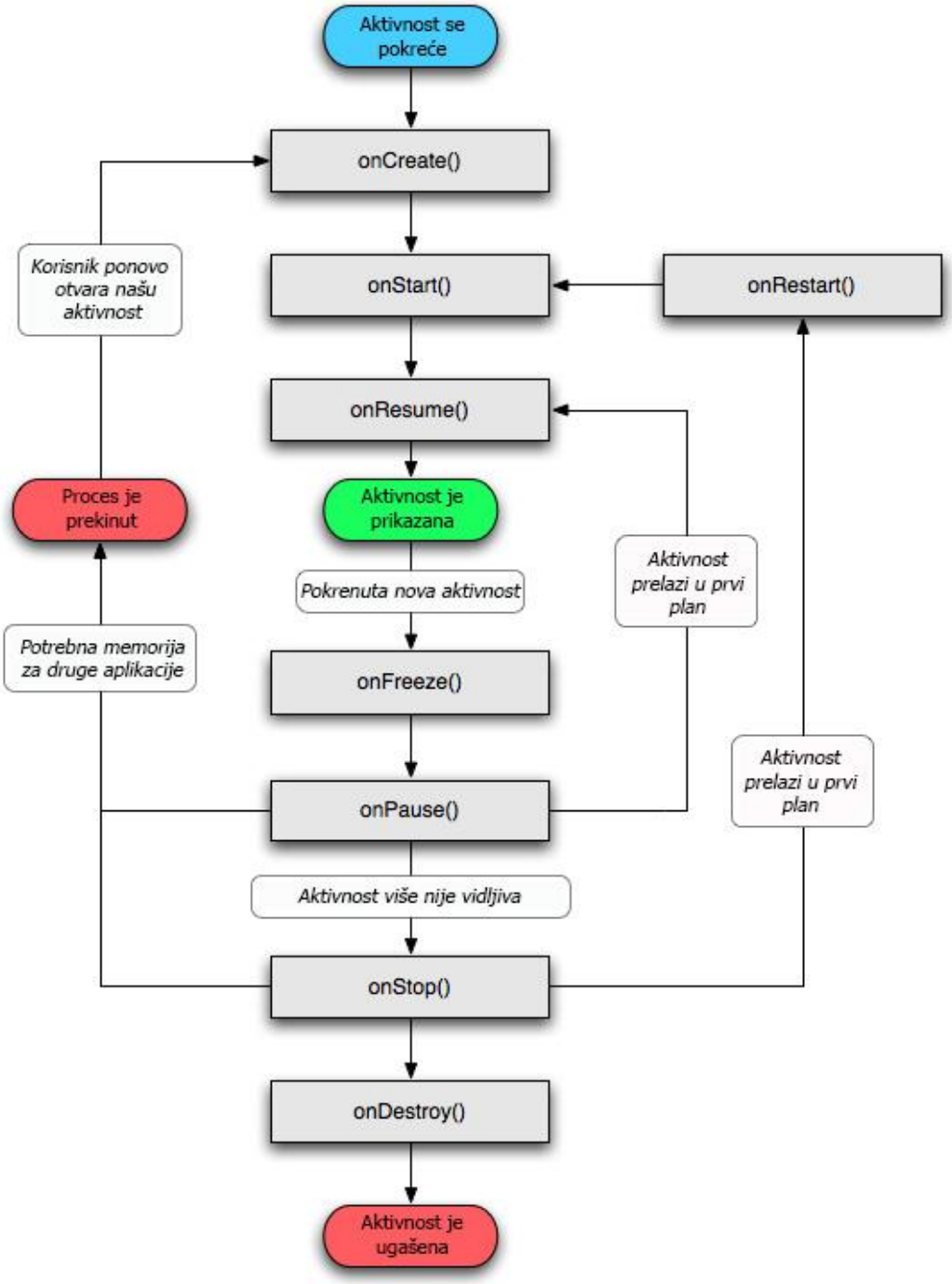
На слици 2. су приказани приоритета процеса. Постоје следеће врсте приоритета:

- активни процеси – оваквих процеса је углавном мало и имају врло високи приоритет из разлога што подржавају апликације с којима корисник управо остварује активну интеракцију;
- видљиви процеси – представљају видљиве, али неактивне процесе. Углавном их је врло мало, те се гасе у крајњој нужди;
- покренути услужни процеси – процеси покренутих услуга са којима корисник нема активну интеракцију, те из тог разлога имају нешто нижи приоритет од активних и видљивих процеса. И даље се сматрају активним, те имају висок приоритет;
- позадински процеси – представљају процесе невидљивих активности без покренутих сервиса. По правилу их има много, те се гасе по принципу "задњи виђен, први прекинут" (*LSFK,* енг. *Last Seen First Killed*);
- празни процеси – представљају запис покренутих процеса у привременој меморији са циљем смањења времена поновног покретања;

На животни циклус апликације утиче искључиво систем, а не сама апликација. Стање у којем се налази апликација одређује њен приоритетни ниво. Често ће се у раду термин апликација поистовећивати са термином "активност". То не би требало да нас збуњује јер је апликација сачињена од једне или више активности и стање активности се може повезати са стањем апликације. Животни ток једне активности приказан је на слици 3.

Прелази између појединих стања невидљиви су крајњем кориснику. Контролисани су од стране система употребом одговарајућих виртуалних метода, тако да свака апликација може да реагује на следеће догађаје:

- `onCreate(Bundle)` – позива се приликом покретања апликације;
- `onStart()` – назначује почетак приказа апликације кориснику;
- `onResume()` – позива се приликом наставка интеракције са корисником;
- `onPause()` – позива се приликом пребацивања у позадински начин рада;
- `onStop()` – позива се у случају дужег преиода некоришћења апликације;
- `onRestart()` – назначује поново позивање апликације из заустављеног стања;
- `onDestroy()` – позива се непосредно пре гашења апликације;
- `onSaveInstanceState(Bundle)` – опциона метода која се позива у случају чувања стања инстанце;
- `onRestoreInstanceState(Bundle)` – позива се приликом реиницијализације активност из стања претходно сачуваног методом *onSaveInstanceState(Bundle)*;



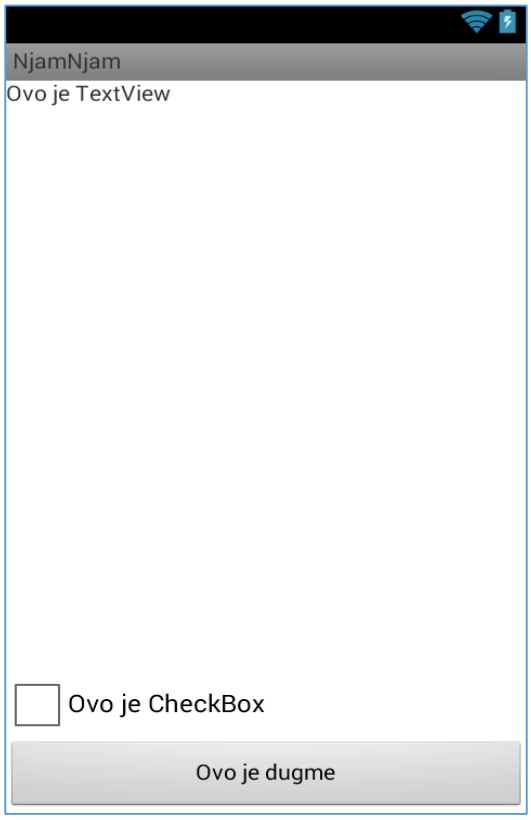
Слика 3. Животни ток једне активности

2.2.3 Кориснички интерфејс

Постоје два начина обликовања корисничког интерфејса: процедурални и декларативни. Процедурални дизајн се односи на писање *Java* кода, а декларативни на описивање интерфејса на *XML*-у. У пракси се за прављење графичког интерфејса углавном користи *XML*. Прављењем интерфејса активности добијају свој графички приказ и на тај начин им је омогућена интеракција са корисником. Основне класе корисничког интерфејса су објекти класа *View* и *ViewGroup.*

- *View* – објекат чија структура у себи носи запис изгледа и садржаја одређеног правоуганог подручја на екрану. Управља исцртавањем елемената и осталим факторима који утичу на изглед дефинисаног дела екрана;
- *ViewGroup* – посебна врста објекта *View* која садржи и управља групом садржаних *View* објеката чиме је омогућено приказивање сложеног корисничког интерфејса;

Исцртавање елемената *xml*-а започиње од корена *xml*-а тако што активност прво позива своју методу `setContentView()` и систем *Android* предаје референцу на корени објекат. Сваки подчвор исцртава се позивањем методе `draw()` и то под условом да чвор сам пошаље захтев за локацијом и величином, а родитељски чвор доноси коначну одлуку о величини простора за исцртавање подчвора и његових локација на екрану. Свака група погледа одговорна је за приказивање својих подчворова. Пример *xml*-а и корисничког интерфејса који настаје на основу њега приказан је на слици 4.




```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout

xmlns:android="http://schemas.android.com/apk/
res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.01"
        android:text="Ovo je TextView" />
    <CheckBox
        android:id="@+id/checkBox1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.00"
        android:text="Ovo je CheckBox" />
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Ovo je dugme" />

</LinearLayout>
```

Слика 4. Изглед једне активности и одговарајући код у позадини

2.2.3.1 Мени

За приступ менију апликације користи се дугме *Menu* које се налази на самом уређају (уколико је хардверски подржано) или се може приказати уколико се одабере нека од компоненти апликација. Структуру менија, која је хијерархијска, није потребно ручно кофигурисати него се то постиже писањем метода `onCreateOptionsMenu()` или

`onCreateContextMenu()` с пописом ставки менија за одређену активност. Управљање догађајима такође се аутоматски одвија позивањем метода `onOptionsItemSelected()` или `onContextItemSelected()` у оквиру активности.

2.2.3.2 Додатне могућности интерфејса

Осим основних компоненти апликација може садржати мноштво додатних елемената и њихових варијација. Неки од тих елемената су контроле, стилови и теме.

Контроле су врста подкласе класе *ViewGroup* и користе се за приказ готових података (за разлику од исцртавања група). Развојни алат нуди готове стилове и теме, али такође је могуће створити и властите. Стил је скуп једног или више атрибута за обликовање посебних елемената у приказу (нпр. дефинисање одређене величине или боје приказа и слова). Тема је скуп једног или више атрибута за обликовање који се примењују на једну или више активности у апликацији (нпр. дефинисање одређених позадниских боја, изгледа текста за одређену активност итд).

Стилови представљају колекцију особина који се односе на изглед и формат погледа или прозора. Могу се дефинисати поља као што су висина, ширина, боја слова, величина слова, боја позадине и остало. Дефинисани су у *XML*-у који описује ресурсе апликације. На овај начин више графичких објеката истог типа могу да имају исти стил. Филозофија стилова је иста као код веб дизајна - омогућавају нам да одвојимо дизајн од садржаја.

2.2.4 Датотека AndroidManifest.xml

Свака апликација обавезно мора садржати датотеку *AndroidManifest.xml.* Ова датотека описује апликацију оперативном систему и другим апликацијама. Налази се у коренском директоријуму пакета и садржи следеће информације:

- назив пакета – служи као јединствени идентификатор апликације
- опис компоненти:
  - активности,
  - услуге
  - испорулилац садржаја (енг: *Content Provider*)
  - филтери намере (енг. *Intent Filters*)
  - пријемник порука (енг: *Broadcast Receiver*)
  - остале;
- декларација дозвола за приступ компонентама апликације од стране других апликација;
- минимална верзија *Android*-а коју захтева апликација;
- попис библиотека које апликација користи;

- попис класа које осигуравају дебаговање и остале информације док је апликација активна - ове декларације су присутне у *AndroidManifest.xml* фајлу приликом развоја и тестирања, те се избацују пре њеног објављивања;

У наставку је приказана структура фајла *AndroidManifest.xml* и свих елемената и атрибута које може садржати.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />

    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>

        <uses-library />
    </application>

</manifest>
```

Од свих елемената, једино су елементи *manifest* и *application* обавезни и могу се појавити само једанпут. Остали значајни елементи су:

- *activity* – свака активност апликације мора имати свој елемент *activity*. Подржава поделемент *intent-filter* којим се специфицирају поједине намере активности као рецимо којој категорији и акцији припада. Остали атрибути могу бити назив активности, орјентација екрана приликом покретања активности, тема, и многи други. Покретање активности која није дефинисана резултираће појавом грешке;
- *service* – дефинише сервисе који се пружају. За разлику од активности, немају графички интерфејс. Обично се извршавају у позадини и сви сервиси морају бити дефинисани овим елементом. Уколико се не дефинишу, систем их неће видети и као такви се не могу покренути;
- *provider* – сви испоручиоци садржаја (енг. *Content Providers*) се описују овим типом елемента. Испоручиоци садржаја који се налазе у другим апликацијама се не наводе, већ само они који се налазе у нашој апликацији;
- *uses-library* – односи се на библиотеку коју апликација користи и која мора бити повезана. Овај елемент говори које библиотеке треба учитати јер се њен код користи у нашим класама. На тај начин се утиче на начин инсталације апликације и њену доступност на *Google Play*-у [35];
- *uses-permission* – део сигурносног модела који садржи сигурносне дозволе додељене од стране корисника, а односе се на функције апликације. Јако битан елемент који утиче и на инсталацију апликације;

Дозволе се дефинишу због хардверског ограничења приступања меморији једног процеса од стране другог, те зато што је свакој апликацији додељен специфични кориснички *ID.* Неке од најчешће коришћених дозвола су:

- *INTERNET* – дозвола за приступање интернету;
- *READ_CONTACTS* – дозвола за читање контакта;
- *WRITE_CONTACTS* – дозвола за уписивање нових и мењање постојећих контаката;
- *RECEIVE_SMS* – прегледање пристиглих СМС порука;
- *ACCESS_COARSE_LOCATOR* – грубо бежично лоцирање преко базне станице;
- *ACCESS_FINE_LOCATION* – фино лоцирање коришћењем GPS-а;

Уређивање фајла *AndroidManifest.xml* се може вршити ручно у текстуалном едитору или коришћењем едитора који долази у оквиру *ADT*-а.

2.3 Развојни алати и библиотеке

За развој *Android* апликација користи се пакет алата за развој софтвера *Android SDK (*енг. *Software Development Kit).* Он пружа подршку за развој и тестирање апликације и проналажење и уклањање грешака. Чине га следеће компоненте:

- ***Android API*** представља скуп протокола и алата за развој *Android* апликација. Обухвата софтверске компоненте и правила како оне комуницирају једне са другима;
- ***ADT*** (енг. *Android Development Tools*) је програмски додатак (енг. *plugin*) за развојно окружење *Eclipse* који омогућује лакше и брже развијање *Android* апликација;
- **Емулатор** оперативног система *Android* служи за симулацију рада мобилног уређаја на рачунару;
- ***DDMS*** (енг. *Dalvik Debug Monitoring Service*) служи за контролу рада *Android* апликације и пружа подршку у проналажењу и уклањању грешака;
- ***AAPT*** (енг. *Android Asset Packaging Tool*) се користи за прављење и дистрибуирање програмског пакета у формату *.apk*;
- ***ADB*** (енг. *Android Debug Bridge*) је клијентско-серверска апликација за инсталирање и покретање *.apk* датотека на емулатору или мобилном уређају и приступ контролама мобилног уређаја. Користи се за проналажење и уклањање грешака на емулатору или мобилном уређају;
- **Детаљна документација**;
- **Примери кода** су једноставни примери за демонстрацију коришћења одређених *API*-ја и могућности које су пружене;

У даљем тексту детаљније ћемо описати функционисање најважнијих пакета *Android SDK*.

2.3.1 Емулатор *Android*-а

Android SDK садржи емулатор мобилних уређаја која омогућава развој и тестирање апликације. Коришћењем *ADT*-а као програмског додатка за *Eclipse*, емулатор се аутоматски позива приликом покретања апликације. Изглед једног од емулатора је приказан на слици 5.

Слика 5. *Android* емулатор

Емулатор је апликација утемељена на емулатору *QEMU* [36] са виртуалним уређајем утемељеним на процесору *ARM* (енг. *Advanced RISC Machine*). *QEMU* је машински емулатор заснован на отвореном коду. Емулатор у потпуности симулира понашање *Android* уређаја. Омогућено је коришћење системских апликација (нпр. бирач бројева), одговарајуће графичко окружење, као и многе наредбе које се могу задати коришћењем одговарајућих дугмића, слично као код правих уређаја. Једино ограничење емулатора је немогућност да успостави долазне и одлазне телефонске позиве.

2.3.2 DDMS

DDMS је алат који омогућава управљање процесима на емулатору или покретном уређају и помаже приликом проналажења и уклањања грешака у програмском коду. Пружа различите услуге као што су употреба *LogCat*-а (компоненте која описује догађаје који су се десили и исписује их у читљивом облику), праћење информације о процесима, снимање активног стања уређаја, контролу *SMS* порука [37], податке о локацији и многе друге елементе окружења. Овај алат се понаша као веза између развојног окружења и апликације која се извршава на уређају. Као што је поменуто, свака апликација се покреће као засебни процес који се изводи као посебна инстанца виртуалне машине. Сваки од тих процеса је независно повезан са програмом за проналажење и уклањање грешака.

2.3.3 Android Debug Bridge (ADB)

ADB је клијентско-серверски алат који служи за комуникацију са *Android* емулатором или правим уређајем преко командне линије. На тај начин омогућава тестирање програма. Укључује три компоненте:

- клијент – апликација која се извршава у развојном окружењу и комуницира са позадинским процесом *ADB*-а преко сервера;
- сервер – позадински процес који се покреће на рачунару на којем развијамо апликацију и управља комуникацијом између клијената и позадинског процеса *ADB*-а;
- позадински процес *ADB*-а на емулатору;

Приликом покретања *ADB*-овог клијента прво се проверава постоји ли активан *ADB*-ов сервис. Уколико не постоји, покреће се и отвара локални *TCP* порт 5037 на који он ослушкује захтеве послате од стране *ADB* клијента. Сви *ADB* клијенти користе порт 5037 за комуникацију са сервером. Након тога, успоставља се комуникација сервера и свих инстанци емулатора или уређаја. То се остварује тако што сервер ослушкује све непарне *TCP* портове у распону од 5555 до 5585. Уколико *ADB* сервер приликом претраживања наиђе на позадински процес *ADB*-а, успоставља се комуникација на том порту. Претраживање само непарних портова врши се из разлога што је успостављање везе потребан један пар портова и то један за везу са емулатором и један за везу са конзолом. Тако се, на пример, конзоли емулатора повезаног са *ADB* сервером на порту 5555 приступа преко порта 5554. Након успостављања везе могуће је контролисати емулатор различитим *ADB* наредбама.

2.3.4 Google API

Google API је уско повезан са *Android SDK*-ом и садржи библиотеке које дозвољавају повезивање *Android* апликација и постојећих *Google*-ових сервиса. Уколико развијамо *Android* апликацију и желимо да омогућимо нашем кориснику да користи *Google*-ове сервисе кроз нашу апликацију, то морамо урадити кроз *Google API*. API је испоручен у фајлу *android.jar* и смештен је у пакету *com.google.**. Постоји релативно мали број пакета у *Google API*-ју. Неки од њих су они који се тичу графике, контаката, календара итд. Нама најинтересантнији пакет је сигурно онај који је се односи на *Google* мапе, пошто ће бити коришћен у изради наше апликације.

2.3.4.1 *Google maps Android API*

Google Maps Android API омогућава коришћење података о мапи и локацији. Користи *GPS* послужилац [37] за мобилне телефоне за проналажење тренутне локације, а као алтернативу користи бежичну мрежу и телефонске репетиторе. Тренутна локација се одређује помоћу телефонског репетитора тако што се анализирају расположиве базне

станице коришћењем мреже мобилног оператера. Триангулацијом различитих јачина сигнала добијених од различитих репетитора и упоређивањем са њиховим локацијама, одређује се тренутна локација. Уколико проналазимо локацију коришћењем бежичних мрежа, препознају се околне базне станице које уређај детектује. Коришћењем њихових локација (добијених са интернета, из базе бежичних мрежа) и одређивањем јачине сигнала , одређује се тренутна локација корисника.

Приоритет начина одређивања локације је следећи:

- *GPS* сервис;
- сервис бежичне мреже;
- преко репетитора;

Мапе и остале информације везане за њих нису иницијално записане на уређају када се инсталира *Google Maps* апликација за *Android*. Неопходна је интернет веза. Мапе се након учитавања привремено кеширају тако да се смањује количина података потребна за преузимање са интернета при наредној употреби.

Да бисмо користили *Google Maps API*, морамо да додамо *Maps API* кључ. Кључ је бесплатан и може се користити у било којој апликацији која позива *Maps API*. Подржава неограничен број корисника. Кључ се добија кроз конзолу за *Google API* тако што се приложи сертификат апликације [39] и њен назив (тј. назив њеног пакета). Кључ се додаје у фајл *AndroidManifest.xml* на следећи начин:

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AImuSyCVgLcR1avzv*WxaysxF7n3B"/>
<uses-library android:name="com.google.android.maps" />
```

Да би мапе радиле на корисниковом уређају, потребно је да постоји активна интернет веза. Поред тога, да би подаци били тачнији и прецизнији, потребно је да буду активни бежична веза и *GPS* навигациони систем уређаја. Уколико аутор *Android* апликације жели да приступи овим сервисима уређаја, он их мора захтевати кроз фајл *AndroidManifest.xml*. Поменуте дозволе се захтевају на следећи начин:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
```

2.3.5 Управљање подацима

Android за управљање подацима користи систем за управљање базама података *SQLite* [40]. *SQLite* је имлементиран као релативно мала *C* програмска библиотека. Изворни код за *SQLite* је у јавном власништву. База користи јако мало меморије

приликом извршавања (око 250 килобајта) и самим тим је одличан кандидат за коришћење у *Android* окружењу. *SQLite* подржава типове података као што су *TEXT, INTEGER* и *REAL*. Остали типови се преводе у један од ових типова пре него што се сачувају у бази. *SQLite* не проверава да ли вредност уписана у одређену колону одговара типу колоне. Тако је могуће уписати цели број у ниску и обрнуто.

SQLite је интегрисан у сваки *Android* уређај. Коришћење базе података *SQLite* у *Android*-у не захтева никакву инсталацију нити административна права. Једино је неопходно дефинисати упите за прављење и одржавање базе. Након тога *Android* платформа аутоматски одржава базу. Приступ бази је исти као да се приступа фајл систему. Ово понекад зна бити споро. Уколико наша апликације прави базу, она ће бити сачувана у директоријуму *DATA/data/APP_NAME/databases/FILENAME*.

- *DATA* – представља фолдер у коме се чувају подаци;
- *APP_NAME* – назив наше апликације;
- *FILENAME* – име базе које смо изабрали у нашој апликацији;

За рад са базом података мора постојати класа која наслеђује апстрактну класу `SQLiteOpenHelper` и имплементира њене апстрактне методе `OnCreate()` и `OnUpgrade()`. У оквиру класе у сваком тренутку можемо добити инстанцу класе позивом методе `this.getReadableDatabase()`. Правимо табеле тако што позивамо *SQL* команде за прављења табела у нашој бази. Пример прављења једне табеле приказан је у наставку:

```
private static String DB_TBL_KORISNIK = "CREATE TABLE korisnik ("
    + " _id integer primary key, username text not null, "
    + " password text not null, naziv text null, adresa text null, "
    + " grad text null, telefon text null);";

@Override
public void onCreate(SQLiteDatabase db) {
   try {

         db.execSQL(DB_TBL_KORISNIK);

   } catch (SQLiteException e) {
         Log.w("greska pri pravljenju", e.toString());
   }
}
```

2.3.5.1 JSON

JSON (енг. *JavaScript Object Notation*) је текстуално заснован отворени стандард записивања података који служи за размену података. Нотација *JSON* је изведена из синтаксе језика *Javascript* за представљање једноставних структура. Упркос вези са *Javascript*-ом, он је језички независан и користи се у многом програмским језицима. Формат *JSON* је првобитно направио Даглас Крокфорд [41].

Формат *JSON* се често користи за серијализацију и преношење структурираних података преко мреже. Пре свега се користи за размену података између сервера и веб

апликације, као замена за *XML* (енг. *Extensible Markup Language*) [42]. Типови података могу бити:

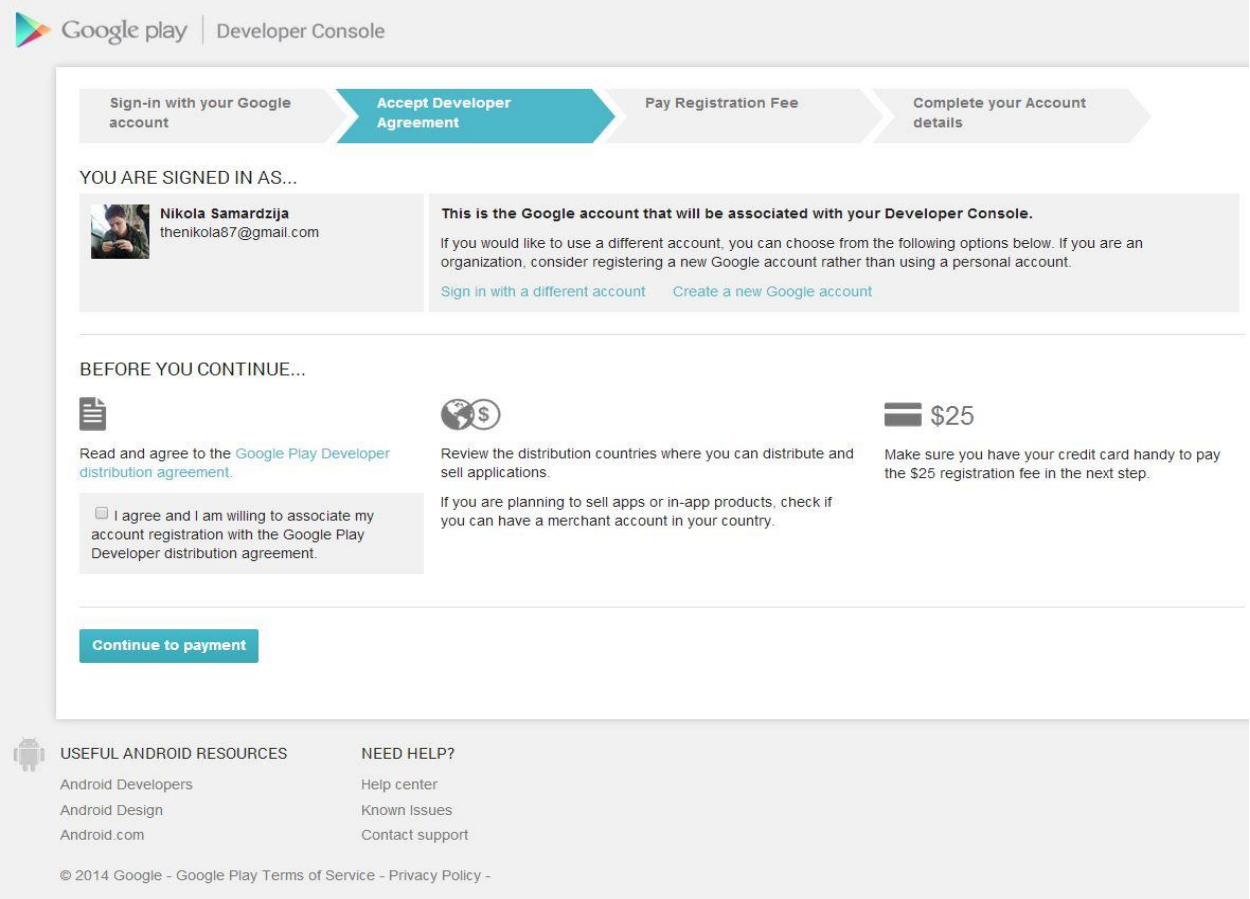
- Број – број у покретном зарезу са двоструком прецизношћу, зависи од имплементације;
- Ниска – *Unicode* формат, са двоструким наводницима, као специјани симбол се користи *backslash*;
- Логичка вредност *Boolean* – може бити *true* или *false*;
- Низ – уређена секвенца вредности, одвојена зарезима и уоквирена средњим заградама. Вредности не морају бити истог типа;
- Објекат – неуређена колекција парова *кључ:вредност* где карактер ':' раздваја кључ од вредности. Парови су раздвојени зарезима и уоквирени витичастим заградама. Кључеви морају бити ниске и не смеју се понављати;
- *null* (празно);

Пример формата *JSON* ће бити наведен у поглављу 4.1. У апликацији за наручивање хране *JSON* смо користили за размену података са сервисима, прикупљање података са сервера, али и слање података о наруџбини или промењеном профилу.

2.4 Регистровање програмера и апликације

Након развоја апликације за оперативни систем *Android*, потребно је нову апликацију поставити на официјални портал *Google Play Store*, на коме су смештене све *Android* апликације. Процес регистровања се састоји из две фазе: регистровање *Android* програмера и регистровање апликације.

Регистровање програмера се врши преко званичне стране за регистрацију https://play.google.com/apps/publish/signup/. Пре саме регистрације потребно је прочитати правила коришћења и сагласити се са њима. Процес регистрације се врши преко *Google* налога и процес је приказан на слици 6.



Слика 6. Приказ форме за регистрацију програмера

Приликом регистрације неопходно је уплатити износ од 25 долара. Процес регистрације се завршава уносом датеља везаних за налог укључујући и назив програмера који ће бити приказан на *Google Play Store*-у. Потребно је сачекати око до 48 сати како би се процес регистрације програмера завршио. Након тога омогућена је употреба *Google Play* конзоле. Помоћу конзоле је омогућена промена профила, постављање апликације коју смо развијали, постављање цене наплате производа и прегледање извештаја везаних за апликацију.

Пре самог регистровања апликације, неопходно је извршити одређене припреме које се огледају у следећим корацима:

- Конфигурисање апликације – потребно је уклонити таг `android:debuggable` из `AndroidManifest.xml` фајла. Потребно је поставити верзију и назив верзије апликације, тј поставити атрибуте `android:versionCode` и `android:versionName` који се налазе у елементу `<manifest>`;
- Производња и постављање верзије објављивања апликације - *Android Development Tools (ADT)* додатак и *Ant* скрипта за изградњу који долазе са *Android SDK* поседује све што је потребно за изградњу и постављање верзије апликације;
- Тестирање апликације – пре регистровања апликације неопходно је тестирати апликацију. Препоруке су да се апликација тестира на што више модела, различитих екрана, како мобилних телефона тако и таблет уређаја;

- Ажурирање ресурса апликације – пре регистровања, неопходно је проверити да ли су сви ресурси које користимо ажурирани. Ту се пре свега мисли на мултимедијалне фајлове и остале графичке елементе;
- Припремање удаљених сервера и сервиса на које се ослања наша апликација – уколико наша апликација зависи од спољних сервиса и сервера, морамо бити сигурни да су они такође припремљени у тренутку регистровања апликације.

Након ових провера *Google Play* захтева додатна подешавања. Потребно је оценити садржај ваше апликације која говори о зрелости апликације коју региструјемо. Зрелост апликације може бити ниска, средња, висока и апликација за све [43]. Затим, одређујемо у којим земљама ће се вршити дистрибуција апликације. Да ли ће бити доступна за све или ћемо се ограничити на изабране земље.

Величина наше апликације може одредити како ће се она регистровати. Максимална величина *apk* фајла не сме прелазити 50 мегабајта. Уколико је наша апликација већа од тога, могу се користити додатни *apk* фајлови који ће се такође налазити на *Google Play*-у и аутоматски скидати заједно са нашом апликацијом. Наравно, уколико нам је ово потребно, неопходно је извршити измене у начину изградње нашег *apk* фајла.

Поред претходно наведених, могуће је још поставити следећа подешавања:

- Одлучити се да ли ће апликација бити бесплатна или ће се наплаћивати;
- Одлучити да ли ће се садржај који апликација пружа продавати;
- Поставити цене за проиводе које апликација пружа у различитим валутама;
- Локализовати апликацију према држави у којој се корисник налази, тј. Превести апликацију на локални језик;
- Поставити опис и слике апликације и на тај начин промовисати апликацију;
- Остала подешавања.

Након свих ових поставки потребно је још једном проверити сва подешавања користећи конзолу *Google Play*. Уколико је све у реду, коначно можемо регистровати апликацију.

3. Анализа и пројектовање

У овом поглављу биће објашњене почетне фазе израде апликације за наручивање хране, које обухватају приказ методологије рада, анализу проблема, случајеве употребе и модел података.

3.1 Мотивација

Услед недостатка времена или услова да се спреми храна, данас је све присутнија потреба корисника да наручује готову храну. Корисник жели могућност избора хране према врсти, тј. категорији, али исто тако и према ресторану који производи храну. Занима га какве су цене хране и које су њене карактеристике. Исто тако, веома је важно знати које је време чекања да се храна достави. Циљ нам је да направимо апликацију која ће испуњавати ове захтеве.

3.2 Процес развоја софтвера

Израду апликације можемо поделити на два дела: први се односи на функционалност, а други на кориснички интерфејс. Које све опције апликација треба да садржи? Свакако, као ентитети се издвајају ресторани и храна. Они морају бити категоризовани због брже и боље претраге. У претрагу ресторана укључује се и мапа и на тај начин пружа се подршка у одабиру ресторана који се налазе у непосредној близини корисника апликације. Сваки ресторан садржи одређени мени, односно списак хране и њене карактеристике. С друге стране, потребно је дефинисати корисника који наручује храну. Он би требало да садржи одређене информације. Једна од њих би била адреса, како би достављач знао где треба да се испоручи храна. Уношењем информација о кориснику избегавамо злоупотребу система и повећавамо поузданост апликације, што је свакако значајно достављачу, односно произвођачу хране. Када смо дефинисали ова два субјекта, произвођача и потрошача, морамо их некако повезати. Пошто се комуникација одвија на удаљеним локацијама, то ћемо учинити коришћењем веб сервиса.

Кориснички интерфејс би требало да буде што једноставнији за коришћење. Као и код осталих *Android* апликација, ограничени смо величином и карактеристикама екрана. Посебну пажњу треба обратити на то да су телефони различитих резолуција и свака од њих би требало да буде подржана.

3.2.1 Процес прегледања и наручивања хране

Корисник покреће апликацију. Апликација ће радити уколико постоји веза ка интернету. Корисник се логује на систем. Уколико нема налог, уноси своје податке и региструје се. Када је корисник пријављен, отвара се почетна страна апликације. На

почетној страни биће приказане опције за избор ресторана из листе ресторана или преко мапе ресторана. Уколико жели да наручи храну, корисник бира жељени ресторан. Након одабира ресторана, отвара се профил ресторана који садржи јеловник са храном и детаљне информације о самом ресторану. У јеловнику је приказана храна, списак састојака и цена. Корисник бира храну, одређује количину и изабрана храна се додаје у корпу. Да би извршио наруџбину, одлази до виртуалне корпе. Ту се налази изабрана храна и укупна сума. Сам садржај корпе се може мењати. Уколико је корисник задовољан изабраном храном, апликација му нуди да се достава изврши на адресу коју је унео, тј. која се налази на његовом профилу. Уколико не жели да се храна достави на ту адресу, постоји опција да ручно унесе адресу или да се изабере његова тренутна локација. У следећем кораку храна се наручује.

Слика 7. Дијаграм активности апликације за наручивање хране

Наручена храна се смешта на удаљеној бази која се налази на серверу. Даљи ток прослеђивања наруџбине до ресторана није тема овог рада и као такав, није имплементиран. Одржавање сервера врши администратор користећи веб портал. Његов задатак је да додаје нове ресторане и храну у тим ресторанима.

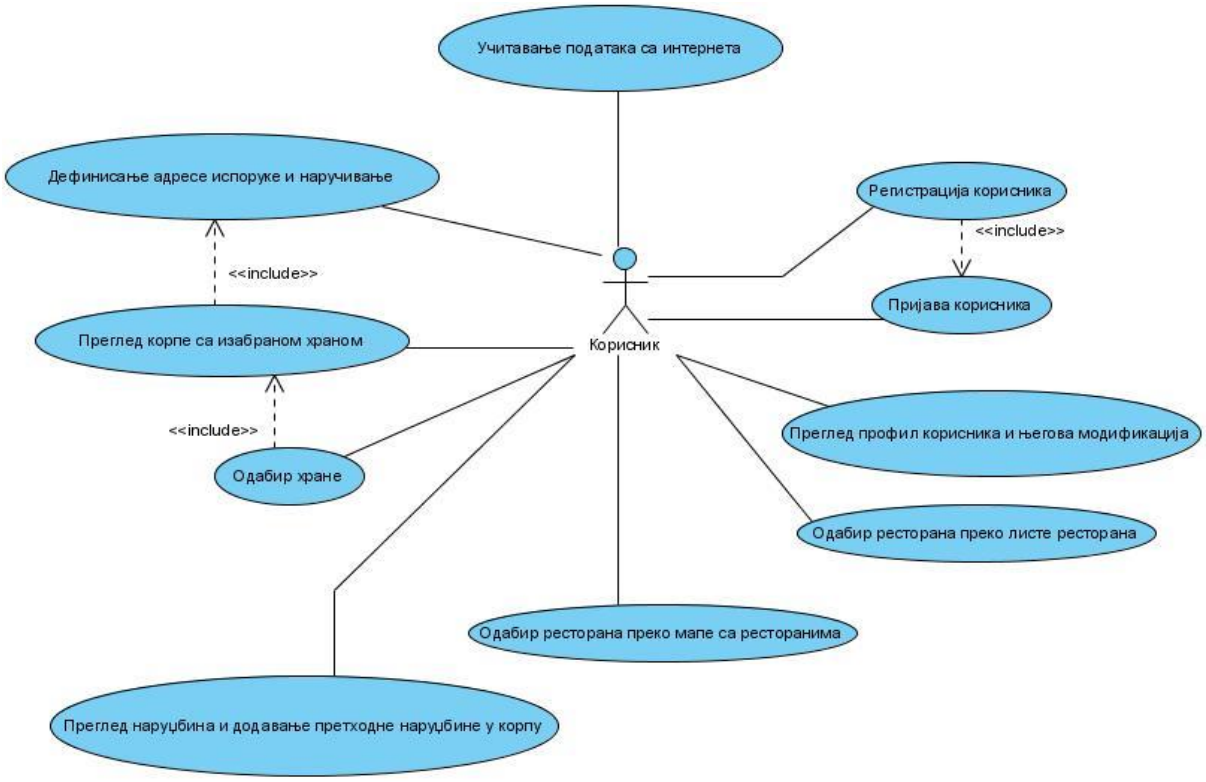
3.2.2 Дијаграм активности

Дијаграм активности представља један од *UML* дијаграма [44] и служи за описивање динамичких аспеката система. Дијаграм активности описује дијаграм тока од једне активности до друге. Активност се може описати као једна операција система.

На слици 7. приказан је дијаграм активности апликације за наручивање хране. Детаљно је представљен целокупан процес наручивања хране од пријаве и регистрације корисника, одабира ресторана и хране до наручивања. Приказан процес биће детаљно описан у случајевима употребе.

3.2.3 Случајеви употребе

На основу дијаграма активности описаног у поглављу 3.2.2 и слике 7. можемо дефинисати случајеве употребе. Случајеви употребе (енг. *Use case*) [45] су спецификације скупа акција које врши систем, које производе видљив резултат који је, по правилу, од вредности за једног или више учесника у систему. Користе се да се прецизира понашање система, без откривања његове унутрашње структуре. Целину апликације за наручивање хране описаћемо случајевима употребе. Графички приказ дат је на слици 8.



Слика 8. Случајеви употребе апликације за наручивање хране

1. Учитавање података са интернета

Опис: При покретању апликације потребно је учитати податке који се налазе на удаљеној локацији. Учитавају се најновији подаци о ресторанима и храни.

Актери: Корисник
Предуслови: постоји интернет веза
Постуслови: подаци су учитани

Главни ток:

1. Корисник покреће апликацију
2. Проверава се да ли постоји интернет веза
   2.1 Уколико постоји интернет веза прелази се на корак 3
3. Преузима се верзија базе на серверу и проверава се локалном верзијом базе
   3.1 Верзије се разликују, прелази се на корак 4
   3.2 Верзије су исте, имамо најновију базу, прелазимо на корак 7
4. Преузимају се подаци са удаљене базе помоћу веб сервиса
5. Уколико постоје подаци у локалној бази, они се бришу
6. Нови подаци се уписују у локалну базу, заједно са новом верзијом базе
7. Подаци су ажурни

Алтернативни токови:

2.2 Не постоји интернет веза, корисник се обавештава о томе и захтева се да је укључи
   2.2.1 Захтева се да корисник укључује интернет везу на свом уређају, уколико то уради прелази се на корак 3
   2.2.2 Корисник затвара апликацију

Слика 9. Кориснички интерфејс случаја употребе „Учитавање података са интернета“

Кориснички интерфејс: састоји се од иконице наше апликације и контроле за учитавање која је активна док год се у позадини учитавају подаци. Приказан је на слици 9.

2. Регистрација корисника

Опис: Уколико корисник није претходно регистрован, неопходно је извршити регистрацију. Подаци о кориснику су неопходни због доставе хране, али и због спречавања злоупотребе апликације.

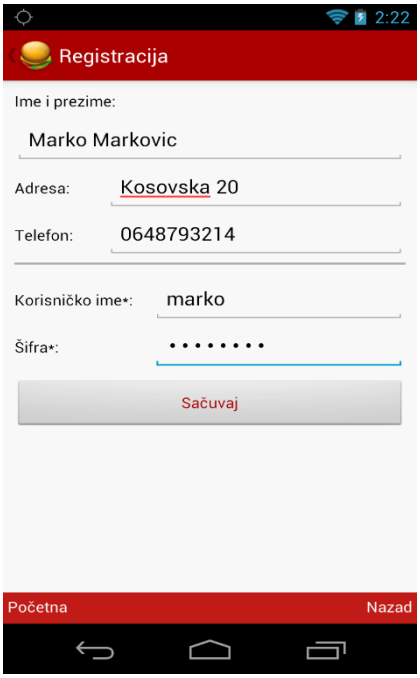
Актери: Корисник
Предуслови: постоји интернет конекције, корисник нема налог
Постуслови: корисник је регистрован

Главни ток:

1. Уколико корисник није регистрован, отвара се форма за регистрацију
2. Корисник уноси неопходне податке: име, презиме, адресу, број телефона, корисничко име и лозинку
3. Корисник покреће акцију регистрације
   3.1 Нису унети сви неопходни улазни подаци, корисник се враћа на корак 2
   3.2 Корисничко име постоји
      3.2.1 Корисник уноси ново корисничко име и прелази се на корак 3
   3.3 Подаци су исправни, прелази се на корак 4
4. Подаци се чувају на удаљеној локацији, тј. серверу и на локалној бази података
5. Корисник је регистрован

Алтернативни токови:

2.1 Корисник прекида унос података и напушта апликацију



Слика 10. Кориснички интерфејс случаја употребе „Регистрација корисника“

Кориснички интерфејс: састоји се од поља за унос података која обухватају име и презиме, адресу, телефон, корисничко име и лозинку. Лозинка се након сваког уноса карактера сакрива. Да би регистрација била успешна морају бити унети сви подаци и корисничко име мора бити јединствено. Корисничи интерфејс приказан је на слици 10.

3. Пријава корисника

Опис: Уколико је корисник претходно регистрован, коришћењем свог корисничког имена и лозинке пријављује се на систем.

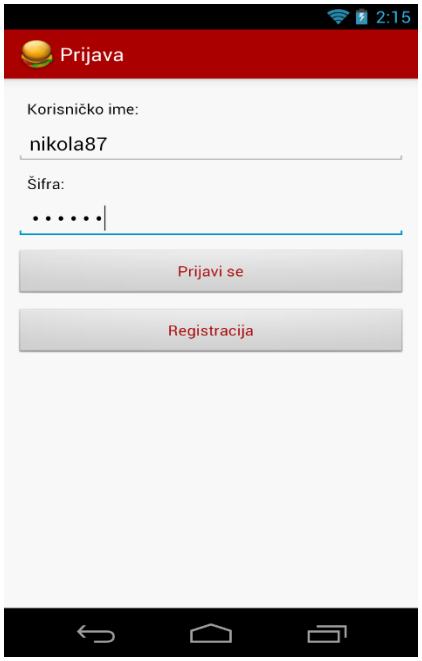
Актери: Корисник
Предуслови: постоји интернет конекције, корисник има кориснички налог
Постуслови: корисник је пријављен на систем

Главни ток:

1. Уколико корисник није пријављен на систем, отвара се форма за пријаву
2. Корисник уноси корисничко име и лозинку
3. Корисник покушава да се пријави на систем притиском на дугме Пријави се
   3.1 Корисник није унео податке, прелази се на корак 2
   3.2 Унети подаци не постоје у систему, прелази се на корак 2
   3.3 Подаци су исправни, прелази се на корак 4
4. Преузимају се кориснички подаци
5. Преузимају се подаци о претходним наруџбинама
6. Корисник је пријављен на систем

Алтернативни токови:

1.1 Корисник је претходно пријављен на систем, прелази се на корак 6.



Слика 11. Кориснички интерфејс случаја употребе „Пријава корисника“

Кориснички интерфејс: састоји се од корисничког имена и шифре. Да би пријава на систем била успешна унети пар корисничко име и шифра мора постојати у систему, тј. морају бити претходно регистровани. Кориснички интерфејс је приказан на слици 11.

4. Преглед профила корисника и његова модификација

Опис: Уколико корисник жели да промени неки од унетих личних података то може да учини на свом профилу. Потребно је да промени податке које жели и сачува измене. Измењени подаци се шаљу веб сервису који чува измене у базу.

Актери: Корисник
Предуслови: постоји интернет конекције, корисник је пријављен на систем
Постуслови: корисник је изменио податке

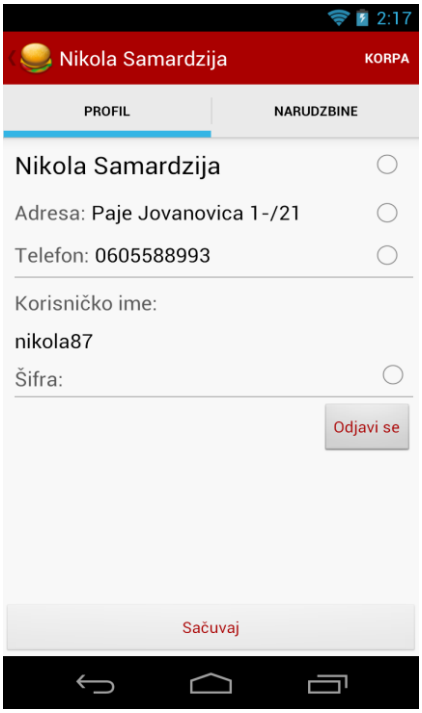
Главни ток:

1. Корисник отвара свој профил
2. Корисник бира податке које жели да измени
3. Корисник мења податке
   3.1 Уколико корисник жели да промени шифру, мора унети прво претходну па онда нову
4. Корисник чува измене
5. Подаци су сачувани

Алтернативни токови:

4.1 Корисник не чува измене, подаци неће бити сачувани

Кориснички интерфејс: активност се састоји од два табулатора. На једном од њих приказан је профил корисника. Притиском на кружић поред поља које желимо да изменимо, отвара се форма за унос нове вредности. На крају је потребно нове вредности сачувати притиском на дугме Сачувај. Интерфејс је приказан на слици 12.



Слика 12. Кориснички интерфејс случаја употребе „Преглед профила корисника и његова модификација“

5. Преглед претходних наруџбина и додавање претходне наруџбине у корпу

Опис: Након отварања профила корисника, имамо опцију прегледа претходних наруџбина. Том приликом можемо видети храну, датум наручивања, количину и износ сваке наруџбине. Постоји могућност одабира исте наруџбине и додавање у корпу.

Актери: Корисник
Предуслови: корисник је пријављен на систем
Постуслови: -

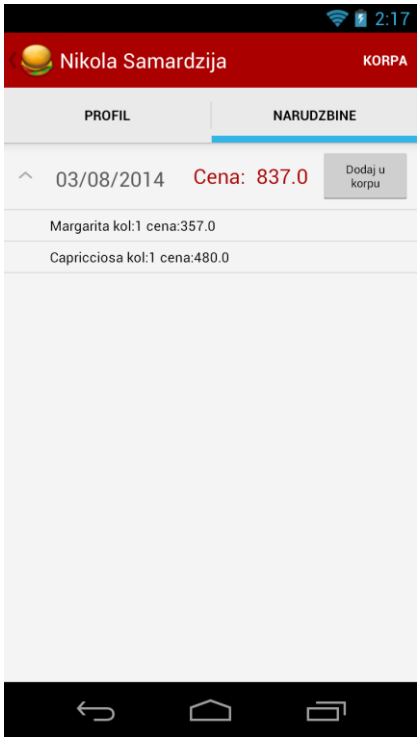
Главни ток:

1. Корисник отвара свој профил
2. Корисник прегледа наруџбине
3. Корисник жели да понови наруџбину
4. Корисник је додао наруџбину у корпу
5. Корисник је напустио преглед претходних наруџбина

Алтернативни ток:

3.1 Корисник не жели да поручи претходну наруџбину, прелази се на корак 5

Кориснички интерфејс: састоји се од листе претходних наруџбина који се могу прегледати. Притиском на дугме Додај у корпу, наруџбина се додаје у корпу. Интерфејс је приказан на слици 13.



Слика 13. Кориснички интерфејс случаја употребе „Преглед претходних наруџбина и додавање наруџбине у корпу“

6. Одабир ресторана преко листе ресторана

Опис: У процесу наручивања хране, прво морамо да одаберемо ресторан. Ресторан можемо да изаберемо куцањем његовог имена у филтеру свих ресторана или да изаберемо према категорији којем одређени ресторан припада. Трећи начин је, наравно, ручним претраживањем списка ресторана.

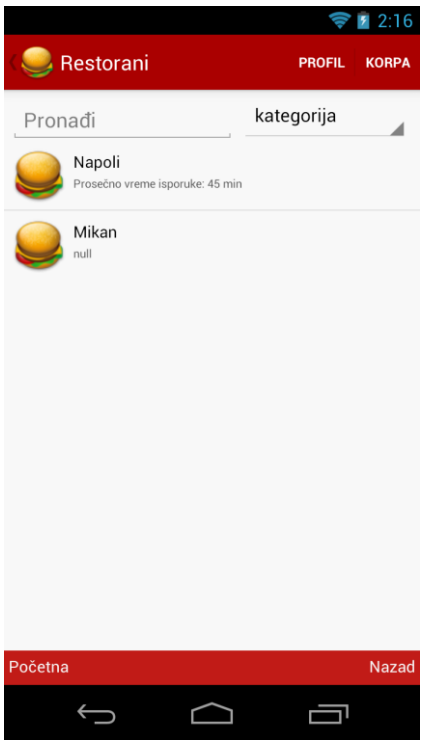
Актери: Корисник
Предуслови: претходно учитани ресторани
Постуслови: корисник је одабрао ресторан

Главни ток:

6. Корисник отвара приказ за избор ресторана
7. Корисник бира ресторан
   7.1 Корисник бира ресторан уносом назива ресторана
   7.2 Корисник бира ресторан према његовој категорији
   7.3 Корисник бира ресторан прегледом листе ресторана
8. Корисник је одабрао ресторан

Алтернативни токови: -

Кориснички интерфејс: приказана је листа ресторана. Сваки елемент листе садржи назив ресторана и задати опис ресторана. Могуће је претражити ресторане уносом текста у текстуално поље или изабрати категорију из падајућег менија. Интерфејс је приказан на слици 14.



Слика 14. Кориснички интерфејс случаја употребе „Одабир ресторана преко листе ресторана“

7. Одабир ресторана преко мапе са ресторанима

Опис: Постоји још један начин да дођемо до хране, а то је преко мапе ресторана. На мапи ће бити исцртани сви ресторани и позиционирани према адреси на којој се они налазе. Корисникова локација ће бити означена, тако да ће знати који су му ресторани најближи што такође може бити значајно због времена испоруке.

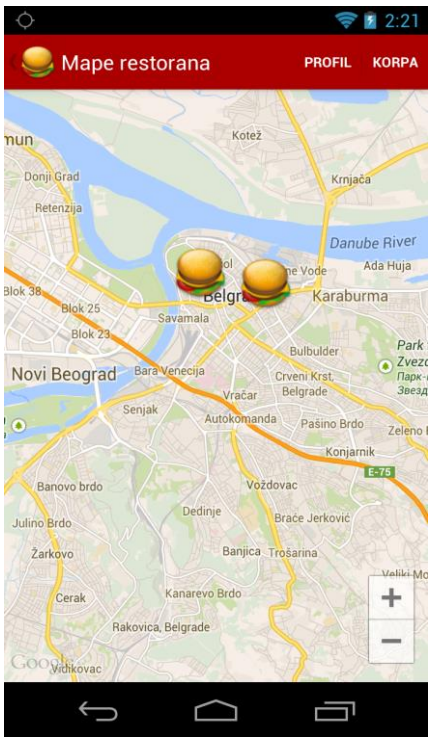
Актери: Корисник
Предуслови: претходно учитани ресторани, подршка за *GPS* локацију корисника
Постуслови: корисник је одабрао ресторан

Главни ток:

1. Корисник отвара мапе на коме су исцртани ресторани на основу њихових адреса
2. Корисник претражује мапу са ресторанима
3. Корисник проверава опис ресторана
4. Корисник је одабрао ресторан

Алтернативни токови: -

Кориснички интерфејс: састоји се од *Google* мапе на којој су приказане локације ресторана и тренутна локација корисника. Притиском на ресторан отвара се могућност приказа профила ресторана. Интерфејс је приказан на слици 15.



Слика 15. Кориснички интерфејс случаја употребе „Одабир ресторана преко мапе са ресторанима“

8. Одабир хране

Опис: Када је корисник одабрао ресторан из којег жели да наручи храну, потребно је види и мени изабраног ресторана. У листи ће бити приказана све ставке са менија тог ресторана и простим прегледом, корисник ће бити у могућности да одабере храну. Када је храна одабрана, корисник може изабрати количину хране и да је дода у корпу.

Актери: Корисник
Предуслови: претходно учитана храна, одабран ресторан
Постуслови: корисник је одабрао храну и додао је у корпу

Главни ток:

1. Корисник отвара ресторан у коме се налази храна
2. Корисник бира храну према њеном опису, састојцима и цени
3. Корисник додаје храну у корпу
   3.1 Корисник бира количину хране коју је одабрао

3.2 Корисник жели да дода још хране из истог ресторана, иде се на корак 2.

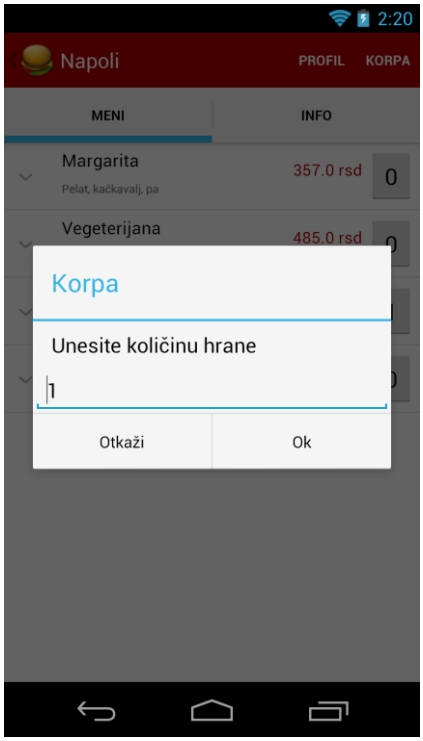
Алтернативни токови:

3.3 Корисник се је погрешио количину хране

3.3.1 Корисник мења количину уносом нове количине

3.3.2 Корисник избацује храну из корпе уносом количине нула

Кориснички интерфејс: на профилу ресторана налази се мени који садржи листу хране који тај ресторан има у понуди. Детаљне информације о храни могу се добити притиском стрелице надоле која се налази поред назива хране. Са десне стране сваке ставке се налази дугме које приказује количину изабране хране. Притиском на дугме, можемо мењати количину. Интерфејс је приказан на слици 16.



Слика 16. Кориснички интерфејс случаја употребе „Одабир хране“

9. Преглед корпе са изабраном храном

Опис: Уколико корисник жели да поручи храну, претходно може да прегледа још једном списак хране која се налази у корпи. Тада је могуће изменити или потврдити садржај корпе

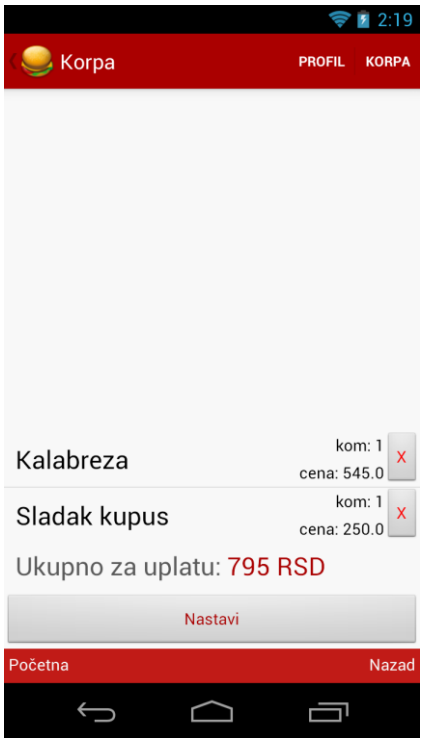
Актери: Корисник
Предуслови: храна је додата у корпу
Постуслови: корисник потврђује садржај корпе

Главни ток:

1. Корисник отвара корпу
2. Корисник прегледа садржај корпе
   2.1 Уколико се корисник не слаже са храном коју је изабрао, може да је избаци из корпе
      2.1.1 Уколико корисник жели да дода другу храну, прелази на случај употребе број 7 – одабир хране.
3. Корисник је сагласан са садржајем корпе

Алтернативни токови: -

Кориснички интерфејс: састоји се од листе изабране хране, укупне вредности и дугмета Настави, где прелазимо у активност за избор адресе доставе. Свака ставка садржи назив хране, изабрану количину, појединачну цену, као и дугме помоћу којег избацујемо храну из корпе.



Слика 17. Кориснички интерфејс случаја употребе „Преглед корпе са изабраном храном“

10.Дефинисање адресе испоруке и наручивање

Опис: Када се корисник одлучи за наруџбину изабраних производа, потребно је да унесе адресу испоруке. Он то може урадити тако што ће ручно унети адресу, изабрати адресу из његовог профила или се одлучити за тренутну локацију на коју се налази.

Актери: Корисник
Предуслови: у корпи се налази храна, интернет конекција

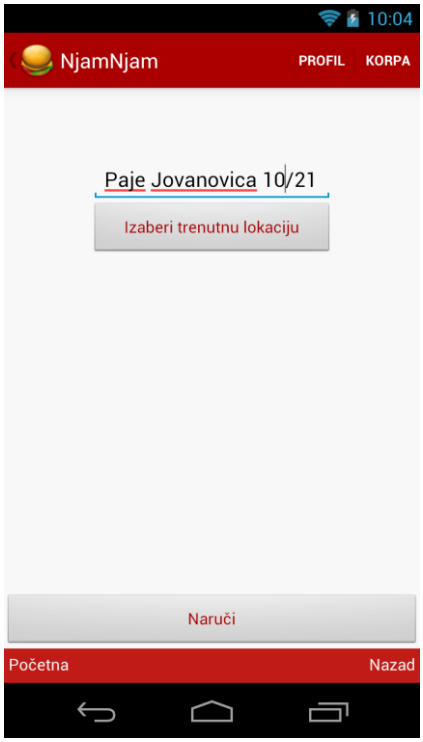
Постуслови: корисник је наручио храну

Главни ток:

1. Корисник отвара форму за унос адресе на којој ће храна бити испоручена
2. Корисник бира адресу испоруке хране
   2.1 Корисник бира адресу из профила
   2.2 Корисник бира тренутну локацију
   2.3 Корисник ручно уноси адресу испоруке
3. Корисник поручује храну

Алтернативни токови: -

Кориснички интерфејс: састоји се од текстуалног поља за унос адресе и дугмета којим бирамо тренутну адресу на којој се корисник тренутно налази. Када је адреса испоруке хране изабрана, можемо наручити храну притиском на дугме Наручи. Интерфејс је приказан на слици 18.



Слика 18. Кориснички интерфејс случаја употребе „Дефинисање адресе испоруке и наручивање“

3.2.4 Модел базе података

На основу описаних случајева употребе можемо препознати ентитете који ће чинити наш модел.

Корисник: особа која наручује храну. Мора бити регистрован. Садржи неопходне податке на основу којих ћемо знати ко је наручио храну и на коју адресу ће она бити испоручена. Податке које ентитет корисник садржи су:

- Јединствени идентификациони број (цео број)
- Корисничко име (ниска) – мора бити јединствено у целом систему
- Лозинка (ниска) – одабрана лозинка са којом ће се корисник пријављивати на систем. Вредност лозинке ће бити криптована и тако ће се чувати у бази
- Име и презиме (ниска)
- Адреса (ниска) – назив улице
- Број улице (ниска)
- Град (ниска)
- Телефон (ниска)

Ресторан: ентитет који ће садржати неопходне информације о ресторану.

- Јединствени идентификациони број ресторана (цео број)
- Назив (ниска)
- Адреса (ниска)
- Опис (ниска)
- Просечно време доставе (цео број)
- Телефон (ниска)
- Град (ниска)

Храна: сваки ресторан има свој мени. Издвајамо храну као ентитет. Једна врста хране припада одређеном ресторану. Храна припада одређеној категорији. Ентитет храна садржи следеће податке:

- Јединствени идентификациони број (цео број)
- Идентификациони број ресторана (цео број) – ресторан коме припада
- Идентификациони број категорије (цео број) – категорија којој припада
- Назив (ниска)
- Цена (децимални број)
- Опис (ниска) – кратак опис хране
- Састојци (ниска) – списак састојака од којих је храна направљена.

Категорија: ресторани и храна припадају одређеним категоријама. Скуп категорија представља шифарник категорија. Свака категорија садржи јединствени број и назив категорије:

- Јединствени идентификациони број (цео број)
- Број категорије (цео број) – јединствено непромењиво поље

- Назив (ниска)

Категорије ресторана: један ресторан може производити више врста хране и самим тим може имати више категорија. Податке које треба садржити:

- Идентификациони број ресторана (цео број)
- Идентификациони број категорије (цео број)

Наруџбина: ентитет који садржи основне податке о наруџбини коју је корисник извршио. Под основним подацима подразумевамо датум наруџбине, вредност наруџбине, ко је поручио и адреса на коју је потребно испоручити.

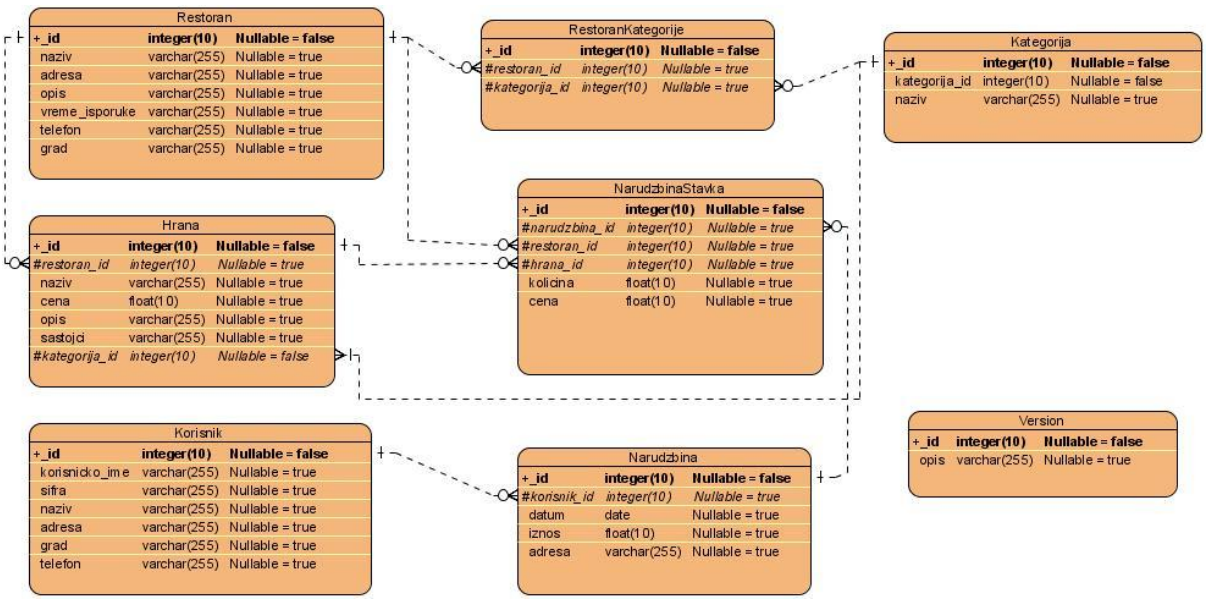
- Јединствени идентификациони број (цео број)
- Идентификациони број корисника (цео број)
- Датум наручивања (ниска) – датум и време поруџбине
- Износ (децимални број) – вредност наруџбине
- Адреса (ниска) – адреса доставе наруџбине

Ставка наруџбине: представља елемент наруџбине. Садржи следеће податке:

- Јединствени идентификациони број (цео број)
- Идентификациони број наруџбине (цео број)
- Идентификациони број ресторана (цео број)
- Идентификациони број хране (цео број)
- Количина (децимални број) – количина наручене хране
- Цена (децимални број) – појединачна цена хране

Верзија базе – ентитет нам говори

На слици 19. је приказан шематски изглед ентитета и њихових односа.



Слика 19. Приказ базе података апликације за наручивање хране

4. Имплементација

Апликација за наричивање хране је имплементирана за уређај са оперативним системом *Android*. Још један од услова који мора бити испуњен је да верзија *Android*-а на уређају буде већа или једнака 3.0 (широко позната као *Honeycomb*). У наставку је приказан део фајла *AndroidManifest* у којем је дефинисан овај услов, као и захтеване дозволе које су потребне да би апликација радила.

```
<uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="17" />

<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission
   android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
```

Корисник приликом инсталације прихвата услове.

4.1 Покретање апликације и регистрација корисника

Први корак у коришћењу апликације је њено покретање и припрема корисника за прегледање и наручивање хране. Приликом првог покретања апликације неопходно је поставити базу података на самом уређају. Након тога потребно је прикупити податке о ресторанима, храни и категоријама са сервера коришћењем сервиса. Подаци се преузимају коришћењем *HTTP* протокола. Процедура је следећа:

- Апликација позива сервис коришћењем класе `HttpPost`, а као аргумент конструктора се прослеђује веб адресу нашег сервиса. Могуће је послати додатне податке, као што су корисничко име и лозинка, коришћењем заглавља;
- Сервис прихвата податке и обрађује их;
- Сервис враћа податке у формату *JSON*;
- У следећем кораку креирамо објекте *JSON*, тј парсирамо податке и користимо у даљем раду;

Пример прикупљања података дат је у наставку кроз методу која се позива приликом покретања апликације за прикупљање информација о ресторанима и храни.

```
public static boolean getDataLoading(Context context) {
Boolean rvalue = true;

try {

  HttpPost post = new HttpPost("http://buzzinn.net/njam/getDataUcitavanje.php");

  String json_return = new CAsinhroniTask().execute(post).get();
```

```
  Log.w("getData JSON return", json_return);
  JSONObject rv = new JSONObject(json_return);
  if(json return == "")
  {
     String obavestenje = rv.getString("obavestenje");
     boolean has_error = rv.getBoolean("has_error");

     if (!has_error)
     {
        // Citamo i kreiramo kategorije
       if (rv.has("restoran_kategorija")) {
         JSONArray restoran_kategorije = rv.getJSONArray("restoran_kategorija");
         CDataWorker.deleteRestoranKategorija();
         for (int i = 0; i < restoran_kategorije.length(); i++) {

            JSONObject row = restoran kategorije.getJSONObject(i);
           int restoran_id = row.getInt("restoran_id");
           int kategorija_id = row.getInt("kategorija_id");

           rvalue = rvalue
          && CDataWorker.updateRestoranKategorija(
          restoran_id, kategorija_id);

       }
       Log.w("data restoran kategorija", restoran kategorije.toString());
     }
   }
}
catch (Exception e) {
    Log.w("getData Exception", e.getMessage());
    return false;
}

return rvalue;
}
```

Пример преузетих података са сервера формата *JSON* дат у наставку:

```
  {
        "has error": false,
        "obavestenje": "Prona\u0111en korisnik!",
        "restoran": [{
                 "_id": "1",
                 "naziv": "Napoli",
                 "adresa": "\u0110u\u0161ina 10",
                 "grad": "Belgrade",
                 "telefon": "0112527764",
                 "napomena": "Prose\u010dno vreme isporuke: 45 min",
                 "opis": ""
        },
        {
                 "_id": "3",
                 "naziv": "Mikan",
                 "adresa": "Mar\u0161ala Birjuzova 21",
                 "grad": "Beograd",
                 "telefon": "011\/2184-543",
                 "napomena": "",
                 "opis": ""
        }],
        "hrana": [{
```

```
        "_id": "1",
        "restoran_id": "1",
        "naziv": "Margarita",
        "cena": "357",
        "sastojci": "Pelat, ka\u010dkavalj, parmezan, masline, origano",
        "opis": "",
        "kategorija_id": "2",
        "ocena": "0"
    },
    {
        "_id": "2",
        "restoran_id": "1",
        "naziv": "Pile\u0107a \u010dorba",
        "cena": "150",
        "sastojci": null,
        "opis": null,
        "kategorija_id": "5",
        "ocena": null
    }]
}
```

Да бисмо могли да прикупимо податке, неопходно је да постоји интернет веза. Уколико не постоји, није могуће коришћење апликације. За проверавање постојања интернет везе користимо класу `ConnectivityManager`. Провера се врши методом `isOnline` приказаној у наставку.

```
public boolean isOnline() {
    ConnectivityManager cm = (ConnectivityManager)
                  getSystemService(Context.CONNECTIVITY_SERVICE);

    boolean rv = cm.getActiveNetworkInfo() != null
            && cm.getActiveNetworkInfo().isConnectedOrConnecting();

    return rv;
}
```

Подаци се прикупљају асинхроно преко уграђене класе `AsyncTask` коју је потребно наследити. `AsyncTask` представља апстрактну класу која помаже при извршавању дугих позадинских операција које бисмо иначе морали да радимо сами преко нити. Имплементација класе `CPreuzimanjePodatakaSaServera` је приказана у наставку.

```
public class CPreuzimanjePodatakaSaServera extends AsyncTask<HttpPost,
Void, String> {

  @Override
  protected String doInBackground(HttpPost... params) {
     StringBuilder builder = new StringBuilder();
     HttpClient client = new DefaultHttpClient();
     HttpResponse response;
     HttpConnectionParams.setConnectionTimeout(client.getParams(), 10000);
     HttpPost post = params[0];
     try {

      response = client.execute(post);
      if (response != null) {
```

```
        InputStream content = response.getEntity().getContent();
        BufferedReader reader =
                new BufferedReader(new InputStreamReader(content));
        String line;
        while ((line = reader.readLine()) != null) {
           builder.append(line);
        }
       }
      }
      catch (Exception e) {
        e.printStackTrace();
      }
      return builder.toString();
   }
}
```

Следећи корак у коришћењу апликације јесте пријава корисника. Провера унетих података се врши пре самог чина регистрације. Лозинка корисника се чува на серверу, али се пре тога шифрује. Шифровање се врши алгоритмом *md5* [46] и на тај начин се осигуравају подаци корисника. Метода која шифрује лозинку приказана је у наставку.

```
public static String md5(String s) {
      MessageDigest digest;
      try {
            digest = MessageDigest.getInstance("MD5");
            digest.update(s.getBytes(), 0, s.length());
            String hash = new BigInteger(1, digest.digest()).toString(16);
            return hash;
      } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
      }
      return "";
}
```

4.2 Одабир ресторана и хране

Након успешне пријаве корисника, приказује се главни мени апликације. Из тог менија корисник може бирати ресторане. Као што смо претходно објаснили, то је могуће урадити из листе свих ресторана или преко мапе ресторана. Уколико корисник жели да одабере ресторан преко мапе неопходно је имплементирати приказ *Google* мапе коришћењем *Google API*-ја. Код који иницијализује мапу приказан је у наставку.

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.mapa_layout);

    map = ((SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.mapview)).getMap();

    Location myLocation = getLokacijaKorisnika();
    LatLng _lokacija;
    Marker  marker;
```

```
    if(myLocation != null)
    {
        _lokacija = new LatLng(getLokacijaKorisnika().getLatitude(),
                        getLokacijaKorisnika().getLongitude());
        _marker = map.addMarker(new MarkerOptions()
        .position(_lokacija)
        .title("Vaša trenutna lokacija")
        .icon(BitmapDescriptorFactory.defaultMarker()));

        map.moveCamera(CameraUpdateFactory.newLatLngZoom(new
                        LatLng(getLokacijaKorisnika().getLatitude(),
                        getLokacijaKorisnika().getLongitude()), 12));
    }
    else
    {
        map.moveCamera(CameraUpdateFactory.newLatLngZoom(new
                              LatLng(44.797283d, 20.460663d), 12));
    }

    for (CRestoran rr : CRestorani.DI().GetRestorani()) {
        GeoPoint point = null;
        Log.w("restorani adresa", rr.getAdresa().toString());

        try {

            HttpPost post = new HttpPost(
                    getGoogleMapUrl(rr.getAdresa()));
            String tmp = new CAsinhroniTask().execute(post).get();

            JSONObject jbs = new JSONObject(tmp);
            point = getLatLong(jbs);

        } catch (Exception e) {
            e.printStackTrace();
            break;
        }

        if (point != null) {
            _lokacija = new LatLng(point.getLatitudeE6() / 1E6,
                                   point.getLongitudeE6() / 1E6);

             marker = map.addMarker(new MarkerOptions()
                        .position(_lokacija)
                        .title(rr.getNaziv())
                        .snippet(rr.getOpis())
                        .icon(BitmapDescriptorFactory
                                .fromResource(R.drawable.hamburger_icon)));

            listaMarkera.put(_marker.getId(), rr);
        }
    }
    map.setOnMarkerClickListener(klikOnMarker);
}
```

Да би се отворила мапа морају бити испуњени предуслови који су описани у одељку 2.3.4.1. Уколико је све у реду, отвара се мапа и постављају се иконице ресторана на мапи на одређеним положајима које одговарају њиховој адреси. На основу адресе,

контактира се *Google* сервис за добијање географске ширине и дужине сваког од ресторана. У наставку су приказане методе које то раде:

```
public String getGoogleMapUrl(String address) {
    address = address.replaceAll(" ", "%20");
    return "http://maps.google.com/maps/api/geocode/json?address="
                + address + "&sensor=false";
}
public String getGoogleMapUrl(LatLng latLng) {

    return "http://maps.googleapis.com/maps/api/geocode/json?latlng="+
                latLng.latitude + ","+ latLng.longitude+"&sensor=false";
}

public GeoPoint getLatLong(JSONObject jsonObject) {
    Double lon = new Double(0);
    Double lat = new Double(0);

    try {
      lon = ((JSONArray) jsonObject.get("results")).getJSONObject(0)
                .getJSONObject("geometry").getJSONObject("location")
                .getDouble("lng");

      lat = ((JSONArray) jsonObject.get("results")).getJSONObject(0)
                .getJSONObject("geometry").getJSONObject("location")
                .getDouble("lat");

    } catch (Exception e) {
        e.printStackTrace();
    }
    return new GeoPoint((int) (lat * 1E6), (int) (lon * 1E6));
}
```

Након постављања ресторана, мапа проналази тренутну локацију корисника и увећава простор око ње. На тај начин корисник може увидети да ли постоје ресторани који се налазе у његовој непосредној близини како би смањили време доставе или можда лично преузели храну.

Кликом на иконицу ресторана приказују се информације о ресторану. Приказује се назив ресторана и његов кратак опис. Такође се приказује и дијалог у коме корисник може отворити ресторан и изабрати храну који ресторан има на менију.

Други начин одабира хране јесте преко листе ресторана. У сваком реду листе приказан је назив ресторана и његов кратак опис. Ресторане је могуће претраживати по називу и по категорији хране коју производе. Листањем ресторана проналазимо онај који нам највише одговара. Имплементација је извршена креирањем активности која наслеђује `ListActivity`. Користи се за приказивање ставки листе које су повезане са одређеним извором података као што је низ или курсор. Да би повезали податке са листом и приказали кориснику, неопходно је да погледа садрђи објекат `ListView` са ид-јем `"@android:id/list"`. Начин имплементације попуњавања листе ресторана приказан је у наставку.

```
private void fillRestoran() {

    Cursor mNotesCursor = CDataWorker.getRestorani();
    startManagingCursor(mNotesCursor);
```

```
    restoranArray = new ArrayList<CRestoran>();
    for (mNotesCursor.moveToFirst(); !mNotesCursor.isAfterLast();
                            mNotesCursor.moveToNext()) {

        restoranArray.add(new CRestoran(mNotesCursor.getString(mNotesCursor
                .getColumnIndex("naziv")), mNotesCursor
                .getString(mNotesCursor.getColumnIndex("napomena")),
                mNotesCursor.getInt(mNotesCursor.getColumnIndex("_id")),
                mNotesCursor.getString(mNotesCursor
                        .getColumnIndex("adresa")), mNotesCursor
                        .getString(mNotesCursor.getColumnIndex("grad")),
                mNotesCursor.getString(mNotesCursor
                        .getColumnIndex("telefon")),
                mNotesCursor.getInt(mNotesCursor
                        .getColumnIndex("kategorija_id"))

                ));
    }

    this.restoranAdapter = new RestoranAdapter(this, R.layout.restoran row,
            restoranArray);
    setListAdapter(this.restoranAdapter);
}
```

Кликом на жељени ресторан, отвара нам се његов профил. Он садржи детаљне информације о самом ресторану и мени са храном. Информације су одвојене табовима. Табови се иницијализују као део контроле `ActionBar`. Прво се мора поставити навигациони мод на више табова, а потом сетовати сваки од табова. У наставку је приказан начин постављања табулатора.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    ActionBar actionBar = getSupportActionBar();

    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);


    if(getIntent().getExtras() != null)
    {
        restoran_id = getIntent().getExtras().getInt("restoran_id");
        restoran_naziv =
                   getIntent().getExtras().getString("restoran_naziv");
    }

    Bundle bundle = new Bundle();
    bundle.putInt("restoran_id", restoran_id);
    bundle.putString("restoran naziv", restoran naziv);


    Tab tab1 = actionBar.newTab()
            .setText(Menu SPEC)
            .setTabListener(new TabListener<HranaActivity>(
            this, Menu_SPEC, HranaActivity.class, bundle));
    actionBar.addTab(tab1);

    Tab tab2 = actionBar.newTab()
            .setText(Info_SPEC)
```

```
            .setTabListener(new TabListener<HranaInfoActivity>(
            this, Info_SPEC, HranaInfoActivity.class, bundle));
    actionBar.addTab(tab2);

    CRestoran rest = CRestorani.DI().GetRestoran(restoran_id);

    actionBar.setTitle(rest.getNaziv());
    if( savedInstanceState != null ){
        getSupportActionBar().setSelectedNavigationItem(
            savedInstanceState.getInt(STATE_SELECTED_NAVIGATION_ITEM));
    }

}
```

4.3 Наручивање хране

У претходном одељку смо објаснили процес одабира ресторана и додавања хране. Храну коју смо изабрали, налази се у виртуалној корпи. Корпа се налази у менију апликације и одатле се може изабрати. Након одабира корпе отвара се листа изабране хране, њена количина и цена, као и укупан износ наруџбине. Уколико се изабере корпа, а она је празна, корисник се обавештава о томе. За сваку од ставки које се налазе у корпи постоји могућност брисања, тј. избацивања из корпе. Инстанца класе `CKorpa` представља синглтон. То значи да можемо имати само једну инстанцу класе на нивоу апликације. Имплементација синглтона представљена је у наставку.

```
private static CKorpa instance = null;
ArrayList<CKorpaStavka> korpaStavkaArray;

public int id;
public String datum;
public double ukupnaIznos;
public String adresa;
public int narudzbinaID;

public String opisStavki;

public CKorpa() {
    korpaStavkaArray = new ArrayList<CKorpaStavka>();

}

public static CKorpa DI() {
    if (instance == null)
        instance = new CKorpa();

    return instance;
}
```

Након прегледа корпе и ставки који садржи, можемо наставити са наручивањем хране. Отвара се активност у коме је потребно одабрати адресу доставе хране. У поље са адресом се аутоматски уписује адреса корисника коју је он уписао приликом регистрације. Кориснику се оставља опција да промени адресу ручно или да апликација упише његову тренутну локацију користећи сервисе за лоцирање. За лоцирање корисника користимо методу `getLokacijaKorisnika` која је приказана у наставку. Она проналази локацију корисника најбољим доступним сервисом, првенствено *GPS*-ом, а затим коришћењем мреже.

```
public Location getLokacijaKorisnika() {

    Location location = null;
    try {
        locationManager = (LocationManager) _context
                .getSystemService(LOCATION_SERVICE);

        boolean isGPSEnabled = locationManager
                .isProviderEnabled(LocationManager.GPS_PROVIDER);

        boolean isNetworkEnabled = locationManager
                .isProviderEnabled(LocationManager.NETWORK_PROVIDER);

        if (!isGPSEnabled && !isNetworkEnabled) {
        } else {
            boolean canGetLocation = true;
            if (isGPSEnabled) {
                if (location == null) {
                    locationManager.requestLocationUpdates(
                            LocationManager.GPS PROVIDER,
                            9000,
                            9000, mLocationListener);
                    Log.d("GPS", "GPS Enabled");
                    if (locationManager != null) {
                      location = locationManager
                       .getLastKnownLocation(LocationManager.GPS_PROVIDER);

                    }
                }
            }
            if (isNetworkEnabled) {
                locationManager.requestLocationUpdates(
                        LocationManager.NETWORK_PROVIDER,
                        9000,
                        9000, mLocationListener);
                Log.d("Network", "Network Enabled");
                if (locationManager != null) {
                  location = locationManager
                   .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);

                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return location;
}
```

Након одређивања адресе, можемо да пошаљемо наруџбину ресторану. Шаљемо текст у *JSON* формату нашем веб серверу чија је даље брига да информације о наруџбини проследи ресторану. Поред тога, његов задатак је да сачува информације о наруџбини корисника, како би корисник имао историју својих наруџбина. Информације које су неопходне приликом слања наруџбине су садржај корпе и корисник који врши наруџбину. Имплементација слања наруџбине приказана је у наставку.

```
public static boolean putNarudzbina(CKorisnik korisnik, CKorpa korpa,
        String adresa, Context ctx) {
```

```
    if (korisnik.PostojiKorisnik == false) {
        Toast.makeText(ctx, "Ne postoji korisnik!", Toast.LENGTH_SHORT)
                .show();
        return false;
    }
    if (korpa == null || CKorpa.DI().GetNarudzbine().isEmpty() == true) {
        Toast.makeText(ctx, "Korpa je prazna!", Toast.LENGTH_SHORT).show();
        return false;
    }

    boolean rv = true;

    JSONObject narudzbina = getJSONFromKorpa(korisnik, korpa, adresa);


    HttpPost post = new HttpPost(
            "http://buzzinn.net/njam/putNarudzbina.php");
    ByteArrayEntity baEntity = null;
    try {
        baEntity = new ByteArrayEntity(narudzbina.toString().getBytes(
                "UTF8"));
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
        rv = false;
    }
    baEntity.setContentEncoding(new BasicHeader(HTTP.CONTENT_TYPE,
            "application/json"));
    post.setEntity(baEntity);

    try {
        String json_return = new CAsinhroniTask().execute(post).get();
        JSONObject rvalue = new JSONObject(json_return);
        JSONObject obavestenje = rvalue.getJSONObject("obavestenje");

        String command = obavestenje.getString("command");
        boolean has_error = obavestenje.getBoolean("has_error");
        String data = obavestenje.getString("obavestenje");

        if (has_error) {
            Toast.makeText(ctx, data, Toast.LENGTH_LONG).show();
        }

        rv = rv && has error;

    } catch (InterruptedException e) {
        rv = false;
        e.printStackTrace();
    } catch (ExecutionException e) {
        rv = false;
        e.printStackTrace();
    } catch (JSONException e) {

        e.printStackTrace();
    }

    return rv;
}
```

4.4 Приказ профила и историје наруџбина корисника

Корисник након пријаве на систем има могућност измене свог профила и прегледа претходних наруџбина. Податке који корисник може мењати су име, адреса, број телефона и шифра. Шифра се може променити уколико се претходно унесе стара шифра. Подаци који се унесу чувају се у локалној бази као и у бази на серверу коришћењем веб сервиса. Инстанца класе `CKorisnik` је синглтон, слично као и инстанца класе `CKorpa`. Објекат класе се инстанцира приликом пријаве на систем корисника или приликом иницијализације апликације, уколико је корисник претходно пријављен. Скраћени приказ имплементација класе `CKorisnik` приказан је у наставку.

```
public class CKorisnik {

    private static CKorisnik instance;
    public boolean PostojiKorisnik = false;

    public String Naziv = "";
    int ID = -1;
    String UserName = "";
    String Password = "";
    String Adresa = "";
    String Grad = "";
    String Telefon = "";

    public CKorisnik()
    {
    }

    public static CKorisnik DI()
    {
        return DI(false);
    }

    public static CKorisnik DI(Boolean ucitaj)
    {

        if(instance == null)
        {
            instance = new CKorisnik();
        }

        if(ucitaj)
            instance.UcitajKorisnika();
        return instance;
    }

    private void UcitajKorisnika()
    {
        Cursor mNotesCursor = CDataWorker.getKorisnik();

        if(mNotesCursor == null || mNotesCursor.getCount() == 0)
        {
            this.PostojiKorisnik = false;
            return;
        }

        mNotesCursor.moveToFirst();
```

```
        this.ID = mNotesCursor.getInt(mNotesCursor.getColumnIndex("_id"));
        this.Naziv =
           mNotesCursor.getString(mNotesCursor.getColumnIndex("naziv"));
        this.UserName =
           mNotesCursor.getString(mNotesCursor.getColumnIndex("username"));
        this.Password =
           mNotesCursor.getString(mNotesCursor.getColumnIndex("password"));
        this.Adresa =
           mNotesCursor.getString(mNotesCursor.getColumnIndex("adresa"));
        this.Grad =
           mNotesCursor.getString(mNotesCursor.getColumnIndex("grad"));
        this.Telefon =
           mNotesCursor.getString(mNotesCursor.getColumnIndex("telefon"));

        this.PostojiKorisnik = true;
    }

    public boolean Save()
    {

        CDataWorker.updateKorisnika(getID()+"", getUserName(),
                          getPassword(), getNaziv(),  getAdresa(),
                          getGrad(), getTelefon());
        return true;
    }

    public boolean Delete()
    {
        return CDataWorker.deleteKorisnik(getUserName());
    }

    ...
}
```

Синглтон објекат се поставља на null приликом одјављивања корисника са система.

4.5 Опис имплементације сервиса

База података која се налази на серверу се састоји од ентитета који су описани у поглављу 3.2.4. Подаци о новим ресторанима и храни се уписују на *Android* уређај на начин који је описан у поглављу 4.1, односно приликом покретања *Android* апликације. База која се налази на серверу ће садржати податке о ресторанима, храни, категоријама, свим корисницима апликације и њиховим наруџбинама. Појединачни корисник ће имати податке о ресторанима, храни, свом профилу и својим наруџбинама.

У претходним одељцима често смо спомињали коришћење сервиса за комуникацију са сервером. Сервиси нам служе за прикупљање података са сервера, као што су најновије информације о ресторанима и храни. Исто тако коришћењем сервиса можемо слати податке серверу када желимо да се нешто упише у базу. У наставку ћемо приказати операције које се извршавају над ентитетима базе података:

- Ресторан: читање
- Храна: читање
- Корисник: читање, писање, ажурирање

- Наруџбина: читање, писање

Да би било могуће извршити поменуте операције, морају бити испуњени одређени услови. Мора да постоји сервер на коме би се чувала база и који подржава извршавање *php* (енг. *Hypertext Preprocessor*) [47] програма. Један од таквих сервера је *Apache* [48]. *Apache* је *HTTP* (енг. *Hyper Text Transfer Protocol*) [49] сервер који се развија од стране *Apache* заједнице као пројекат отвореног кода. Он подржава различите функционалности који се протежу од подршке за извршавање различитих скрипт језика као што су *Perl, Python, PHP* до подршке пружања сигурности у раду једног веб сајта. Све што је потребно да урадимо је да направимо базу података и сервисе, и поставимо их на један овакав сервер коме ћемо моћи да приступимо спољa, тј на сервер који је активан.

У наставку ће бити приказан садржај скрипте која се налази на серверу и омогућава регистрацију корисника, а позива из *Android* апликације.

```
<?php

$json = file_get_contents('php://input');

if($json == NULL){
    $json=$_GET ['json'];
    echo json encode($json);
    die();
}

$obj = json decode($json);

$username = $obj->{'un'};
$pass = $obj->{'pass'};
$ime = $obj->{'naziv'};
$adresa = $obj->{'adresa'};
$grad = $obj->{'grad'};
$telefon = $obj->{'telefon'};

$return data = array();
$return_data["command"] =
     "SELECT * FROM db496392122.korisnik WHERE username = '".$username."'
                              ";
$return_data["has_error"] = false;
$return_data["obavestenje"] = "";

header('Content-type: application/json');
if($username == "" || $pass == "")
{
   $return_data["has_error"] = true
   $return_data["obavestenje"] =
               "Morate uneti korisničko ime ili lozinku!";

   echo json_encode($return_data);
   die();
}

define('DB NAME', 'db496392122');
define('DB_USER', 'dbo496392122');
define('DB_PASSWORD', 'njamnjam');
define('DB_HOST', 'db496392122.db.1and1.com');

$con = mysql_connect(DB_HOST,DB_USER,DB_PASSWORD);
```

```
$db = mysql_select_db(DB_NAME, $con);

if (!$con)
{
  die('Could not connect: ' . mysql_error());
}
else
{
  $result = mysql_query($return_data["command"]);
  $num_rows = mysql_num_rows($result);

  if ($num_rows > 0) {
    $return_data["has_error"] = true;

     $return_data["obavestenje"] =
           "Postoji korisnik sa istim korisničkim imenom!";

    echo json_encode($return_data);
    mysql_close($con);
    die();
 }
 else
 {
   $return_data["command"] =
        "INSERT INTO db496392122.korisnik(username, password, naziv,
adresa, grad, telefon) VALUES ('".$username."','".$pass."', '".$ime."',
'".$adresa."', '".$grad."',    '".$telefon."')";

   $result = mysql_query("INSERT INTO db496392122.korisnik(username,
password, naziv, adresa, grad, telefon) VALUES
('".$username."','".$pass."', '".$ime."', '".$adresa."', '".$grad."',
'".$telefon."')");
    if(!$result)
    {
      $return_data["has_error"] = true;
      $return_data["obavestenje"] =
            "Greška prilikom pisanja u bazu!".mysql_error()." AAA";

      echo json_encode($return_data);
      mysql_close($con);
      die();
    }
    else
    {
      $return_data["has_error"] = false;
      $return_data["obavestenje"] = "Uspešno kreiran novi korisnik!";

      echo json_encode($return_data);
      mysql_close($con);
      die();
    }
  }
}
```

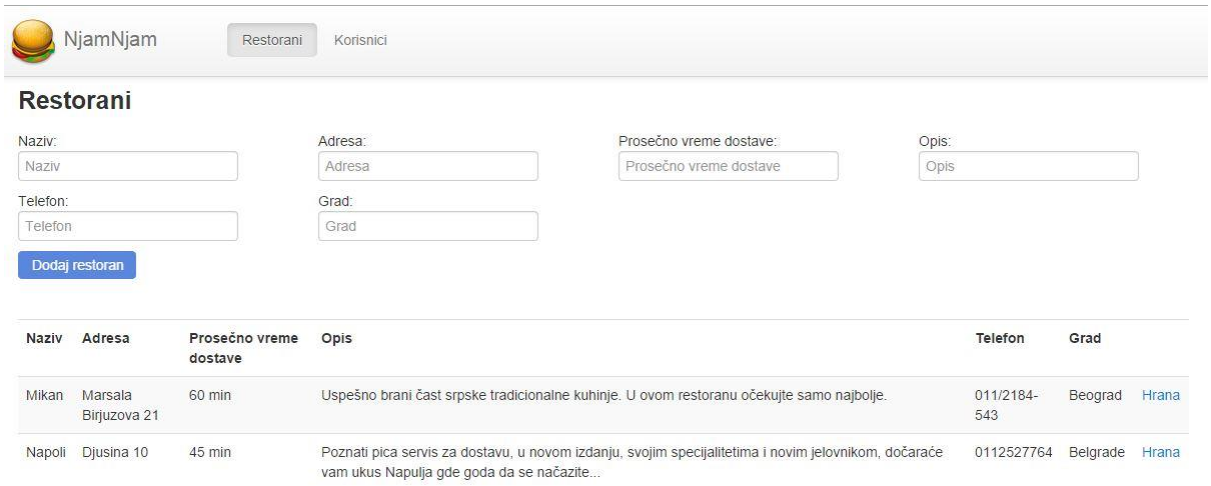
Можемо уочити да се прво прихвата послати објекат *JSON* који садржи податке о кориснику. Након тога се проверава да ли корисник са прослеђеним корисничким именом постоји у бази. Убрзо затим, уколико не постоји, врши се упис новог корисника у базу. На крају враћа се одговор апликацији у коме се сажете информације о успешности креирања новог корисника.

4.6 Ажурирање информација о ресторанима, храни и наруџбинама

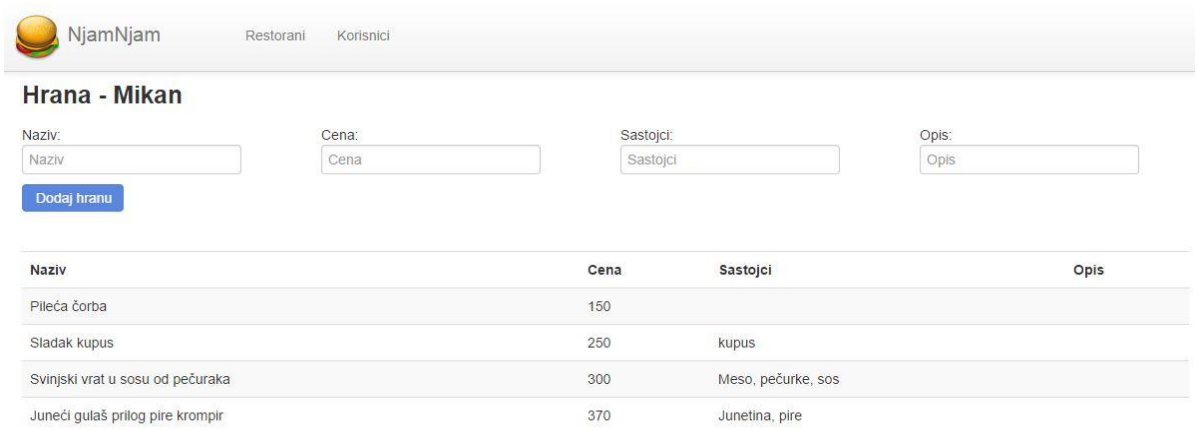
За одржавање базе података која се налази на серверу задужен је администратор. Информације које може видети су списак свих ресторана и хране која се налази у систему. Може видети регистроване кориснике и наруџбине које су они остварили. Поред одржавања, његов задатак је и унос нових ресторана и хране. Администратор извршава своје задатке коришћењем веб портала који се налази на адреси http://tinyurl.com/NjamAplikacija. На слици 20. приказана је форма за приказ постојећих и унос нових ресторана. Администратор уноси неопходне податке за ресторан као што су назив, адреса, телефон, опис, просечно време испоруке и град. Уколико су сви подаци унети, могуће је креирати нови ресторан. Након успешног додавања ресторана, кликом на линк *Храна*, који се налази у истом реду као и ресторан, отвара се приказ хране која се налази у том ресторану. Уколико желимо да додамо нову храну, потребно да се попуне захтевана поља као што су назив, цена, састојци и опис и кликнемо на дугме *Додај храну*. Форма за унос хране налази се на слици 21.

Прилико додавања нових вредности у базу, мења се верзија базе на серверу у табели *version*. На тај начин, *Android* апликација за наручивање хране приликом покретања зна да ли има нових података на серверу или не. На тај начин се не преузимау подаци са интернета уколико то није потребно.

Приспеле наруџбине би требало да се прослеђују ресторанима. То би могао да ради администратор или посебна апликација тј. сервис. Тај део није имплементиран у овом мастер раду. Наруџбине долазе до сервера и везују се за клијенте. Списак наруџбина сваког корисника се може видети на веб порталу.



Слика 20. Приказ форме за додавање ресторана



Слика 21. Приказ форме за додавање хране у изабраном ресторану

Веб портал је креиран коришћењем *MVC* (енг. *Model View Control*) [50] оквира *AngularJS* [51]. Детаљи око израде веб портала неће бити представљени зато што нису тема овог рада.

5. Закључак

У претходни поглављима упознали смо се оперативном системом *Android,* са његовим историјатом и видели како је све почело. Сазнали смо како изгледа архитектура оперативног система и од којих се све слојева састоји. С друге стране, приказан је процес развоја једне апликације и представљено знање које је потребно да би се једна апликација креирала. Поред знања, неоходни су и алата како би наше идеје биле спроведене у делу. Кроз практичан пример, израдом апликације за наручивање хране, представљен је део могућности који пружа оперативном системом *Android.*

Као највећи проблем при изради апликације истакао бих усклађивање изгледа са многобројним величинама и резолуцијама екрана. Ту треба бити јако обазрив и пажљиво пратити упуства званичне документације. Наравно, неопходно је тестирати понашање апликације на различитим уређајима. Такође, треба обратити пажњу и на верзије оперативног система за које се апликација развија. Из верзије у верзију додају се нове могућности и проширују постојеће. Мора се пазити које се функције позивају јер се лако може десити да оне нису подржане на претходним верзијама. *ADT* неће пријављивати грешку приликом израде апликације уколико користимо такве функције. Сама апликација ће пријавити грешку тек када се позове функција коју верзија оперативног система не подржава. Решење је коришћење библиотеке подршке (енг. *Support Library*) која нову функционалност успешно пропагира на ниже верзије оперативног система.

Од унапређења постојећег апликативног решења издвојио бих неколико ствари. Прва би била додавање оцена и коментара корисника везана за храну и ресторане. На тај начин би корисници олакшали одабир хране другим корисницима. С друге стране, потребно је дорадити апликацију и укључити ресторане у систем наручивања, тако што би наруџбина заиста стизала до њих. На тај начин бисмо имали функционалан производ који би могао да буде конкурентан на тржишту наручивања хране.

Android је за кратко време постао најпопуларнији оперативни систем за мобилне телефоне и таблете. Томе су поред његове архитектуре и интерфејса, допринели и инжењери који су направили многобројне занимљиве и корисне апликације доступне корисницима. Више него добра документација и јасна објашњења и смернице које потпомажу развој апликација омогућили су брзо савладавање система и једноставан развој апликација. Као програмер који се пре апликације за наручивање хране није сусрео са проблемом израде *Android* апликације, могу рећи да сам брзо проналазио решења за проблеме са којима сам се сусретао. Највећу помоћ ми је представљала званична *Android* документација и програмерска заједница.

Оперативни систем *Android* је тренутно неприкосновен на тржишту мобилних уређаја. Конкуренција је далеко иза њега и на све начине се труди да га сустигне. У блиској будућности не видим да ће у томе успети и угрозити чврсту и јаку подлогу на којој тренутно стоји *Android.* Тржиште се полако окреће уређајима као што су паметни сатови и паметне наочаре и очекујем да ће *Android* опет бити у самом врху. Генерално, предвиђам лепу будућност овом оперативном систему који свакодневно напредује како би својим корисницима пружио већу функционалност, удобност и сигурност.

6. Литература

[1] http://www.itu.int/en/ITU-D/Statistics/Pages/default.aspx (приступао 24. августа 2014)

[2] http://www.gartner.com/newsroom/id/2692318 (приступао 24. августа 2014)

[3] *Hello, Android: Introducing Google's Mobile Development Platform*, Ed Burnette, Pragmatic Bookshelf (2009)

[4] http://www.idc.com/getdoc.jsp?containerId=prUS24676414 (приступао 24. августа 2014)

[5] Eclipse, https://www.eclipse.org, *The Eclipse Foundation* (приступао 24. јануара 2014)

[6] *Android SDK*, http://developer.android.com/sdk/index.html, (приступао 24. јануара 2014)

[7] *Linux Kernel Architecture*, Wolfgang Mauerer, Wiley Publishing, Inc (2008)

[8] A complete history of Android, http://www.techradar.com/news/phone-and-communications/mobile-phones/a-complete- history-of-android-470327 , Gareth Beavis (приступао 26. јануара 2014)

[9] *Google*, https://www.google.com/about/, *Google Inc* (приступао 26. јануара 2014)

[10] *Andy Rubin*, http://www.crunchbase.com/person/andy-rubin, *Crunch Base* (приступао 26. јануара 2014)

[11] *Open Handset Alliance*, http://www.openhandsetalliance.com, *Open Handset Alliance* (приступао 26. јануара 2014)

[12] *HTC Dream*, http://en.wikipedia.org/wiki/HTC_Dream , *Wikipedia* (приступао 26. јануара 2014)

[13] *Android 1.6 Platform Highlights*, http://developer.android.com/about/versions/android-1.6-highlights.html, *Google Inc* (приступао 26. јануара 2014)

[14] *Android 1.5 Platform*, http://developer.android.com/about/versions/android-1.5.html, *Google Inc* (приступао 26. јануара 2014)

[15] *The Open Source Initiative*, http://opensource.org, *Open Source Initiative* (приступао 1. фебруар 2014)

[16] http://www.openhandsetalliance.com/android_overview.html (приступао 26. јануара 2014)

[17] *Intraduction to Virtual Machine*, http://jes.ece.wisc.edu/papers/vm.pdf, 0J. E. Smith and Ravi Nair (приступао 26. јануара 2014)

[18] *An introduction to android*, H Xuguang, Dababase Lab (2009)

[19] *2D computer graphics*, http://en.wikipedia.org/wiki/2D_computer_graphics, *Wikipedia* (приступао 26. јануара 2014)

[20] *Computer Graphics with Open GL*, Donald D. Hearn, M. Pauline Baker, Warren Carithers Prentice Hall Press Upper Saddle River, NJ, USA (2010)

[21] *ARM System Architecture*, Steve B. Furber, Addison-Wesley Longman Publishing Co. (1996)

[22] *x86 Instruction Set Architecture*, Tom Shanley, Mindshare Press (2010)

[23] *Android Application Development: Programming with the Google SDK*, Rick Rogers, John Lombardo , Zigurd Mednieks, Blake Meike, *O'Reilly Media* (2009)

[24] *Apache License*, http://www.apache.org/licenses/LICENSE-2.0.html, *The Apache Software Foundation* (приступао 26. јануара 2014)
[25] *GNU General Public License*, https://www.gnu.org/copyleft/gpl.html, *Free Software Foundation* (приступао 26. јануара 2014)
[26] *The C programming language*, Brian W. Kernighan and Dennis Ritchie, Prentice Hall; 2 edition (1988)
[27] *The design and implementation of the 4.3BSD Unix operating system*, Leffler, Samuel J ; Karels, Michael J, Reading, MA : Addison-Wesley (1989)
[28] *JavaScript: the definitive guide*, David Flanagan, O'Reilly Media, Inc (2002)
[29] *The Dalvik Virtual Machine*, Davide Hringer, Techn. Report (2010)
[30] *Java 2: the complete reference*, Herbert Schildt, McGraw-Hill Professional (2000)
[31] *Application programming interface* http://en.wikipedia.org/wiki/Application_programming_interface (приступао 12. Фебруара 2014)
[32] *Android Building and Running*, http://developer.android.com/tools/building/index.html, Google Inc (приступао 12. Фебруара 2014)
[33] *Microsoft Windows*, http://windows.microsoft.com/en-us/windows/home, Microsoft (приступао 17. фебруара 2014)
[34] *LIFO (computing)*, http://en.wikipedia.org/wiki/LIFO_(computing), Wikipedia (приступао 17. фебруара 2014)
[35] *Google Play*, http://en.wikipedia.org/wiki/Google_Play, Wikipedia (приступао 22. фебруара 2014)
[36] *QEMU: a Multihost, Multitarget Emulator*, Daniel Bartholomew, Linux Journal (2006)
[37] *Short Message Service*, http://www.3gpp2.org/public_html/specs/cs0015-0.pdf, 3GPP2 (приступао 22. фебруара 2014)
[38] *Global Positioning System*, Prof. Dr. Bernhard Hofmann-Wellenhof, Dr. Herbert Lichtenegger, Dr. James Collins, Springer-Verlag (1993)
[39] https://developers.google.com/console/help/#generatingdevkeys (приступао 26. фебруара 2014)
[40] *The Definitive Guide to SQLite*, Michael Owens, Apress (2006)
[41] *The application/json Media Type for JavaScript Object Notation (JSON)*, Douglas Crockford, http://tools.ietf.org/html/rfc4627 (2006)
[42] *Extensible Markup Language (XML)*, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, John Cowan, W3C (2006)
[43] *Rate your application content*, https://support.google.com/googleplay/android-developer/answer/188189 (приступао 26. фебруара 2014)
[44] *How UML is used*, Brian Dobing, Jeffrey Parson, Magazine ACM (2006)
[45] *Use Case Modeling*, Kurt Bittner, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA (2002)
[46] *The MD5 Message-Digest Algorithm*, R. Rivest, MIT Laboratory for Computer Science and RSA Data Security, Inc. (1992)
[47] *Core PHP programming*, Leon Atkinson, Zeew Suraski, Prentice Hall Ptr Core Series (2003)

[48] *A case study of open source software development: the Apache server*, Audris Mockus, Roy T. Fielding, James Herbsleb, ACM New York, NY, USA (2000)
[49] *Hypertext Transfer Protocol -- HTTP/1.0*, T. Berners-Lee, R. Fielding, H. Frystyk, http://www.hjp.at/doc/rfc/rfc1945.html (приступао 12. марта 2014)
[50] *Web-application development using the Model/View/Controller design pattern*, IEEE, Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International (2001)
[51] *AngularJS web application development*, Peter Bacon Darwin, Pawel Kozlowski, Birmingham,Packt Publ., (2013)