Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Индустријско инжењерство
Ниво студија: Мастер академске студије
Модул: Пословни информациони системи
Предмет: Индустријски рачунарски системи
Број индекса: 341/2018

Филип З. Васић

Андроид апликација за аутоматско управљање складиштем

Мастер рад

Комисија за преглед и одбрану:

1. **др Миладин Стефановић - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

У оквиру овог мастер рада кандидат треба да

Имплементира систем и апликацију који дају могућност аутоматског управљања складиштем, уз примену методе скенирања бар-кодова и QR кодова. Затим прикаже архитектуру система и методологију решења. Циљ рада је приказ система аутоматског управљања, на примеру складишта, уз примене савремених технологија и решавању реалних проблема. Резултат односно исход рада је приказ теоријско-истраживачког рада студента, примењен на решавању сложеног реалног проблема.

Препоручена литература:

[1] Pulungman Reza, (2013), "*Design of an Intelligent Warehouse Management System".*

[2] Wang Jingna, Guowei Hua, (2019), "*Design of Android Warehouse Management Software based on Web Service*".

[3] Mingxing Deng, Jian Mao, Xingwen Gan, (2016), "*Development of Automated Warehouse Management System*".

Крагујевац, 13.05.2019.

Ментор:
др Миладин Стефановић

Садржај

Abstract:
This paper describes Android and Django technologies, also presents the theory of communication between these two systems, as well as the necessary theoretical knowledge to understand how such system works. A system that combines these two technologies is described, system that automates management of warehouse. Technology for scanning bar codes and QR codes is described, for reader to understand practical usage of system later. Then papire describes, applying this technology to system implemented on the Android platform and the Django project. The advantages, capabilities and potential that can be achieved by implementing such a warehouse management system are described. The system enables simple commands and application of the camera as a code scanner to manage warehouse, with the management process being as automated as possible. The paper describes the solution and shows the layout of the practical system, more precisely application. At the very end, the paper presents the direction of further development of this or similar system.

Key words: Android, QR code, barcode, scanning, warehouse, automation

Резиме рада:
У раду су описане технологије Андроид и Django, такође је представљена теорија комуникације између ова два система, као и увод у потребна теоријска знања за разумевање функционисања оваквог система. Описан је систем који у спрези има ове две технологије, представљен је систем који аутоматизује управљање, односно рад аутоматског управљања складиштем. Описана је технологија скенирања бар-кодова и QR кодова. Применом ове технологије на систему који је имплементиран на Андроид платформи и Django пројекту. Приуказане су предности, могућности и потенцијал које имплементацијом једног оваквог система у складишту могу бити постигнуте. Систем даје могућност једноставним командама и применом камере као скенера кодова управљање складиштем, при чему је процес управљања максимално аутоматизован, колико је то могуће. У раду је описано решење и приказан изглед практичног система, односно апликације. На самом крају у раду је представљен смер даљег развоја овог или сличног система.

Кључне речи: Андроид, QR code, бар-код, скенирање, складиште, аутоматизација

Предговор

Пре свега захваљујем се ментору на помоћи приликом одабира теме мастер рада, као и на помоћи коју је пружио приликом израде самог рада. Такође се захваљујем фирми АС из Аранђеловца, и свим запосленима у истој, као и власнику са којим сам у сарадњи израдио и применио систем у пракси, систем који је описан у раду. Захваљујем се и свим професеорима који су били уз мене током досадашњег школовања.

1. Увод

Развојем информационих технологија, система база података и WEB технологија отварају се нове могућности аутоматизације великог броја система. Последњих година напретком технологија је приближена многима, поготово развојем интернета и WEB базираних технологија. Велики допринос томе је поједностављење, како доступности тако и развоја. Често долази до потребе за сједињавањем реалног света и виртуелног, односно реалних проблема са апликативним решењима. Свет који нас окружује пред нас поставља разне препреке, а развојем технологије долазимо у ситуацију где је све чешћи начин решавања ових проблема, најбољи применом информационих технологија. Развојем база података, WEB система, паметних телефона и њиховог софтвера, са доступношћу свима, и могућности да сви приступају систему путем интернета, отворене су нове могућности за даљи развој. Такође доступношћу камера већини, и све чешћом применом кодова за скенирање и скенера, системи аутоматизације помоћу скенирања су све чешћи. Њихове предности су да олакшавају послове запосленима, као и смањивање могућности за грешком до које може доћи људском фактором, јер већину послова на којима може доћи до грешке не би обављали људи, већ систем, односно софтвер (Mingxing et al., 2012).

Складиште као систем, односно складиште које посматрамо као систем, има велики потенцијал, али и велику потребу за аутоматизацијом, сходно својим карактеристикама. Складиште производа у себи садржи огроман потенцијал примене информационих технологија, чување где се шта и колико налази, вођење података о улазу односно излазу робе и друго. Уз то применом савремених технолгоја скенирања (коришћења камере коју поседује већина телефона) и применом технологија паметних телефона, овакав систем је приближен свим људима, уз једноставно коришћење и ниску цену. За ниску цену је јако важан податак да већина углавном већ поседује потребну опрему (паметни телефон са камером). Доступност информација је велика и може им се приступити било када, с обзиром да се подаци налазе сачувани у бази, а њој се приступа са мобилног телефона који је готово увек код корисника. Поред низа предности пружа могућност проширивања система за друге видове примене. Слични системи су и раније прављени и описани у радовима [4, 5, 8], идеја је далеко од нове, али начин имплементације као и другачији аспкти погледа на решење су јединсвени.

Овај рад се бави теоријским аспектима аутоматизације једног система, ако и практичне реализације аутоматизације. Кроз пример аутоматизације складишта, представљене су основе технологије мобилних апликација и WEB система. Кроз примере примене датих технологија је показана једноставност коришћења истих, и бенифити увођењем једне овакве аутоматизације система (Wang et al., 2019). Као главне предности су показане уштеде времена, новца и олакшавање послова које је потребно да раде људи, а послови које могу обављати машине. У првом делу рада описане су теоријске основе потребне за разумевање рада. То су основе андроид апликација, тероија иза Django пројекта, као и теоријиска основа бројних кодова (QR и barcode). Уз то су представљене технологије базе података и комуникације између елемената система, представљене су у

теорији, али се касније примењују и практично. Након тероијског увода који читаоца овог рада припрема за разумевање целог система, представљена је структура система, архитектура имплементираног решења, као и практично решење свих проблема до којих одлази током имплементације сличног система. Описана је база података која се налази иза структуре, односно система. Разјашњена је комуникација у систему, између компоненти, и појашњени су задаци које и који део система треба да обави. На крају је описано практично решење, приказан изглед апликације и описано коришћење, односно практична примена имплементираног система.

Кроз овај рад читалац је уведен у тероију једног оваквог система, представљена је имплементација система и приказана примена, односно коришћење. На самом крају рада дата је теоријска смерница како даље допринети развоју сличаних система, и којим смером би један овакав систем и пратеће технологије били развијени. Циљ рада је направити систем који решава реалан проблем и описати његове теоријске делове. Циљ је направити систем, односно апликацију која применом савремених технологија (паметних телефона, база података, методама скенирања бар-кодова и других технологија) и описати архитектуру једног таквог система. Резултат рада, односно исход рада, треба да буде приказ теоријско-истраживачког рада студента, примењен на решавању сложеног реалног проблема. Односно спремност студента за рад на практичним проблемима са којима се може срести у свом даљем животу.

2. Теоријске основе и преглед технологија

2.1. Андроид

Андроид је оперативни систем заснован на “Linux” језгру, првенствено дизајниран за мобилне уређаје са екраном осетљивим на додир, као што су паметни телефони и таблет уређаји. “Android Inc.” је основан 2003. године у Калифорнији од стране Анду Рубина, Рич Минера, Ник Сеарса и Крис Bsajt-а. Андроид је развила истоимена компанија (енгл. Android Inc.) коју је компанија “Google” финансијски подржавала, а касније и купила, 2005. године. Андроид је представљен 2007. године заједно са оснивањем удружења “Open Handset Alajans” (OHA) (engl. Open Handset Alliance, OHA): конзорцијума хардверских, софтверских и телекомуникационих компанија посвећених развоју отворених стандарда за мобилне уређаје. Први Андроид телефон је продат у октобру 2008. године. За разлику од осталих оперативних система, као што су на пример iOS, који покреће iPhone, BlackBerry OS, ili Winodws Phone OS који су под потпуном контролом компанија Apple, Research in Motion, односно Microsoft. Андроид је оперативни систем отвореног кода доступан под Апач лиценцом (енгл. Apache License) (Šukalo, 2016). Она допушта слободну измену и дистрибуцију софтвера од стране произвођача уређаја, телекомуникационих оператера и програмера ентузијаста (Šukalo, 2016). И поред тога већина Андроид уређаја долази са додатним комерцијалним софтвером. Андроид, такође, има велику заједницу програмера апликација, које проширују функционалност уређаја и најчешће се развијају у програмском језику Јава (Војиновић, 2012). Андроид је најпопуларнији мобилни оперативни систем, а од 2013. уређаји са Андроид оперативним системом продати су у више примерака од Winodws Phone OS, iOS i Mac OS уређаја заједно. У трећем кварталу 2013. удео Андроида на глобалном тржишту смарт телефона био је 81,3%, највиши икад. Од јула 2013. “Google Play” продавница има објављено преко 1 милион Андроид апликација, а преко 50 милијарди апликација је преузето од стране корисника. Истраживање спроведено међу програмерима у априлу-мају 2013. утврђено је да Андроид користи 71% мобилних програмера. Овакавим успехом Google је од Андроида направио идеалну мету за подизање оптужница за кршење патената од стране других технолошких компанија у такозваном „smartphone wars“-у (ратови паметних телефона). А од септембра 2013. активирано је милијарду Андроид уређаја (Feinbube, 2011).

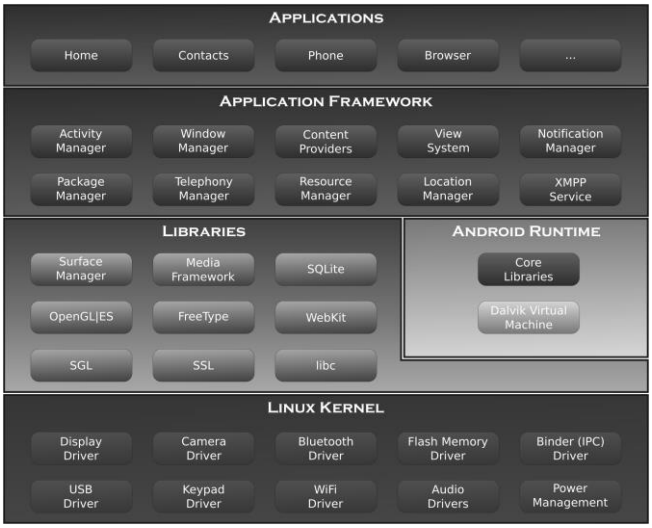
2.1.1. Архитектура Андроид оперативног система

Архитектура Андроид оперативног система се може поделити у четири слоја:

- Апликацијски слој (Application layer),
- Апликацијски оквир (Application Framework),
- Библиотеке (Libraries),
- Linux kernel.

Од 2008. године па до данас Андроид је имао бројне надоградње које су побољшавале рад оператвног система, додајући нове карактеристике и уклањајући грешке претходних

верзија. Свака већа промена подразумевала је и нову верзију система. Верзије су добијале име по слаткишима (Vincent, 2015).



Слика 1. Слојеви “Android OS”
Извор: https://ars.els-cdn.com/content/image/1-s2.0-S1742287610000381-gr1.jpg

Треба напоменути да је андроид далеко најприлагодљивији оперативни систем за мобилне уређаје. Произвођачи мобилних уређаја као и провајдери мобилних услуга имају могућност прилагођавања оперативниог система у зависности од њихових потреба и потреба њихових корисника, а све у сврху остварења предности на тржишту у односу на конкуренте. То се најчешће постиже додавањем нових функција, апликација или сервиса, као и променом визуелног идентитета корисничког окружења. Андроид оперативни систем је најбољи оперативни систем за мобилне уређајена светском тржишту, са преко 1.4 милијарде корисника (Vincent, 2015). Константим и учесталим развојем, унапређивањем, циљ андроида је да се што више прилагоди потребама крајњег корисника.У данашњем свету, где је живот постао незамислив без мобилног уређаја, јасно је колика је улога квалитетног оперативног система. Сваким даном се активира преко 1.500.000 нових корисника андроида из чега се може закључити да андроид има блиставу будућност. Пружајући надоградњу сваких пола године, сваки пут са бољим перформансама, пратећи константан развој хардвера, андроид је несумњиво водећа платформа за мобилне уређаје на свету.

2.1.2. Верзије Андроид оперативног система

Нове верзије Андроид-а се углавном како је време показало појављују једном годишње, али како је систем отвореног кода, који на својим уређајима примењује велики број произвођача. Те уређаје затим дистрибуира велики број добављача, провајдера мобилних мрежа и других, прихватање нових верзија система је јако спор процес, који се често добија куповином новог уређаја, а не надоградњом старог. Сходно реченом у табели испод су приказане верзије система, верзија API кода, од којег зависе прва три слоја

приказана на слици 1. У последњој колони је представљн, процентуално, број уређаја који се активно користе и имају дату верзију система. Када се генерише, а касније програмира пројекат у андроид студију, потребно је дефинисати минималну верзију андроид оперативног система, коју дата апликација задовољава (Pulungman, 2013).

Табела 1: Верзија Android OS (Android, 2019)

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%

Сходно потребама апликације, и тржишта које обухвата потребно је дефинисати одговарајућу минималну верзију система за коју апликација има подршку. Разлика у API је начин програмирања одређених ствари (Android, 2019).

2.2. Django Web framework

“Django” је бесплатан веб frameworkотвореног кода, настао 2003. године, развијен у Лавренц од стране искусних веб програмера Адријан Холовати и Сајмон Вилисон, написаног у пајтону који се користи за креирање сложених веб сајтова који се базирају на употреби база података. Код у пајтону прати “MVC” архитектуру која се састоји од поновног коришћења софтверског кода, олакшавању развоја и одржавања веб апликације.”Django” је своје име добио по чувеном џез гитаристи “Django Reinhardt” (Holovaty, 2007).

2.2.1. Шта је веб фрејмворк?

Веб frameworkје софтверски frameworkкоји је дизајниран да помогне у развоју веб апликација укључујући веб сервисе, веб ресурсе и веб “API”-је. Пружају стандардан начин израде и постављање веб апликација на интернет. Циљ веб фрамефоркова је да олакшају оптерећења која су повезана са честим активностима у развоју веб апликација. Многи фрејмворкови обезбеђују библиотеке за комуникацију са базама података, као и могућност управљања сесијама. Често се користе код динамичких сајтова, али постоји могућност да се користе и на статичким (WEB framework, 2014).

2.2.2. Компоненте Djanga

Иако је имао сопствени начин за давање имена, као што су опозиви објекти који генеришу “HTTP” одговоре „прегледа“, “Django” frameworkможе се посматрати као “MVC”. Између модела података и релационих база података, посредује ОРМ (објектно-релациони мапер), то је систем за обраду “HTTP” захтеве са системом веб шаблона и диспечера веб адреса регуларног израза (WEB framework, 2014).

Фејмворк такође садржи:

- Лак и самосталан веб сервер за тестирање и развој веб апликација
- Систем форми које могу да преводе између “HTML” форми и вредности погодних за складиштење у базу података
- Систем шаблона који уомогућава принцип наслеђа позамљен од објектно-орјентисаног програмирања
- Могућност коришћења било којег начина кеширања помоћу frameworkвеб кеша
- Подршку за класе посебног софтвера који може да извршава ручно нAPIсане функције и да интервенише током фаза обраде захтева
- Систем унутрашњег диспечера који омогућава комуникацију између компонената апликације преко предходно дефинисаних сигнала
- Могућност превода “Django” компоненти у друге језике помоћу система интернационализације и локализације
- Систем серилизације који подржава читање и креирање флајлова у формату XML и/или JSON
- Систем за проширивање могућности шаблона
- Интерфејс до пајтоновог уграђеног фрејмворка за тестирање

При креирању веб апликације, “Django” нуди опциони административни интерфаце који је генерисан динамично кроз самопосматрање и који је подешен у моделу администратора. Такође има као опцију проверу корисника, управљања садржајем, аутоматско преузимање садржаја са веба. За разлику од “PHP”, “Django” има висок ниво жаштите и његов систем провере корисника омогућава сигуран начин за управљање над базом података. “Django” при инсталацији долази са уграђеним апликацијама као што су:

- Алати за генерисање “RSS” формата

- Можемо покренути више цајтова одједном и да сваки има свој садржај
- Алате за генерисање “Google Sitemaps”
- Одбрана од познатих хакерских напада
- frameworkза прављење ГИС апликација

“Django” је подељен на модуле који функционишу и оврађују податке самостално и комуницирају међусобно само по потреби или наредби. Модули чије једну моћну и брзу целину која чини “Django” frameworktако ефикасним и популарним.

Модели

Модел је објекат који описује неку таблицу података а сваки атрибут једно поље. Уместо директног рада са базом података, упити се извршавају позивањем методе објеката. “Django” аутоматски додаје речник у коме је наведено која се база података користи и аутоматски га ажурира позивом макемиграте и миграте наредбе. Модели се налазе у датотеци “models.py” која се налази у директоријуму апликације (Ервин, 2016).

“***URLs” и “Views”***

“URLs” је путања до одређеног садржаја на интернету која се још назива и веб адреса. Да бисмо имали квалитетну и функциоалну веб апликацију, “Django” нам омогућава брзо и лако креирање прегледног урл линка који нема суфикса. “Views” у ђангу веб фрејмворку предстаља пајтон функцију која узима веб захтев и враћа веб одговор (Ервин, 2016).

Форме

“Django” поседује велику библиотеку форми које раде са форматима као што су “HTML”, проверавајући корисничке информације и захтеве и шаљу повратне информације натраг њима. Такође преко библиотека можемо генерисати форме из постојећих модела и атрибута (Django, 2019). О свим компонентама Ђанга, детаљније у поглављу 8.

2.2.3. Могућност проширивања

Дозвољава да се већ постојећи код имплементира у тренутни пројекат, под условом да прати конвенције кода који се поново користи (Django, 2019). Постоји висе 2500 пакета за проширавање оригиналног фрејмворка, тако да је могуће обезбедити додатне функције које основни пакет не подржава, као што су: регистрација, претрага, “API” одредба и потрошња, “CMS” (Django, 2019).

Међутим, могућност проширења је смањена тиме што постоји интерна зависност компоненти. Ниједан од ових филтера или уграђених апликација није обавезна за покретање “Django” пројекта, али поновно коришћене апликације у већини случајева зависе од њих, тако охрабрују програмере да наставе са коришћењем оригиналних пакета да би имали све бенефиције од екосистема апликације (Django, 2019).

2.2.4. “MVC” образац

Многи фрајмворкови користе “MVC” архитектонски образац да би одвојили модел података са пословним правилима од корисничког интерфејса. Ово је добра пракса пошто се постиже модуларизација кода, постиче се поновно коришћење кода, и омогућује да се примени више интерфејса. У веб апликацији то омогућава да се прилажу различити погледи, као што су веб стране за људе и веб сервиси за апликације (Trygve et al., 2009).

2.3. Бар-код і “QR Code”

У овом поглављу ћемо детаљно објаснити принцип рада кодовања бројева или текста, начин примене и теоријске аспекте Bar-koda і “QR Code”-а.

2.3.1. Бар-код

Баркод представља другачији начин кодирања бројева и слова користећи комбинацију танких линија различитих ширина и размака између њих. Ово је само другачији начин за уношење података у компјутер. Баркод не садржи описне податке. Он је референтни број који компјутер користи да би нашао њему додељени зAPIс који садржи описне податке и остале важне информације. На пример, баркод који се налази на неком производу не може садржати име производа, тип производа, или цену, уместо тога он садржи дванаестоцифрени број производа.

Зашто се бар-код користи?

Коришћењем Баркод технологије у продавницама знатно упрошћава послове, као што су мењање цена прозивода, праћење продаје производа, скраћује процес пописа производа који су на стању, проверу украдених производа и друго. Бар код технологија омогућава да централизовану базу, која прати производе, њихове цене и залихе држите на компјутерском систему. Цене се могу мењати често, без потребе да се стављају нове ознаке цена на све производе. Такође се одмах може видети када се залихе одређених производа смањују, и када је потребно набавити још производа. Бар код технологија је веома поуздана и прецизна, да потврђује сумњу да су производи који недостају (а нису продати) украдени, тако да се ти производи премештају у сигурнији део продавнице или се заштићују заштитним таговима. Систем залиха базиран на бар коду има три главна дела. Први, постоји централни рачунар који покреће базу података која чува све производе који се продају, ко их прави, колико коштају и колико их има на лагеру. Други, бар кодови се налазе на свим производима. И трећи део је да постоји један или више скенера који могу да читају баркодове (Barkod, 2009).

Слика 2. Баркод *(Лерво),* кодовање баркод-а *(десно)* (Barkod, 2009)

Баркодови представљају бројеве од 0-9

Сваки производ има свој јединствени код, или број. Свака цифра у броју производа добија исту количину хоризонталног простора, тачно 7 јединица. Затим, да би се представио било који број од 0 до 9, обоји се тих седам јединица са различитим обрасцима црних и белих линија. Дакле, број један је представљен бо у две беле линије, две црне линије, две беле линије и једну црну линију, док је број два представљен са две беле линије, једном црном линијом, две беле линије и две коначне црне линије. Баркодови могу бити прилично дугачки и то је зато што морају представљати три различите врсте информација. Први део баркода садржи информацију о земљи у којој је производ издат. Други део открива произвођача производа, и трећи идентификује сам производ (Barkod, 2009).

2.3.2. QR Code

QR код се може користити на мобилним уређајима за читање података. QR је скраћеница од Qуицк Респонсе, што представља брз приступ информацијама у коду. QR код користи опен соурце технологију. Имају већи капацитет података у односу на обичне баркодове. QR код се у свом поцетку користио за праћење делова у производњи возила, међутим његова примена је постала све распрострањенија. Данас се QR код користи у тв рекламама, на производима, на визит картама,а помоћу њега можемо предефинисати СМС поруку, додати све податке са визит карте директно у телефон, прочитати детаљније инфорамције о неком производу, а такође нам омогућава и хуперлинк ка сајтовима.

Слика 3. Структура QR кода (Tan, 2008)

QR код је симбол типа матрице са структуром ћелија распоређеном у квадрат. Састоји се од функционалних шаблона за једноставно читање и области података у којој се подаци чувају. QR код има обрасце за проналажење, обрасце поравнања, узорке времена и мирну зону. Како би скенер препознао QR код, сам код односно црни и бели пиксели морају увек бити квадратног облика.

Ознаке положаја

Указују на смер у ком се код штампа. Узорак за откривање положаја QR кода. Постављањем овог узорка на три угла симбола може се детектовати положај, величина и угао симбола. Овај узорак проналаска се састоји од структуре која се може детектовати у свим правцима (360°) (Tan, 2008).

Ознаке поравнања

Ако је QR код велики, ознаке поравнања помажу при оријентацији. Образац за исправљање искривљења QR кода је веома ефикасан за исправљање нелинеарних дисторзија. Централна координата обрасца за поравнање биће идентификована да би се исправило изобличење симбола. У ту сврху, црно изолована ћелија се налази у шаблону поравнања како би се лакше детектовала централна координата поравнања (Tan, 2008).

“Timing pattern”

Користећи ове линије, скенер одређује величину података. Узорак за идентификацију централне координате сваке ћелије у QR коду са црно-белим обрасцима распоређеним наизменично. Користи се за корекцију централне координате ћелије података када је симбол

изобличен или када постоји грешка за висину ћелије. Распоређен је у вертикалном и хоризонталном правцу.

Верзија информација

Одређује која се верзија QR кода користи. Постоји 40 различитих верзија.

Формат информација

Садржи информације о грешкама и једноставније се скенира код.

Кључеви за исправљање података и грешака

Овај патерн садржи саме податке.

Мирна зона

Овај размак је важан за програм скенирања како би се разликовао QR код од његовог окружења. Маргинални простор потребан за читање QR кода. Ова тиха зона олакшава проналажење симбола из слике коју чита ЦЦД сензор. Четири или више ћелија су потребне за мирну зону (Tan, 2008).

2.3.3. Спецификације QR кода

Ранија верзија дводимензионалног баркод система мерила је само 21 x 21 модул и садржавала је само четири карактера података, а највећа тренутна верзија (40) се простире на 177 x 177 модула и може садржати 1264 знакова ASCII текста или до 7,089 знакова. Подаци могу бити кодирани различитим методама, а различити типови кодирања могу се користити у истом QR коду (Struktura QR, 2019):

- “Numeric” (10 битова по цифри)
- “Alphanumeric” (11 битова на 2 знака, али не може складиштити мала слова)
- “Byte” (8 битова по карактеру)
- “Kana” (13 битова по карактеру)

Specifications		
Smallest symbol size	21 x 21 modules	
Largest symbol size	177 x 177 modules	
Maximum data capacity	Numeric	7089 characters
	Alphanumeric	4296 characters
	Kanji	1817 characters

Слика 4. Спецификације QR код (Struktura QR, 2019)

Примена QR кода

QR код је доста једноставнији у односу на баркод. Потребно је скенирати QR код са сензором слике, као сто је камера, апликација це претворити код у бинарни, а затим ће приказати информације или ће извршити програмирану акцију, као што је отварање веб сајта. Постоји огроман број апликација за читање QR кода, а неке познате апликације имају интегрисане секенер за специфичне карактеристике.

QR код се састоји од низа квадрата, од којих се неки користе за позиционирање сензора слике (то су велики квадрати на три угла), док остале ћелије садрже информације о верзији и формат. QR кодови су дводимензионални у односу на једнодимензионалне баркодове И омогућавају скенирање из било ког правца уместо само једног. Они могу да садрже хиљаде алфанумеричких знакова. Што више грешака QR код има, то мање података може складиштити, што је више података складиштено у QR коду, то ће више квадрата он имати. Већи број квадрата је потребан јер се ниво корекције грешака повећава.

2.4. REST и SQLite

REST је скраћеница за Репресентатионал Стате Трансфер који се у суштини односи на стил web архитектуре који има много подкарактеристика и управља понашањем клијената и сервера. REST API вам дозвољава да сарађујете са PARSOM са било чега сто може да пошаље HTTP захтев (Masse, 2012).

REST API у стварном свету (Microservices, 2019):

- Twiter REST API
- Facebook REST API
- Google translate REST API
- Flickr REST API

- Dropbox REST API
- Ebay developer REST API
- BING Maps REST API
- BING saobracajne nesrece API
- Magento REST API

2.4.1. RESTful API

Имати REST-API прави web сервис "RESTful" (Војиновић, 2012). RESTful API — такође знан као RESTful web служба - базирана је на Репресентатионал Стате Трансфер (REST) технологији и архитектуралном стилу и прилазу комуникацијама често коришћеним у разради web служби (Masse, 2012).

REST технологија се више преферира у односу на робуснију Симпле Објецт Аццесс Протоцол (СОАП) технологију јер REST нивелише мањи проток, чинећи је прилагођенијом за употребу на интернету. API за веб сајт је код који дозвољава да два софтвера програма комуницирају један са другим. API програмеру објашњава прави начин којим он треба да напише програм који тражи захтеве од оперативног система или неке друге апликације (Masse, 2012). REST који користе претраживачи може бити гледан као језик интернета. Са порастом употребе Cloud-a,API се користе за откривање WEB услузи. REST је логичан избор з градњу API-ја који дозвољава кориснику да се прикључи и ради са цлоуд службама. RESTful RESTкористе сајтови као што су: Amazon, Google, LinkedIn, Twitter и други (Masse, 2012).

Прилагођавајући се ограничењима REST-а генерално се на то гледа као „RESTful". REST се може сматрати „RESTful", ако има следеце карактеристике (главне карактеристике):

- КЛИЈЕНТ-СЕРВЕР: клијент рукује предњим делом,а сервер задњим и оба могу бити замењена независно једно од другог
- STATELESS: никакви подаци клијента нису складиштени на сервер између захтева,а стање сесије се складишти на клијента
- CACHEABLE: клијент може да сакупља одговоре(као претраживач који сакупља статичне елементе web странице) да би побољшао свој перформанс (Masse, 2012).

Слагајући се са овим ограничењима и тако правећи REST архитектурални стил омогућује било коју врсту дистрибуираних хипермедијских система да имају жељене спајајуће особине, као што су перформанс, скаларност, једноставност, видљивост, преносивост и поузданост. Уобичајени тренд је да се користе апсолутни URL за RESTful API-је који су значајни (Masse, 2012).

„RESTful" клијент-сервер архитектура

HTTP, на пример, има јако богат речник кад су у питању глаголи (тј „методи"), URL, типови интернет медија, кодова за захтеве и одговоре итд. REST користи ове постојеће

особине HTTP протокола и тако дозвољава постојећем слојевитом заступнику и пролазу компонената да извршава додатне функције на мрежи као што су HTTP складиштење и појачавање осигурања (Masse, 2012).

RESTful web usluge (API)

RESTful web услуга (такође звана RESTful web API) је web услуга имплементирана коришћењем HTTP-а и принципима REST-а. То је колекција ресурса са четири дефинисана аспекта: база URL за web службе, као сто је http://example.com/recources/ интернет медија тип података подржана од стране web службе. То је најчешће JSON, XML или YAML, али може бити и било који други валидни тип интернет медија уколико има валидни хипертекст стандард, сет операција подржан од стране web услуга које користе HTTP методе (GET, PUT, POST ili DELETE). API мора бити покренут хипертекстом (Masse, 2012).

Популарни REST API формати за захтеве:

- REST
- XML-RPC
- SOAP (Microservices, 2019)

Популарни REST API формати за одговоре:

- REST
- XML-RPC
- SOAP
- JSON
- PHP (Microservices, 2019)

2.4.2. Начин функционисања RESTful API-ja

RESTful API раставља трансакцију да би направила серију малих модула. Сваки модул обрађа се посебном поделу трансакције. Ова модуларност пружа програмеру доста флексибилности, али за програмера може бити изазов да то дизајнирају од почетка. Тренутно, модели које пружају: Amazon Simple Storage Service, Cloud Data Management Interface и OpenStack Swift су најпопуларнији. RESTful API изричито даје предност HTTP методологијама дефинисаним RFC 2616 протоколом. Они користе GET да добију ресурс; PUT да промене стање или ажурирају ресурс, који може бити објекат, фајл или блок; POST да генеришу тај ресурс; и DELETE да га уклоне (Masse, 2012).

Зато што су позиви без стања, REST је употребљив на cloud апликацијама. Компоненте без стања се могу слободно поново искористити ако нешто не успе, и могу се скалирати да би се прилагодиле промени садржаја. Ово је зато сто се било који захтев може послати било којој инстанци компонента; ништа не треба да се чува да би га запамтила следећа трансакција. Ово чини REST најбољим за web употребу, али је RESTful модел такође од помоћи у cloud-у службама, јер везивање службе кроз API је начин контроле декодирања URLa. Cloud решења и микротрансакције ће скоро сигурно RESTful API направити у правило будућности (Masse, 2012).

2.4.3. SQLite

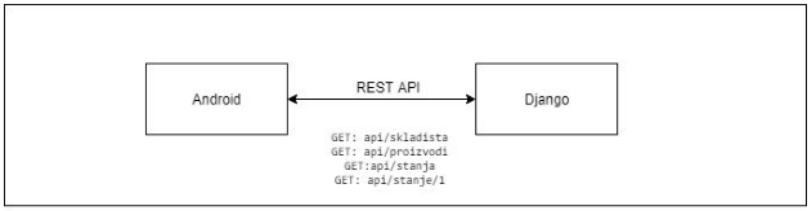
SQLite је библиотека која се имплементира самостално, без сервера, нулта конфигурација, трансакцијски SQL базе података. Код за SQLite је у јавном домену и стога је слободан за употребу у било коју сврху, комерцијалну или приватну. SQLite је најшире распоређена база података на свету, укључујући и неколико пројеката високог профила. SQLite је уграђена SQL база података. За разлику од већине других SQL база података, SQLite нема посебан сервер. SQLite чита и пише директно у фајлове који се налазе на диску. Комплетна SQL база са више табела, индекса, окидача и приказа, садржана је у једној датотеци диска. Датотека базе података је cross-platforma, можете слободно копирати базу података између 32-битних и 64-битних система или између великих. Ове карактеристике чине SQLite популарним избором као формат датотеке апликација. Датотеке базе података SQLite су препоручени формат за складиштење од стране US Librari of Congress (SQLite, 2019).

Једина мана SQLite-а је ограничење у броју корисника, који у истом моменту, могу приступити самој бази. Сходно томе да је база уствари филе базирана база, само један корисник може приступити бази у једном моменту. База се при приступању корисника закључава и друг корисници нису у могућности да је мењају. У зависности од потребе датог система ово може бити ограничавајући фактор. У нашем случају то није, јер бази приступа један сервер, и база није дистрибурана, сходно датом испуњавању свих предуслова, као база у Django пројекту, примењен је SQLite.

3. Архитектура и имплементација

У увом поглављу је описана архитектура решења, заједно са детаљном комуникацијом између елемената система. Такође описана је детаљна структура целог система. Циљ је да се након читања поглавља, читалац припреми за даље разумевање целог система, као и да добије теоријску основу целе функционалности система.

Као прво представљена је основна идеја архитектуре система, и његовог основног функционисања. На слици 5, је приказана блок шема система. На слици 5 су приказана два основна дела система, под-системи, који сачињавају основну структуру система, овог решења. То су Android и Django. Андроид апликација представља „front-end“, кориснички део система, док Django представља пословну логику система, „back-end“. Раније у поглављима 2 и 3 су детаљније описани концепти које представљају Android и Django.



Слика 5. Блок шема система

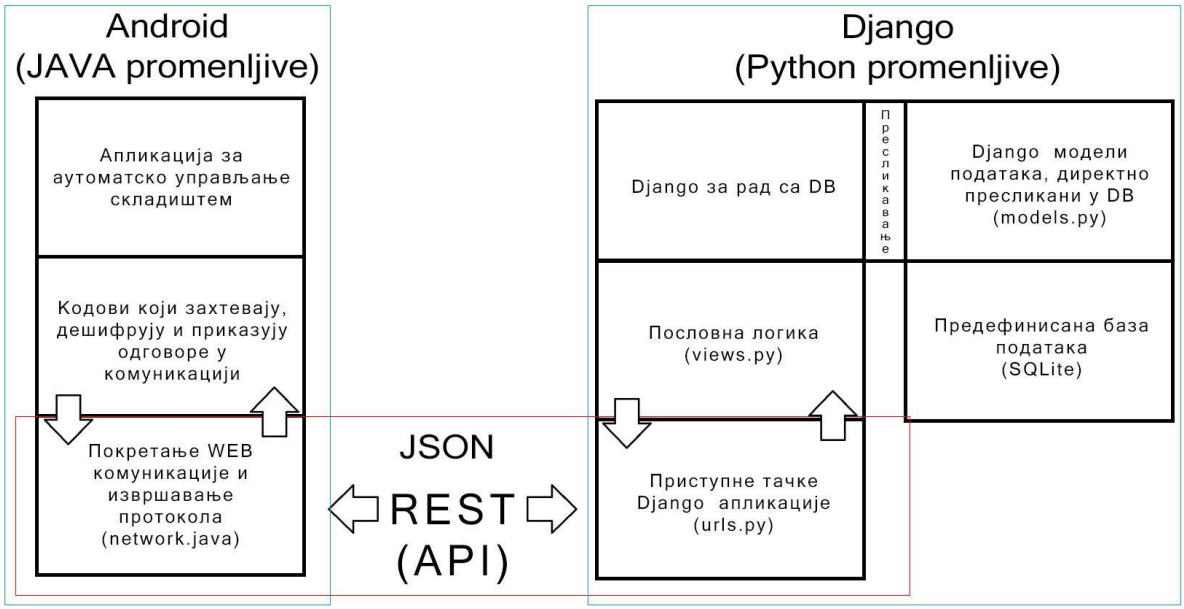
Слика је апстрактна блок шема функционалних делова система, са основним приказом функционисања система. На слици је апстрактно приказана и комуникација међу компонентама система, у питању је REST API, са апстрактним примером комуникације. У поглављу 5 је објашњен концепт REST API-ја, док је даље у раду детаљније опсиан протокол комуникације међу компонентама система.

3.1. Комуникација у систему

Као што је речено у поглављу 6, систем се састоји из две целине, два основна дела. Начин комуникације и преноса инфорамција у ове две целине и између ове две целине, се састоји из три различита начина. Тако да се систем може посматрати, са аспекта комуникације, као три засебне целине (Wang et al., 2019). На слици 6, је приказан процес комуникације, заједно са битним компонентама система, које имају директан или индиректан утицај на обављање комуникаце по раније дефинисаном протоколу комуникације. Сходно томе даље у тексту су објашњене ове три целине.

3.1.1. Комуникација у андроид делу система

На слици 6. андроид део система (лево на слици), издељен је на три основне целине, који представља апстрактни приказ андроид дела система, апстракција са становишта комуникације. Ако кренемо од врха, први део је сама апликација, што апстрактно представља свеукупну функционалност андроид дела система, без посебно издвојених комуникационих компоненти. У позадини дате апликације се налазе кодови задужени за подношење захтева, као и за превођење одговора добијених са стране сервера, и коначно представљања добијених одговора кориснику (кроз саму апликацију, највишљи слој). На дну се налзи део система, који уствари обавља послове комуникације, пријема података и превођење истих у одговарајућу форму. Целокупан процес комуникације се обавља кроз променљиве, у случају андроид дела, који је испрограмиран у JAVA програмском језику, променљиве JAVA програмског језика. Што се тиче даљег тока комуникације, обавља се путем интернет канала, а сходно потреби општег преноса података, и могућности комуникације са свим системима на другој страни комуникације, JAVA променљиве се преводе у „JSON“ формат. Са стране андроида, JAVA програмског језика, за овај задатак је задужена библиотека „GSON“, која је детаљније описана у поглављу опис решења (опис самог кода апликације). За сада је довољно знати да је упитању директна трансформација променљивих у „JSON“ код.



Слика 6. Комуникација система

3.1.2. Django, python комуникација

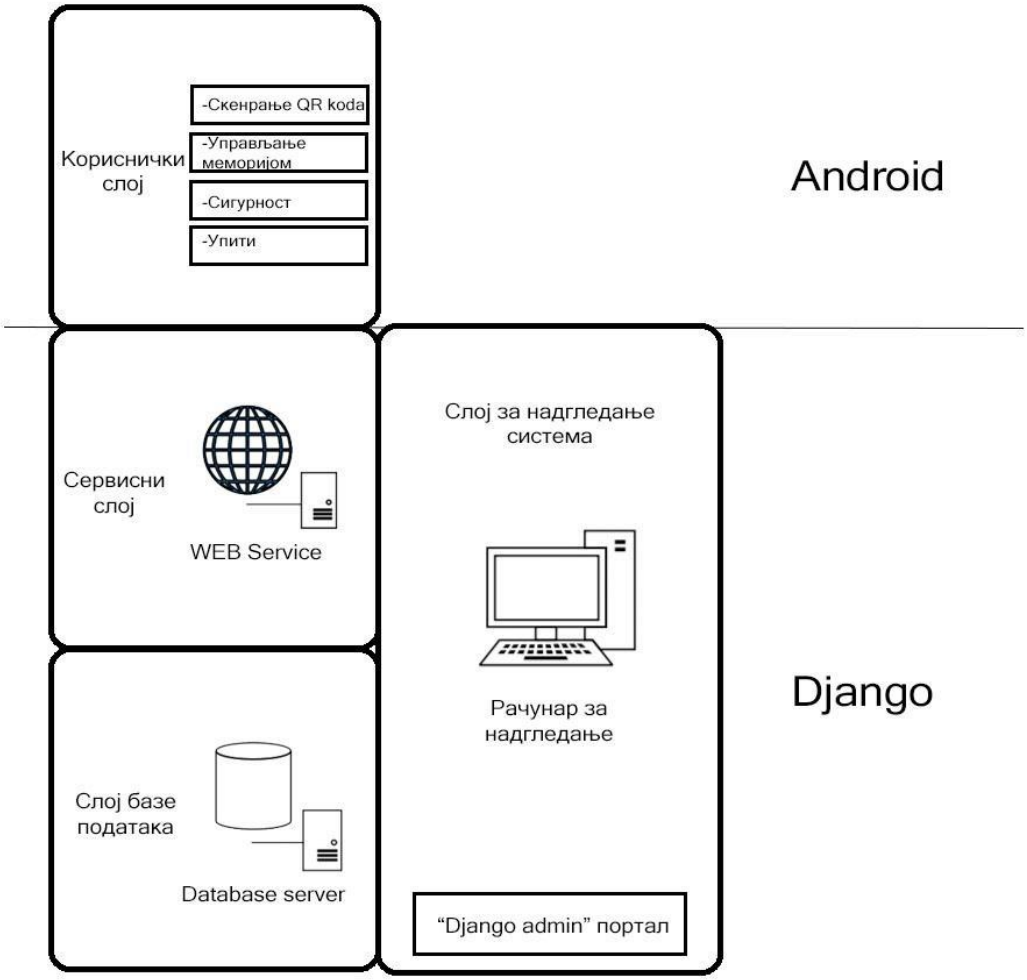
Када је у питању Django пројекат, и комуникација у том делу система, комуникација се обавља стандардним каналима за Django пројекат. Целокупна комуникација се обавља кроз „python“ променљиве. Податак који пристиже серверу је у JSON формату, директном трансформацијом истог, добија се „python“ речник, што је врста „python“ променљиве. На слици 6, одоздо гледајући прва компонента у делу пројекта за који је задужен Django, приказане су приступне тачке, што на један начин и јесте приступни канал Django апликације. Након приступа одређеној тачки, позива се одговарајући код у пословној логици Django апликације („view“, „python“ класа са одговарајућим методама). За остатак комуникације у Django делу пројекта, је задужен сам framework Django. Укратко, Django врши пресликавање раније дефинисаних модела података, директно у базу која је раније предефинисана у подешавањима пројекта, у случају овог пројекта у питању је SQLite база података. Сам процес чувања, пресликавања и осталих трансформамција које ради сам framework Django нису тема овог рада.

3.1.3. Комуникације две главне компоненте система

Што се тиче комуникације између андроид и Django дела система, овај део комуникације се обавља једноставним REST API позивима, сам стандард комуникације је описан у поглављу 5. У питању су једноставни позиви, при чему се подаци преносе у JSON формату. Целокупан процес комуникације је преко интернета.

3.2. Структура система

Структура система се може представити кроз апстрактну поделу система на слојеве. Основна подела сваког система може бити на слој који корисници виде и користе и на слој који обавља послове које корисник задаје. Као додатна подела из ове основне, могуће је проширити на три слоја, као код трослојне архитектуре. У том случају издваја се база података и сервер базе података.



Слика 7. Функционална структура система

Слојеви на које се дели овај систем, подела која је приказана на слици 7:

- **Кориснички слој**: Андроид део система за аутоматско управљање складиштем, је слој који даје опције за рад са складиштем и у продавницама. Даје могућности кориснику кроз једноставан кориснички интерфејс контролисано управљање подацима, задавање команди систему. Уз то даје могућности скенирања QR кодова, управљање сигурним деловима система, слање упита слоју испод, као и дешифровање комуникације, приказ одговора добијених од сервисног слоја

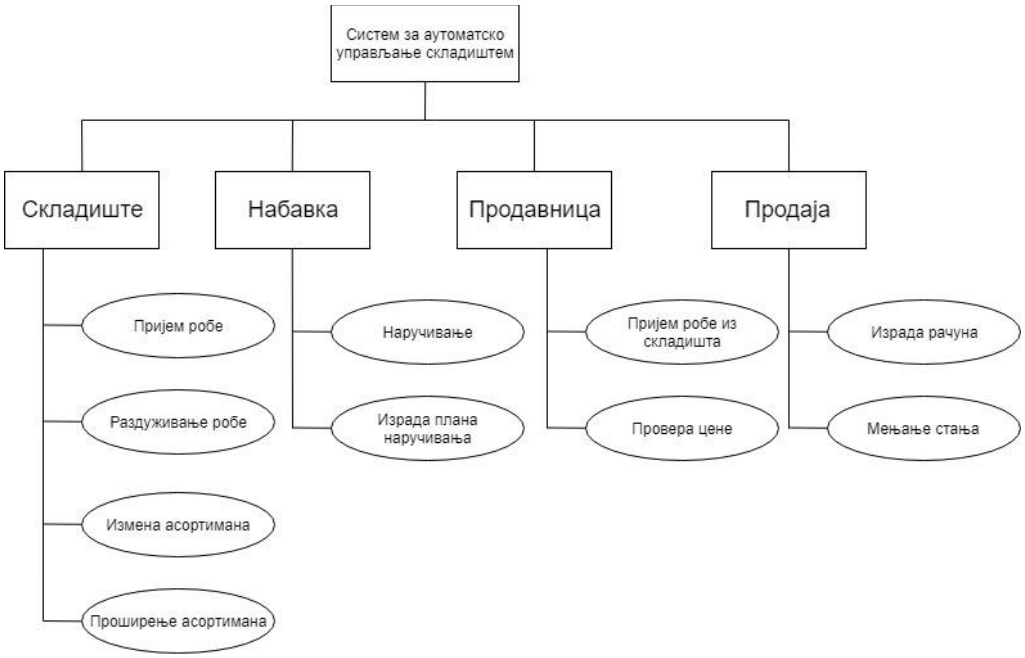
- **Сервисни слој**: Сервисни слој применом WEB Service технологије обезбеђује комуникацију андроид апликације са слојем базе података. Овај слој је имлементиран кроз "Django framework", и применом свих принципа REST API стандарда протокола. Овај слој је индиректно задужен за функционисање сигурносних компоненти система. Саму сигурност обезбеђује слој изнад, чување података о кориснику и друто, али на захтев корисничког слоја, сервисни слој одговара са одговарајућим подацима о кориснику апликације.
- **Слој базе података**: Слој базе података се налзи на самом дну система. Обезбеђује рад над подацима за терминале којима се приступа систему, андроид апликације или путем рачунара. У систему који је описан у раду, овај слој је интегрисан заједно са сервисним слојем, део је Django framework-a. У описаном систему база је дефинисана као SQLite база. Да је било која друга технологија примењена, нема разлике у иплементацији, само је потребно обезбедити одговарајућћу инфраструктуру сервера.
- **Слој за надгледање система**: Је слој који се налази паралелно уз цео систем, сврха слоја је да обезбеди проверу рада система, као и контролу над подацима и изменама истих. У систему који се описује и овај слој је интегрисан уз претходна два слоја. Представља Django WEB admin портал. За потребе оваквог система није било сврхе дефинисати и ширити дати слој, док за комерцијалну примену сличног система постоји потреба за проширењем овог слоја. За глобалнију примену система овај слој би био проширен са сервером за логовање грешака, праћењем промена података и верзија кода. Као и за тестирање стабилоности система.

3.3. Функционална подела система

Систем се може поделити и на функције које обавља. Поделом система за аутоматско управљање складиштем опсианом у овом раду, добијају се четири основне функцијоналне целине, од којих свака извршава одређене задатке. Подела је приказана на слици 8, а функције по којима је систем подељен су:

- **Складиште**: функције система које се извршавају за потребе складишта. Магационер има приступ овуим функцијама, а то су: Пријем робе у складиште од добављача, раздуживање робе - слање исте у продавницу, измену постојећег и проширивање асортимана додавањем нових производа, односно изменом постојоћих.
- **Набавка**: Овај део система нема имплементиран интерфејс, јер није тема изучавања овог рада, али потребна инфраструктура за извршавање наведених функција постоји, табеле у бази, потребно је израдити генерисање извештаја и израда потребног интерфејса за обављање послова набавке.
- **Продавница и продаја**: су једна целина система, која је у подели на слици 8, издељена на две под-целине ради лакшег разумевања система. Послове који се обављају у продавници, пријем робе из складишта и провера цене одређеног производа (на захтев

купца, ову функцију и сам купац без приступа систему може да обави). Док функције продаје обавља продавац (исто као код фунцкија продавнице), и то су израда рачуна, односно продаја робе и мењање стања, на основу те продаје.



Слика 8. Функционална подела система

4. Примена и студија случаја

Систем је имплементиран као две основне целине, андроид апликација и Django сервер. Андроид део система је имплементиран у "android studio" развојном окружењу, док је тестирање рада апликације обављано на стварном андроид телефону, не на емулатору. Због потребе коришћења камере ради скенирања QR кодова, није било могуће користити емулацију уређаја. Уз коришћење окружења, као и основних кодова које андроид платформа обезбеђује, коришћене су две додатне библиотеке. "ZXing" (GIT zxing, 2019) и "GSON" (GIT gson, 2019) библиотеке су отвореног кода. Прва (ZXing) је библиотека која олакшава имплементацију скенирања QR и Bar-code кодова. Написана је у програмском језику Java, и оптимизована за рад на андроид пројектима, чиме је била погодна за примену у пројекту. "GSON" библиотека је имплементирана од стране Google-а и служи превођењу Java објеката у JSON објекте. Само примена ове две библиотеке је описана у поглављу 8. За праћење и детектовање грешака коришћен је подразумевани "debugger" у андроид студију.

"Django" пројекат је имплементиран у "PyCharm" окружењу, док је за потребе тестирања коришћен тест сервер који је подразумеван у "Django" окружењу. Целокупан процес тестирања и покретања пројекта је у локалној рачунарској мрежи. Део који се односи на базу података је интегрисан у овом делу пројекта, и потпуна контрола је на страни "Django" интерног кода, никаквих додатних подешавања или покретања додатног

сервера нису потребна. Сходно томе да се као база података користи SQLite, база података „у фајлу“. Сама структура базе података је дефинисана кроз “Django model-e”, детаљна структура базе је описана у под-поглављу 7.1. Док су сами модели из Django пројекта описани у поглављу 8.

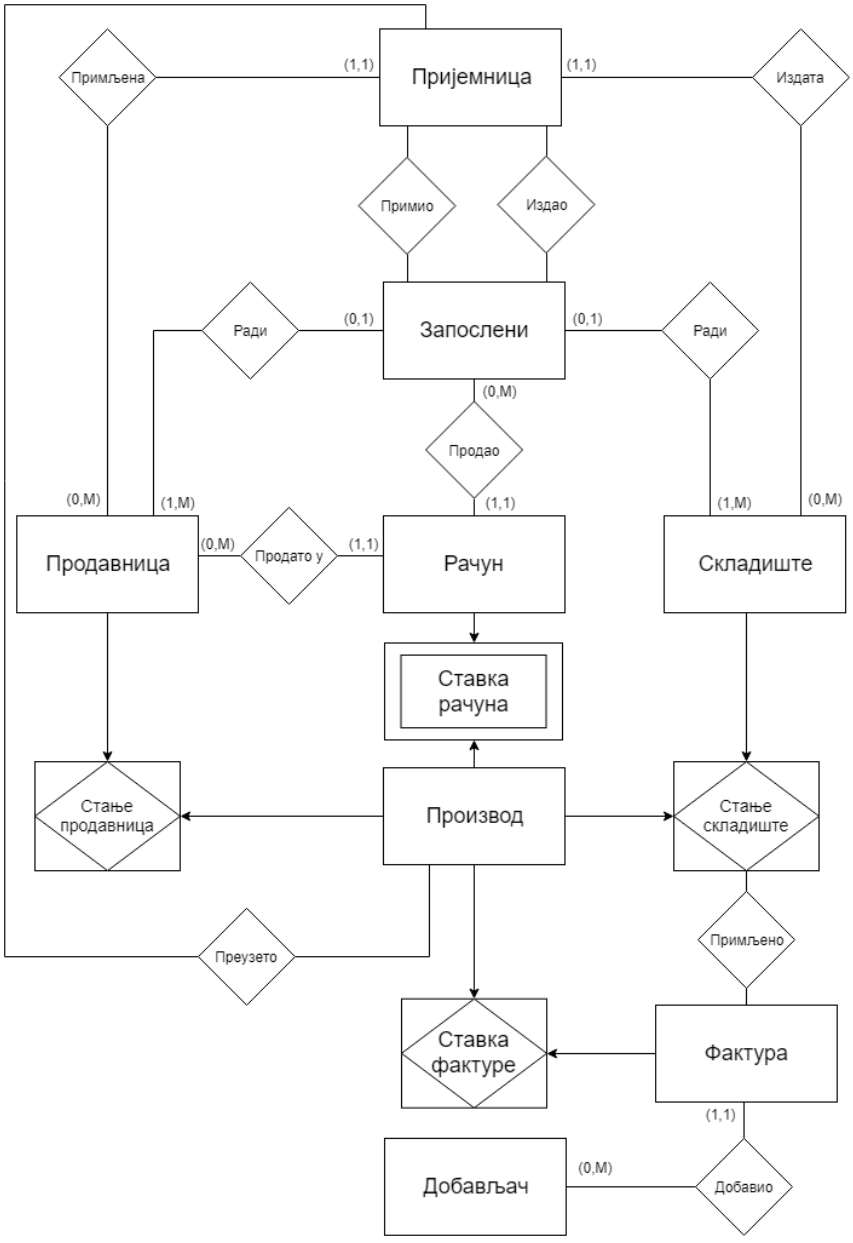
4.1. Имплементација базе података

Као што је већ напоменуто у раду, база података се имплементира индиректним путем помоћу Django model-a, на основу којих се аутоматски генеришу табеле у самој бази, одговарајуће технологије, у овом случају SQLite табеле. Тако да се структура базе диктира помоћу “python” кода. На слици 9 је представљена структура базе података, њене табеле, као везе и нотације веза тих табела. Поља табела нису приказана на слици, а у циљу мање компликоване слике.

На слици се види 12 табела, од тога 3 табеле представљају агрегацију, а једна табела слаб објекат, одностно директно је зависна од друге табеле. База података обухвата следеће табеле:

- **Добављач**: У овој табели се чувају подаци о добављачима робе за дату продавницу, и имају везу само према табели фактура, за сваку фактуру улазне робе мора да постоји добављач од којег роба пристиже.
- **Фактура**: Табела која садржи податке о пристиглој роби, од одређеног добављача, дата роба поставља/мења стање у одређеном складишту, и на фактури се налази списак пристигле робе, табела ставка фактуре.
- **Ставка фактуре**: Списак производа, по одређеној фактури и одређена количина тих производа.
- **Производ**: Табела у којој се налазе подаци за сваки производ у систему. Активни, стари производи, у продаји неке продавнице или у неком складишту.
- **Продавница**: Табела у којој се воде све продавнице у систему. Поред ове табеле постоји и табела “**Стање продавница**”, табела која повезује производ и продавницу и чува количину датог производа у датој продавници.
- **Складиште**: Табела у којој се воде сва складишта у систему. За свако складиште води се табела “**Стање складиште**”, где се води стање за одређени производ у датом складишту, поред овога за сваку промену стања обавезно је вођење фактуре по којој долази до промене стања.
- **Пријемница**: Табела у којој се подаци генеришу приликом транспорта робе из складишта у продавницу. Тако да табела садржи податке о запосеном раднику у складишту, самом складишту из којег полази роба, затим податке о производу и количини. Приликом пријема робе у продавници, табела се допуњује подацима о запосленом у продавници и самој продавници у којој је примљена роба.
- **Запослени**: Табела о свим запосленима у систему, сваки запослени припада или складишту или продавници. Један запослени не може да ради на два радна места у исто време.

- **Рачун**: Приликом продаје робе у продавници, генеришу се подаци о продаји, одређени продавац издаје дати рачун.
- **Ставка рачуна**: За сваки рачун се води списак продатих производа, са продатом количином.



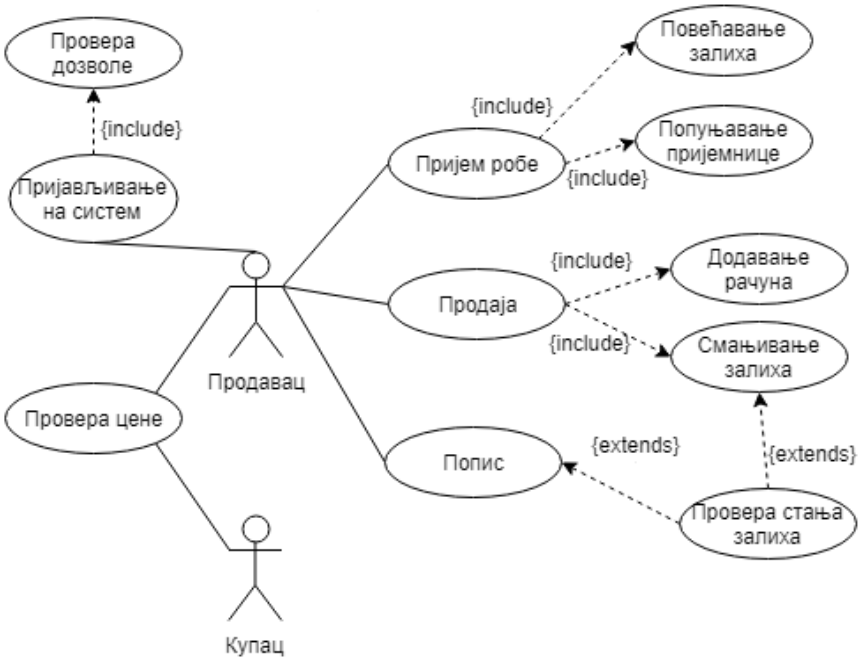
Слика 9. Структура базе података

4.2. Случај коришћења

Сличај коришћења је поглед на систем са становишта коришћења система, приказују се функционалности система и њихово коришћење од стране свих предвиђених корисника

система. У случају овог система, постоје три врсте корисника: Купац, продавац и магационер (особа запослена у складишту). На слици 10, приказан је систем са становишта коришћења датог система од стране купца и продавца.

Купац има само једну могућност коришћењем датог система, проверу цене одређеног производа.



Слика 10. Случај коришћења, Купац - Продавац

Продавац има четири основна случаја коришћења датог система и један додатни случај. Пријава корисника на систем је додатна функционалност система и не убрајамо је у случај коришћења система, али је приказана на слици 10. Пријава на систем укључује и проверу дозволе датог корисника, за приступ систему и за приступ само одређеним деловима система. Продавац као и купац има могућност провере цене производа. Уз то купац има три додатна случаја коришћења:

- Пријем робе, овај случај коришћења обухвата и функције повећавања залиха у датој продавници и попуњавања пријемнице приликом пријема робе.
- Продаја, случај коришћења који обухвата и функције смањивања залиха, као и додавања рачуна у систем. Такође приликом смањивања залиха, прво је потребно проверити стање залиха, што је приказано на слици као проширење фунцкионалности.
- Попис, случај коришћења који је проширен функцијом провере стања залиха.

Магационер (особа запослена у складишту) као и продавац мора да се пријави на систем, при чему се проверава његова дозвола за приступ топ делу система. Уз пријављивање магационер има још 5 случаја коришћења који су приказани на слици 11, а то су:

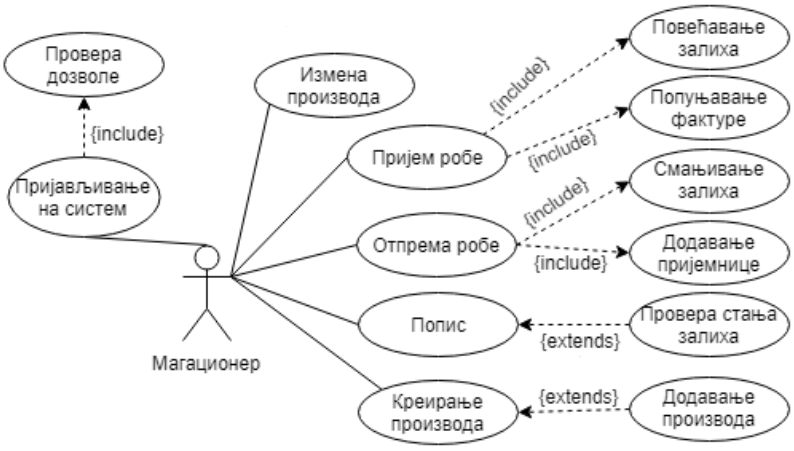
● Измена производа, магационер има могућност скенирања података у виду QR кода, а затим позива систему ради измене неког производа.

● Пријем робе, магационер приликом пристизања робе у складиште, потврђује пријем робе и количине, овај случаја коришћења обухвата функције повећавања залихе, као и попуњавања фактуре.

● Отпрема робе, овај случај коришћења је припрема транспорта робе из складишта у продавницу, приликом које се дати случај обухвата функције смањивања залихе и додавања пријемнице, пријемницу касније допуњава продавац који робу прима у продавницу.

● Попис, случај коришћења који је проширен функцијом провере стања залиха.

● Креирање производа, случај коришћења који је проширен функцијом додавања производа у систем. Магационер користи овај случај приликом пријема робе, а када дата роба није већ унета у базу података.



Слика 11. Случај коришћења, Магационер

Случаји коришћења приказани на сликама 10 и 11 дају могућност читаоцу рада да лакше схвати коришћење датог система. Олакшавају коришћење система корисницима. Циљ није показивање како свака од функција ради, већ како се систем у целини користи и у којој улози може бити који корисник система.

4.3. Опис решења

Код коришћен у решењу описан је детаљно, али нису описане све класе, методе и друго. Описани су најбитнији делови и такође скренута је пажња на остале сличне делове, који нису описани. Решење се састоји из два дела, андроид и Django пројкта. Прво иде опис андроид пројекта, а затим Django пројекта.

4.3.1. Андроид опис решења

Приликом генерисања пројекта за андроид апликацију као део пројекта, битна су два фајла основних подешавања:

AndroidManifest.xml, фајл у којем се дефинишу основна структура паликације, овде су дефинисане све активности, као и њихови мета подаци. Поред тога у истом фајлу су затражене дозволе за, приступ интернету и приступ камери. Поред тога овде је LoginActivity дефинисан као почетна, одностно активност која се прва покреће.

build.gradle, пројекат и његове верзије, као и build процес се обављају уз помоћ gradle алата за управљање пројектом. На основу тога већина почетног подешавања се контролише у датом фајлу. Прво приликом генерисања пројекта, а као део овог фајла подешава се верзија андроид оперативног система, минимална верзија и верзија за коју се на крају пројекат "build"-yje. У случају пројекта, минимална API верзија је 23, а пројекат дефинисан на верзију 28. Поред верзије система у истом фајлу се дефинишу библиотеке које је потребно увести ради коришћења у пројекту, као и њихове верзије. Као што је приказано на слици 12, у делу "dependencies", уз помоћ команде "implementation" додају се наведене библиотеке.

```
01. dependencies {
02.     implementation 'com.google.code.gson:gson:2.8.2'
03.     implementation 'com.journeyapps:zxing-android-embedded:3.6.0'
```

Слика 12. Део кода за увоз библиотеки у пројекат

Након иницијалног подешавања пројекта, остатак кода се може поделити на две најбитније целине, сам Java код који представља логику и .xml фајлове који представљају изглед апликације, такозвани "layout"-и. Ови xml фајлови су генерисани уз помоћ дизајнера који је интегрисан у андроид студију, сходно томе код који се налази у њима неће бити приказан у раду. Што се тиче Java дела пројекта, може се поделити на три целине, што уствари представља три пакета. Први пакет представљају андроид активности или екрани које сам корисник апликације види, затим средњи слој односно код који је задужен за пословну логику и комуникацију преко интернета са другим делом пројекта и као задњи елемнт су модели, односно подаци који постоје у паликацији.

Активности

У пројекту су посебно подељене три категорије активности, опште активности у које спадају:

SettingsActivity, активност у којој се подешава IP адреса на којој се налази локални сервер, Ова активност служи зас једноставније тестирање апликације у време програмирања и није потребна у случају комерцијалног коришћења апликације.

LoginActivity, активност у којој се корисник пријављује на систем, или приступа систему у улози купца, где се стартује активност "KupacActivity", као што је приказано на слици 13. Поред тога из ове активности се уз помоћ дугмета приступа активности "SettingsActivity".

```
1.  Button b1 = (Button) findViewById(R.id.guestLogin);
2.  b1.setOnClickListener(new View.OnClickListener() {
3.     @Override
4.     public void onClick(View v) {
5.       startAktivnost(KupacActivity.class);
6.     }
7.  });
8.  private void startAktivnost(Class x){
9.    Intent i = new Intent(context, x);
10.   context.startActivity(i);
11. }
```

Слика 13. Покретање активности купца

KupacActivity, активност која даје могућност скенирања QR кода на производу и проверу цене производа, ову активност корисници апликације могу користити и без пријаве на систем. Иста активност се позива опцијом продавца, када жели да провери цену производа.

```
1.  private BarcodeCallback callback = new BarcodeCallback() {
2.      @Override
3.      public void barcodeResult(BarcodeResult result) {
4.          if(result.getText() == null || result.getText().equals(lastText)) return;
5.          lastText = result.getText();
6.          barcodeView.setStatusText("Proizvod skeniran...");
7.          beepManager.playBeepSoundAndVibrate();
8.          n.getProizvod(ip, lastText);
9.      }
10.     ...
11. };
12.
13. @Override
14. protected void onCreate(Bundle savedInstanceState) {
15.     super.onCreate(savedInstanceState);
16.     setContentView(R.layout.activity_kupac);
17.     barcodeView = (DecoratedBarcodeView) findViewById(R.id.barcode_scanner);
18.     Collection<BarcodeFormat> formats = Arrays.asList(BarcodeFormat.QR_CODE, BarcodeFor
    mat.CODE_39);
19.     barcodeView.getBarcodeView().setDecoderFactory(new DefaultDecoderFactory(formats));

20.     barcodeView.initializeFromIntent(getIntent());
21.     barcodeView.decodeContinuous(callback);
22.     barcodeView.setStatusText("Skenirajte barkod proizvoda...");
23.     beepManager = new BeepManager(this);
24.     ...
25. }
26. private Runnable reportStatus = new Runnable() {...};
27. private void setValues(ProizvodModel x){...}
```

Слика 14. “KupacActivity” најбитнији делови кода

На слици 14 је приказан главни део кода активности за проверу цене производа, од линије 1 до 11 је приказана дефиниција методе callback, ово је метода која се касније поставља на методу коју скенер QR кода позива приликом успешног скенирања. У линији 4 је if провера, која осигурава да се исти код неће више пута очитати скенером. Након тога се податак чува, затим се мења текст и пуста се звук који означава успешно скенирање. На

самом крају методе barcodeResult налази се позив методе у класи Network, метода која за одређени „ИД“ прибавља податке о производу. У методи onCreate је приказан део кода који обавља иницијално подешавање скенера, поставља формате, почетни текст и све потребне објекте иницијализује. На самом крају апстрактно су приказане две методе, прва је уствари објекат класе Runnable и она се прослеђује класи Network која дату методу позива у тренутку када пристигне одговор од сервера. Из разлога што се не може знати када ће одговор од сервера да пристигне, цео процес комуникације је имплементиран асинхроно. Последња метода setValues је метода која параметре датог производа поставља у одговарајућа поља на екрану (textView), приказује их кориснику. На слици није приказано, али поред приказаног за рад скенера је потребно позивати одговарајуће методе на onResume и onPause, у андроид циклусу. У провј се покреће процес активне претраге обејката за скенирање, а у другој се тај процес зауставља из разлога оптимизације оптерећења андроид уређаја у случају када пликација није активна. Помоћу дате активности је описан рад и коришћење скенер дела апликације, сходно томе даље у тексу биће наглашено да се скенер користи, али не и како, јер је то објашњено овде.

Поред општих активности које су описане, постоје још две категорије активности, активности за продавца и активности за радника у складишту. Све активности независно од типа корисника су имлементиране на сличан начин. Приликом пријаве корисника на систем (“LoginActivity”), одређује се тип корисника, затим се отвара одговарајућа активност са опцијама главног менија.

```
1.  public class ProdavacActivity extends AppCompatActivity {
2.      @Override
3.      protected void onCreate(Bundle savedInstanceState) {...}
4.      public void onPrijem(View v){
5.          Intent myIntent = new Intent(this, ProdavnicaPrijemRobe.class);
6.          this.startActivity(myIntent);
7.      }
8.      public void onPopis(View v){...}
9.      public void onRacun(View v){...}
10.     public void onProvera(View v){...}
11. }
```

Слика 15. Приказ кода главног менија за тип корсника, продавац

Као што је приказано на слици 15, код је једноставан, за свако дугме из менија се поставља по једна метода, која затим покреће одговарајућу активност, као што је приказано за методу “onPrijem”, у линијама 4 и 5 на слици 15. Остале методе су приказане апстрактно, и на исти начин функционишу. Иста ситуација је и код радника у складишту, принцип је исти само су опције менија друге, и активности које се покрећу друге.

Кроз пример једне од активности које примењеују процес скенирања, позива средњег слоја и мењања стањеа или провере стања у бази, објашњен је целокупан процес, на основу којег се базирају све активности у пројекту. За све активности идеја и процес је исти, присутне су мање варијације примене. На слици 16 приказан је део кода активности “ProdavnicaRacunActivity”. Све активности у пројекту наслеђују андроид

“AppCompatActivity”, ова класа је изведена од основне класе активности у андроиду, тиме наша класа постаје једна активност. Затим у линијама 2-6 је апстрактно приказана дефиниција потребних променљивих у активности (нису приказане све променљиве). У линији 6 је дефинисана статичка променљива, у овом случају поља заглавља у табели, која се касније појављује у активности, променљиве и статичке се обично дефинишу на почетку класе.

```
1.  public class ProdavnicaRacunActivity extends AppCompatActivity {
2.      private ProizvodModel proizvod;
3.      ...
4.      List<TableModel> dataTable = new ArrayList<>();
5.      List<ArtikalModel> artikliRacun = new ArrayList<>();
6.      private static final String[] TABLE_HEADERS={ "Naziv", "Cena", "Kolicina","Cena" };
7.
8.      @Override
9.      protected void onCreate(Bundle savedInstanceState) {
10.         ...
11.         context = this;
12.         resetSetUp();
13.     }
```

Слика 16. Почетак активности за издавање рачуна

Затим је од линије 8 до линије 13 апстрактно приказана метода “onCreate”, ово је стандардна андроид метода, која се позива приликом првог покретања активности. Овде се постављају почетни параметри, позивом методе “resetSetUp”, или директно као пример “context”. На слици 17, је приказана метода за подешавање почетних параметара. У методи се постављају објекти класа, листе се поново иницијализују, подешава се начин функционисања скенера и мењају се параметри на корисничком интерфејсу (линија 24, онемогућивање дугмета). Све ово је могло бити написано и у “onCreate” методи, али потреба да се исти код више пута позове довела је до прављења овакве методе. Слична пракса није ретка у програмиању, где се поступци који се понављају у процесу дефинишу као метода, а затим по потреби позивају.

```
15.     private void resetSetUp(){
16.         dataTable = new ArrayList<>();
17.         artikliRacun = new ArrayList<>();
18.
19.         n = new Networking(reportStatus);
20.         setScanner();
21.         setTable(getList());
22.
23.         proizvod = null;
24.         disableNaplati();
25.         ukupno = 0.00;
26.         setUkupno(0.0);
27.     }
```

Слика 17. Методе за почетна подешавања активности

На слици 18 је приказан остатак активности, апстрактно, само најбитнији делови за овај рад. У линији 29 (слика 18) се види дефиниција методе за подешавање скенер дела, који је раније описан у раду. У линији 30 (слика 18) је дефиниција методе која попуњава табелу на екрану корисника, метода прихвата дводимензијонални низ текста, који метода затим претвара у податке табеле. Линија 31 је дефиниција методе која само одређене параметре враћа на почетну вредност приликом позива методе додавања производа на рачун, подешава се текст у скенеру, количина се враћа на вредност 1 и ако дугме за наплату рачуна није омогућено, омогућава се, јер постоји производ на рачуну, а наплата није могућа са празним рачуном.

Линије 32 и 33, су дефиниције метода које се позивају у случају притиска дугмета. Метода “onNaplati” се покреће притиском одговарајућег дугмета, затим покреће процес наплате рачуна, ово је пример како функционише процес, прво се подешава метода коју класа “Network” позива након успешно извршеног позива према серверу, узимају се потребни подаци за дати позив серверу, линија 35, податак о пријављеном кориснику на систем (продавцу), генерише се објекат, односно податак за слање. На крају линија 37. позив саме методе у класи “Networking”.

```
29.     private void setScanner(){...}
30.     private void setTable(String[][] data){...}
31.     private void resetOnDodaj(){...}
32.     public void onDodaj(View v){...}
33.     public void onNaplati(View v){
34.         n = new Networking(reportNaplata);
35.         int uID = session.getUserId();
36.         RacunModel racun = new RacunModel(uID, artikliRacun);
37.         n.naplati(ip, racun);
38.     }
39.     private void setUkupno(double novo){
40.         TextView ukupno = findViewById(R.id.ukupnoTV);
41.         this.ukupno += novo;
42.         ukupno.setText(this.ukupno + " rsd");
43.     }
44.     private Runnable reportStatus = new Runnable() {...};
45.     private Runnable reportNaplata = new Runnable() {...};
46. }
```

Слика 18, Остале методе у активности

Приликом одговра од сервера позива се метода “reportNaplata”, метода дешифрује статус одговора и о датом статусу обавештава корисника апликације (продавца). Одговор може бити позитиван, где је наплата успешна, или ако постоје грешка у систему са стањем или нешто друго, о датом проблему се обавезтава продавац. У линијама 39-43 је дат пример методе која мења стање корисничког интефејса, у овом случају се мања укупна вредност рачуна.

Класе средњег слоја, пословна логика

Послове средњег слоја андроид апликације обављају две класе. Класа “Session”, класа која је задужена за рад са сесијским подацима на телефону, то јест подацима у локалном складишту података. И класа “Networking”, класа која комуницира са сервером, на захтев слоја изнад (на захтеве из активности).

Класа “Session” обавља послове чувања података у “SharedPreformances” делу андроида. Ово локално складиште је предвиђено за чување малих података, којима је приступ потребан из свих делова паликације. На слици 19 је приказан начин на који ова класа ради. За сваки податак класа има две методе, једну за чување и другу за учитавање података. Уз то класа чува референцу према контексту апликације, тако да се при прављењу објеката, односно позиву конструктора мора проследити контекст из којег се приступа апликацији. Довољно је да се у активности каже “new Session(this);”.

```
1.  public class Session {
2.      Context app;
3.      public Session(Context context) {
4.          this.app = context;
5.      }
6.      public void saveUserId(int id) {
7.          SharedPreferences.Editor editor =
8.                  app.getSharedPreferences("FASIC", MODE_PRIVATE).edit();
9.          editor.putInt("id", id);
10.         editor.apply();
11.     }
12.     public int getUserId() {
13.         SharedPreferences prefs = app.getSharedPreferences("FASIC", MODE_PRIVATE);
14.         int id = prefs.getInt("id", -1);
15.         return id;
16.     }
17.     public void saveIp(String ip){...}
18.     public String getIp(){...}
19. }
```

Слика 19, Класа“Session”

У линијама 6-11 је приказана метода која чува податке, прво се генерише објекат едитора, затим се додаје податак са методом “put” (у овом случају у линији 9, ID пријављеног корисника), на крају се уз помоћ методе “apply” податак чува. Метода за читање податка позива методу “get” као у линији 14 на слици, затим се податак враћа кроз методу, позиваоцу.

Класа “Networking” као и класа “Session”, је дефнисиана као обична JAVA класа и не наслеђује ниједну класу. Уз то битно је рећи да класа садржи као параметар објекат класе “Runnable”. Кроз конструктор је овај параметар дефинисан као обавезан. На слици 20 се поред датог виде и дефиниције неких атрибута класе.

```
1.  public class Networking {
2.      private Runnable onUpdate;
3.      final Handler mHandler = new Handler();
4.      public int status;
5.      ...
6.      public Networking(Runnable onUpdate) {
7.          this.onUpdate = onUpdate;
8.      }
```

Слика 20. Класа “Networking” дефиниција и конструктор

Основни задатак класе је комуникација са сервером. На слици 21 су приказана два начина на који класа комуницира са сервером, “GET” и “POST” позивима. На слици су приказане две “JAVA” методе које обављају задатак комуникације.

```
9.      private String getStringFromURL(String url) {
10.         try {
11.             HttpURLConnection urlConnection = null;
12.             URL urlClass = new URL(url);
13.             urlConnection = (HttpURLConnection) urlClass.openConnection();
14.             urlConnection.setRequestMethod("GET");
15.             ...
16.             urlConnection.connect();
17.             String allText = new BufferedReader(...).readLine();
18.             return allText;
19.         ...
20.     }
21.
22.     private String postStringFromURL(String url, String message) {
23.         try {
24.             ...
25.             OutputStreamWriter wr = new OutputStreamWriter(...);
26.             wr.write(message);
27.             wr.flush();
28.
29.             BufferedReader reader = new BufferedReader(...);
30.             String line = reader.readLine();
31.             ...
32.             return ...;
33.         ...
34.     }
```

Слика 21. “GET” и “POST” методе у класи “Networking”

Од линије 9 до линије 20 је приказана метода која шаље “GET” захтев серверу, приказ је апстрактан, одређена подешавање комуникације као и catch део кода нису приказани на слици. Метода као параметар прихвата путању према серверу, којој се шаље упит, а на свом излазу враћа примљени одговор добијен од сервера. Протокол комуникације је једноставан, успоставља се конекција са сервером, а затим се уз помоћ “readLine” методе чита одговор сервера. Одговр је у текстуалном облику, у случају овог система “JSON” формата. На линијама 22-34 је приказана метода која шаље “POST” захтев, дата метода је доста слична “GET” методи и није је потребно детаљно објашњавати. Једина релевантна разлика је у томе што дата функција прима два параметра, путању на којој се налази

сервер и податак који је потребно проследити серверу. Податак који се шаље серверу мора бити у текстуалном облику, па је ранијим позивим GSON библиотеке, објекат JAVA класе преведен у текстуални податак у “JSON” формату. Поред ове две методе класа садржи и по једну методу за сваки задатак, однсоно функцијоонална ост која је активностима потребна. Па тако су ту методе за пријављивање корисника, прављење рачуна, потврду пријема робе, добијење података о производу и друге. Све ове функције раде на истом принципу, уз мање варијације у имплементацији. Описом једне од ових метода читаоцу ће бити јасније како уопштено све раде.

```
35.     public void login(final String ip, final String jmbg, final String pin) {
36.         Thread thread = new Thread() {
37.             @Override
38.             public void run() {
39.                 Gson gson = new Gson();
40.                 String message = gson.toJson(new LoginModel(jmbg, pin));
41.                 String jsonString =
42.                     postStringFromURL("http://" + ip + "/api/login", message);
43.                 logedInUser = gson.fromJson(jsonString, LoginModel.class);
44.                 status = logedInUser.status;
45.                 mHandler.post(onUpdate);
46.             }
47.         };
48.         thread.start();
49.     }
50. }
```

Слика 22. Једна од метода класе “Networking”

На слици 22 је приказана једна од метода кеје се позивају из активности, а која обавља потребан позив према серверу, затим поступа по протоколу. Приказана је метода “login” која служи пријављивању корисника на систем. Метода прихвата 3 параметра, први параметар као и већина метода, IP адресу на којој се налази сервер. Након тога прихвата два потребна параметра за пријаву на систем, то су јмбг број запосленог и његов пин број. Целокупан процес који функција обавља се покреће на посебној нити, због перформанси ситема. У случају позива према серверу се не може знати када тачно ће сервер дати одговор на постављени упит, задржавање система на оваквом процесу би довело до успреног рада апликације, па се из тог разлога слични процеси обављају на посебној нити. Као први корак у линији 40 подаци о корисику који се пријављује на систем се преводе у одговарајући формат, текстуални податак “JSON” формата. Затим се у линији 41 позива одговарајућа метода класе која шаље “POST” захтев, њој се прослеђују одговарајући парамтери. Уједно се прихвата одговор сервера и чува у стринг променљивој. Дати податак се преводи у објекат JAVA класе, на крају се поставља параметар који означава статус одговора. И као последњи корак позива се одговарајућа метода која је постављена приликом прављења објекта класе, тако што се метода додаје у “handler” уз помоћ методе “post()”. Последњи сегмент андроид апликације су модели,

односно класе које представљају моделе, ко помоћ у комуикацији са сервером или као директно пресликавање табеле у бази.

```
1.  public class TableModel {
2.      String naziv;
3.      String cena;
4.      String kolicina;
5.      String cenaUkupno;
6.      public TableModel(...) {...}
7.      public String[] getArray() {
8.          String[] a = {naziv, cena, kolicina, cenaUkupno};
9.          return a;
10.     }
11. }
12. public class ArtikalModel {
13.     int id;
14.     int kolicina;
15.     public ArtikalModel(int id, int kolicina) {
16.         this.id = id;
17.         this.kolicina = kolicina;
18.     }
19. }
```

Слика 23. Пример модела у андроид пројекту

На слици 23 су приказана два модела, пројекат има већи број модела, на слици су приказана само два примера модела. Све класе које предтављају моделе су обична јава класа, без наслеђивања код које атрибути представљају вредности. Поред атрибута модели садрже одговарајући конструктор, или већи број потребних конструктора. Уз то као на примеру на слици 23, могу садржати и неку додатну методу, на примеру модела за табеле (линије 1-11), метода “getArray”, је помоћна метода која од једног објекта класе прави низ стрингова, за потребе лакшег попуњавања табеле.

4.3.2. Django опис решења

Пројекат серверског дела апликације који је рађен у Django framework-у, а у PyCharm окружењу, не захтева претерано коликована почетна подешавања, потребно је свега неколико основних параметара подесити. На слици 24, приказане су најважније промене у Settings.py фајлу, главном фајлу пројекта у којем се мењају основна подешавања. У првој линији је приказано подешавање дозвољених адреса на којима се дати сервер може огласити као видљив. У случају овог пројекта је важно подесити ову адресу из разлога што се серверу приступа споља, односно са андроид уређаја, па је из тог разлога за адресу приступа постављена јавна адреса рачунара на којем се налази сервер, у локалној мрежи. Остале линије на слици 24 приказују подешавања базе података, у пројекту. Ова подешавања су подразумевана и нису мењана у односу на основна подешавања Django пројекта. У случају потребе да апликација користи другу технологију базе података, приказани код би био изењен одговарајућим кодом за ту технологију базе података.

```
1. ALLOWED_HOSTS = ['192.168.1.111']
2. DATABASES = {
3.     'default': {
4.         'ENGINE': 'django.db.backends.sqlite3',
5.         'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
6.     }
7. }
```

Слика 24. Settings делови Django пројекта

Након прављења пројекта, у оквиру истог могуће је направити неколико апликација, које су део датог пројкта. У случају овог система у питању је једна апликација и приликом генерисања исте потребно је направити неколико мањих корекција пројектних фајлова, како би апликација била видљива и валидна у самом пројекту. У Settings.py фајлу у делу инсталираних апликација је потребно додати ново креирану апликацију, начин је приказан на слици 25 линија 3.

```
1. INSTALLED_APPS = [
2.     ...
3.     'MasterRad.apps.MasterradConfig',
4. ]
5. class MasterradConfig(AppConfig):
6.     name = 'MasterRad'
```

Слика 25. Подешавања апликације у Django пројекту

Да би апликација била придодата пројекту, у дату апликацију и њен подфолдер се мора додати одговарајући фајл подешавања, најчешће се фолдер апликације зове исто као и сама паликација, а дати фајл се зове apps. У линији 3 видимо да је наведена путања према фајлу, <folder>.apps.<naziv_klase>. Једини садржај фајла је класа, која у себи садржи атрибут, назив апликације.

Целокупан Django пројекат садржи глобалне приступне тачке. У случају овог система, за глобалне тачке имамо две, приказане на слици 26. Прва путања представља приступну тачку админ портала, док је друга приступна тачка за генерисану апликацију, одностно серверски део пројекта. У на слици 26 у линији 3 је приказана путања којом се приступа коду серверског дела пројекта, “/api”.

```
1. urlpatterns = [
2.     path('admin/', admin.site.urls),
3.     path('api/', include('MasterRad.urls')),
4. ]
```

Слика 26. Глобалне путање Django пројекта

У случају потребе коришћења подеразумеваног Django админ портала, а у овом пројекту се користи, потребно је поставити основна подешавања портала. Као прво потребно је направити супер корисника, односно админа целокупног админ пројекта. Особу која може да се пријави на систем и преко админ портала да мења вредности у табелама базе. Осим тога потребно је и подесити које се све табеле у бази могу мењати кроз админ портал.

```
1. admin.site.register(Proizvod)
2. admin.site.register(Skladiste)
3. ...
4. admin.site.register(Dobavljac)
5. admin.site.register(Zaposleni)
```

Слика 27. Подешљвање админ портала

На слици 27, у скраћеном облику је приказан фајл admin.py, који у себи садржи списак табела базе које се кроз админ портал могу директно мењати.

Приступне тачке серверу система

Поред раније поменутих, глобалних тачака приступа, свака од апликација у Django прокејту садржи своје посебне тачке приступа. На слици 28 су приказане све приступне тачке система. Тачке се дефинишу тако што се задаје приступна путања тачке, ово је уствари допунски део путање у односу на раније постављену путању у глобалној дефиницији, затим се додељује метода која се позива приступм датој тачки. Као опциона могућност је додељивање назива, ради лакших позива у одређеним ситуацијама. Иако опциона могућност, давање имена је препоручено. На слици 28 се могу видети примери разних начина дефинисања путањи. У линији 2 на слици 28, приказана је основна путања, односно почетна страница система, у случају овог система ова страница нема примену.

```
1. urlpatterns = [
2.     path('', views.index, name='index'),
3.     path('login', views.login, name='login'),
4.     path('proizvod/<int:id>', views.proizvod, name='proizvod'),
5.     path('proizvod/generisanje', ...),
6.     path('proizvod/izmena', views.proizvod_izmena, name='proizvod_izmena'),
7.     path('skladista', views.skladista, name='skladista'),
8.     path('dobavljaci', views.dobavljaci, name='dobavljaci'),
9.     path('prijemnica/<int:id>', views.prijemnica, name='prijemnica'),
10.    path('prijemnica/provera', views.prijemnica_provera, name='prijemnica_provera'),
11.    path('prijemnica/potvrda', views.prijemnica_potvrda, name='prijemnica_potvrda'),
12.    path('racun/naplata', views.kasa_naplata, name='kasa_naplata'),
13.    path('popis/potvrda', views.popis_potvrda, name='popis_potvrda'),
14.    path('skladiste/otprema', views.skladiste_otprema, name='skladiste_otprema'),
15.    path('skladiste/faktura', views.skladiste_faktura, name='skladiste_faktura'),
16.    path('skladiste/faktura/potvrda', ...),
17. ]
```

Слика 28. Приступне тачке серверу

Затим у линији 3 је представљена путања која на глобалну путању додаје још само једну реч. У наредна два реда (ред 4, односно 5), приказане су путање које постојећу глобалну путању проширују за две или више речи. Уз то је у линији 4 је приказан пренос параметара кроз путању, однсно бројне вредности, у приказаном случају ID-а производа (<int:id>).

Пример дефиниције модела система

Модел у Django пројекту представља python класу која наслеђује модел класу. Атрибути класе су поља у табели у бази, док методе представљају операције које је могуће и потребно радити на датом пољу. Методе такође могу бити помоћне, у случају да су потребне одређене трансформације података табеле у бази. На слици 29 је дат пример

дефиниције једног модела. На истој слици у линијама 2-16 су дефиниције поља у датој табели, поља се задају позивом “models.”, а затим тип поља, док се у загради задају параметри поља, односно понашање поља у разним случајевима, или одређена ограничења (constraints). Од линије 16 до 33 су дефинисане одређене методе које су потребне за лакши рад са датим моделом. Прва метода, toString (__str__()), трансформише објекат класе и текст, и најчешћа примена је код админ портала (приказ једног објеекта изведеног из модела).

```
1.  class Prijemnica(models.Model):
2.      skladiste = models.ForeignKey(Skladiste, on_delete=models.CASCADE)
3.      prodavnica = models.ForeignKey(Prodavnica,
4.                                     on_delete=models.CASCADE,
5.                                     blank=True,
6.                                     null=True)
7.      kolicina = models.IntegerField()
8.      proizvod = models.ForeignKey(Proizvod, on_delete=models.CASCADE)
9.      zaposleni_prodavnica = models.ForeignKey(Zaposleni,
10.                                              on_delete=models.CASCADE,
11.                                              related_name='pradavac',
12.                                              blank=True,
13.                                              null=True)
14.     zaposleni_skladiste = models.ForeignKey(Zaposleni,
15.                                             on_delete=models.CASCADE,
16.                                             related_name='skladistni_radnik')
17.
18.     def __str__(self):
19.         return str(self.skladiste_id) + "->" +
20.                str(self.prodavnica_id) + " (" +
21.                str(self.proizvod) + " - " +
22.                str(self.kolicina) + ")"
23.
24.     def as_json(self):
25.         return dict(
26.             id=self.id,
27.             proizvod=self.proizvod.naziv,
28.             kolicina=self.kolicina,
29.             skladiste=self.skladiste.naziv,
30.             status=1)
31.
32.     def potvrdjena(self):
33.         return self.zaposleni_prodavnica != None
```

Слика 29. Пример модела

Друге две методе су помоћне методе, и олакшавају позиве према објектима датог модела, прва објекат трансформише у одговарајући JSON формат, док друча даје одговор на постављено питање, да ли је пријемница потврђена. Овако дефинисан модел се једноставним позивом за миграцију кроз Django пројекат (“migrate”), трансформише у табелу са одређеним пољима, у одговарајућој технологији базе података (SQLite, у примеру овог система).

Методе Views.py

Дефинисањем метода, које се позивају приступном тачком, прави се средњи слој система, односно пословна логика, тачније шта се, у ком случају, при одређеним параметрима дешава са системом.

```
1.  @csrf_exempt
2.  def login(request):
3.      return safeCall(loginDo, request)
4.
5.  def proizvod(request, id):
6.      return safeCall(proizvodGet, id)
7.
8.  def skladista(request):
9.      return safeCall(skladisteGet, request)
10.
11. @csrf_exempt
12. def prijemnica_provera(request):
13.     return safeCall(prijemnicaPotvrda, request)
14.
15. def safeCall(whatToCall, param):
16.     try:
17.         return whatToCall(param)
18.     except Exception as ex:
19.         print("SafeCall ERROR")
20.         print(ex)
21.         return getJsonReturnCode(0)
```

Слика 30. Методе позива у Views.py

На слици 30 је приказан део метода које се позивају из приступних тачака. Чињеница да се већина ових метода позива у “try-except” блоку наредби је довела до потребе за доданом методом која обезбеђује сигуран позив сличних метода. На слици 30 у линији 15 је дефиниција “safeCall” помоћне методе, позивом ове методе са параметром методе коју је потребно сигурно позвати, се обезбеђује несметани рад система у сваком случају, и при евентуалним грешкама у параметрима. Од линије 1 до линије 13 су дати примери неколико метода које могу бити позване, неке представљају крајње таче GET протокола, друге су крајње тачке POST протокола. Нотација “@csrf_exemp”, даје назнаку методама POST протокола да није потребно извршити CSRF заштиту.

На слици 31 приказане су методе које обављају пословну логику система, директно. Тачније прве методе, помоћу “safeCall” методе, позивају ове методе, које директно обављају функције и обраду података. На слици 31 у лији 40 је приказана трансформамција, пристиглих JOSN података, који се пребацују у python податак. Задавање одговора се види неколико пута на слици 31, а то је позив методе HttpResponse са одговарајућим параметрима. Добијање података од базе се постиже позивом класе која представља модел, односно ентитет у бази на који се надовезују команде “objects” и “get” (Zaposleni.objects.get). У загради позване методе се задају параметри за селекцију појављивања ентитета у бази.

```
23. def skladisteGet():
24.     sk = list(Skladiste.objects.values())
25.     ret = {'status': 1, 'data': sk}
26.     return HttpResponse(
27.                 JsonResponse(ret, safe=False),
28.                 content_type="application/json"
29.             )
30. 
31. def proizvodGet(id):
32.     i = Proizvod.objects.get(pk=id)
33.     i = i.as_json()
34.     return HttpResponse(
35.                 JsonResponse(i, safe=False),
36.                 content_type="application/json"
37.             )
38. 
39. def loginDo(request):
40.     data = json.loads(request.body.strip())
41.     z = Zaposleni.objects.get(jmbg=data["jmbg"], pin=data["pin"])
42.     if (z):
43.         return HttpResponse(
44.                     JsonResponse({'status': z.radno_mesto, 'id': z.id}, safe=False),
45.                     content_type="application/json"
46.                 )
47.     else:
48.         return getJsonReturnCode(0)
```

Слика 31. Радне методе Views.py

5. Изглед корисничких екрана и примена система

Као последњи сегмент рада, у овом поглављу приказан је изглед самиог система, као и опис коришћења, односно примене у пракси. Као прво приказан и опсисан је изглед, а затим описана примена система.

5.1. Изглед андроид апликације

На слици 32 приказани су екрани андроид апликације, тачније четири екрана. Гледајући с’ лева на десно (од а до г слике), прва слика представља екран за пријаву корисника на систем, на екрану се виде параметри које је потребно да корисник унесе да би се пријавио на систем, уносом, а затим притиском дугмета “логин” корсиник приступа систему по одређеном улогом. У случају да корисник није запослен, да је у питању купац, преко дугмета “купац” пристуа одговарајућем делу система. Поред ових команди на дну екрана се налази помоћна команда, односно дугме помоћу којег се покреће екран за подешавање рада система у локалној мрежи, односно подешавање IP адресе сервера система. На другој слици (б), приказан је екран који представља мени, односно екран избора могућих функција које корсиник може обавити, на слици је приказан екран менија за запосленог у складишту, екран за продавца је сличан, разликује се број и распоред дугмића. Корисник притиском дугмета прелази на следећи екран, односно екран на којем

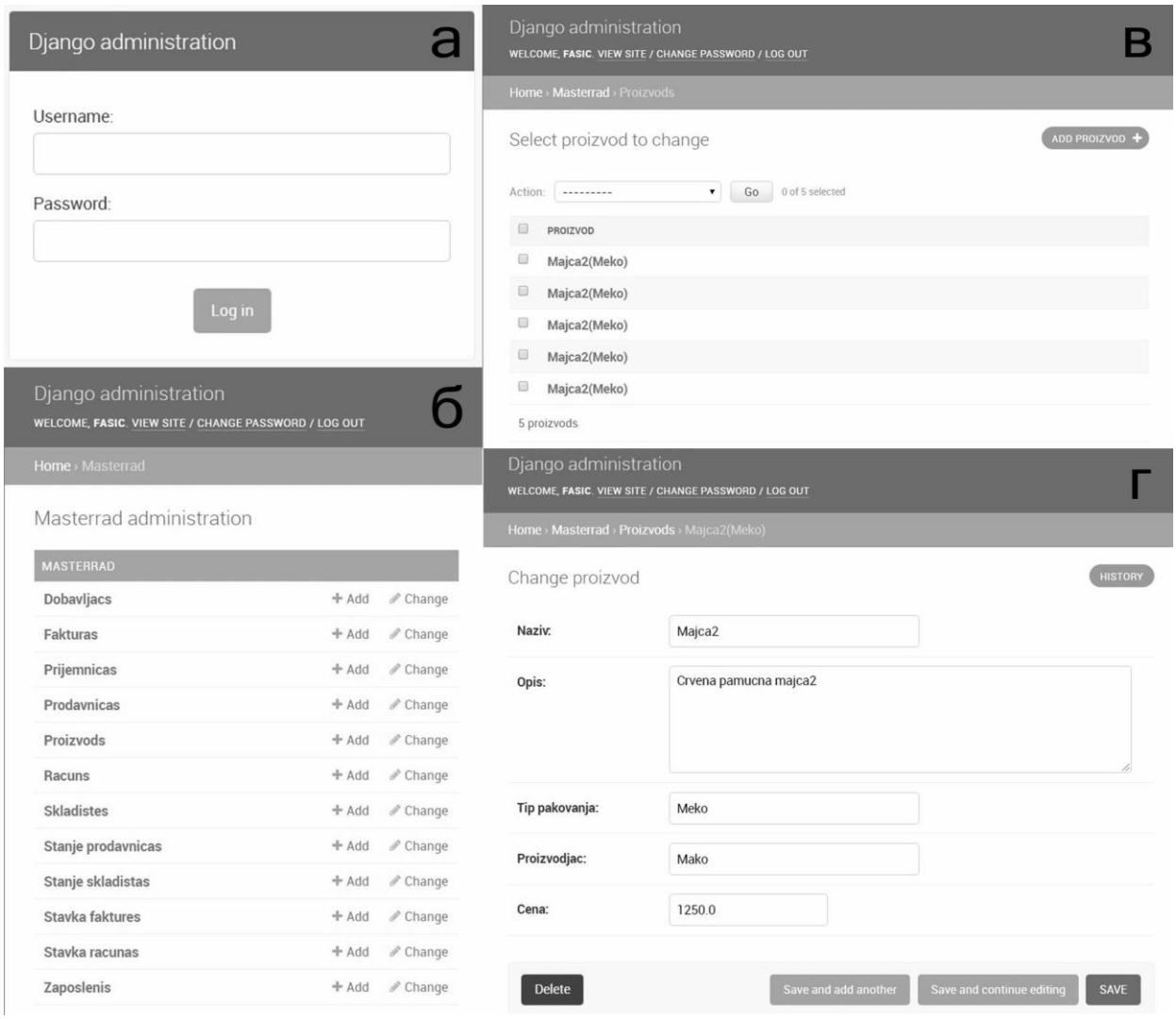
се обављају одређене функције. На последње две слике (в и г), приказани су радни екрани, екрани на којима се обављају функције система.

Слика 32. Приказ екрана андроид апликације, *а) пријава корисника на систем, б) главни мени запосленог у складишту, в) провера цене, г) издавање рачуна*

На трећој слици је приказан екран провере цене производа, исти екран може да користи продавац или купац (корисник система који се не пријављује на систем). На врху екрана је део корисничког интерфејса који служи за скенирање кодова. Испод се налази приказ података о скенираном производу, цена, назив и друго. На последњој четвртој слици је приказан екран који продавац користи приликом израде рачуна. На врху екрана је сегмент интерфејса за скенирање кодова, затим испод тога се налази сегмент у којем корисник бира количину производа која је продата, затим се налази дугме за додавање производа на рачун. Испод тога се може видети табела, односно приказ додатих производа на рачун. На дну екрана се налази укупна вредност рачуна, као и дугме за наплату, притиском на које се шаље упит серверу за скидање производа са стање и осталих провера.

5.2. Изглед админ портала

На слици 33 су приказана четири основна екрана админ портала, односно корисничког интефејса Django пројекта. Админ портал има једноставну структуру и принцип коришћења. Прва слика (а) приказује екран пријаве корсиника на портал, податке које корисник треба да унесе, односно корисничко име и шифра се генеришу у Django пројекту, то су подаци superuser-а. Након успешне пријаве на портал, кориснику се приказује екран са изгледом приказаним на слици 33б. Приказује се списак табела у бази, са линковима где корисник приступом линку може мењати податке, додавати нове и брисати постојеће.



Слика 33. Приказ екрана админ портала, *а) Пријава на портал, б) Избор табела у бази, в) Избор ентитета за измену, г) Измена вредности*

Посећивањем линка отвара се страница са згледом приакзаним на слици 33в, приказ списка свих ентитета унетих у дату табелу. Поврх екрана је дугме за додавање новог ентитета. Селекцијом одређеног ентитета, а затим покретање команде која се налази на врху екрана (“action”), корисник може обрисати дати ентитет. Покретање линка отвара се измена ентитета, измена је приказана на слици 33г. Поља која се отварају зависе од табеле и поља која садржи дата табела. На дну екрана се налазе дугмичи за контролу, односно брисање, чување или одустајање од измене датог ентитета.

5.3. Коршћење односно примена система

Целокупан процес примене, коришћења оваквог система је пут од врха према дну, са концептом потпуне аутоматизације. Овај систем могу користити два (односно три ако се

урачуна у купац) типа корисника, са мањом дорадом односно мањим изменама на систему, могућ је и већи број корисника, у зависности од потреба. Ако се крене од првих задатака, функција које је могуће обављати, набавка робе је једноставном провером, увидом у стање у складиштима и продавницама лако могућа. Недостају још извештаји такве провере и механизам наручивања од добављача директно, сходно томе да то није тема овог рада о томе није причано. Након провере, затим наручивања, када роба пристигне, аутоматизован је процес пријема односно означавања примљене робе, и додавање на стање у складишту. Поред тога запослени у складишту има апстрактно приказане још две могућности, а то је уношење нових и мењање постојећих производа. Ово је приказано само као апстрактна могућност, односно као опција која се ослања не неког ко је раније генерисао потребне податке за унос, односно измену. Поред тога запослени у складишту може да проверава стање кроз опцију пописа. На крају може да покрене процес напуштања робе у складишту, односно отпреме исте. Након тога роба пристиже (физички) у продавницу, где даље управљање преузима продавац запослен у датој продавници. Он такође помоћу скенера, скенира податке са робе, и кроз једноставан интерфејс управља подацима о роби, прима робу, касније продаје, а повремено врши попис или проверава цену на упит купца. Цео овај процес је потпуно аутоматизован, од врха до дна, систему за комерцијалну примену недостају одређени извештаји, који нису део рада, и одређени механизми за даљу комуникацију.

Једноставним скенирањем података са кутија, паковања или штампаних на папиру, могу се постићи све функционалности једног оваквог система. Цео процес убрзава рад, олакшава посао, и смањује могућност за грешком, односно губљењем података о роби или саме робе. Трошкови одржавања оваквог система су знатно мањи од смањивања трошкова који могу настати проблемима у пракси, на пословима складишта без примене сличног система. Примена сличног система је широка, уз спајање система са фискалним касама систем би мога бити примењн свуда и у сваком примеру, односно случају. Једна од великих предности за коришћење сличног система би била у продавницама које имају велики асортиман робе, односно честе промене асортимана.

6. Закључак

У раду је апстрактно представљен систем који може бити јако комликован, али је у овом раду као концепт, због једноставности објашњавања, поједностављен. Систем се састоји из два дела, андроид апликације и Django сервера. Прво су описане ове две целине уз теоријски опис технологија, које се користе приликом израде система. Након тога је приказана структура и архитектура система, начини комуникације у систему и примена система. Систем даје могућност примене за три различита типа кориска, сви процеси у систему су оптимизовано аутоматизовани и предвиђени су да олакшавају рад запосленима. Велика предност оваквог система је широк спектар примене, уз јако ниске трошкове за имплементацију система, није потребна додатна опрема за скенирање, или приступ систему. Систему се приступа путем андроид телефона и апликације, а кенирање се обавља камером која је уграђена у сам уређај.

Систем има велики потенцијал за надоградњу, у примени истих технологија приликом израде, али и у проширењу применом других технологија. Повезивање система са фискалном касом би био један од првих задатака проширења, затим проширење капацитета у виду различитих типова корисника и њихових задатака, које могу обављати. Практично би могле бити одвојене целине као што су: маркетинг, набавка, продаја, финансије и друго. Па би тако сваки од наведених типова корисника имао приступ систему, са одговарајућим задацима. Уз то даља дорада сигурносних система, односно разних провера и заштита приступа систему, али и неким деловима система би даље олакшало одржавање система односно коришћење истог. Имплементацијом делеова система који нису део овог рада, као што су уношење складишта, продавница, интерфејс за унос робе и слчично, систем би могао бити коришћен у комерцијалне сврхе.

Такође систем је могуће унапредити и применом других и нових технологија. Два основна концепта која се могу поставити су примена машинског учења и примена хардвера за решавање неких препрека. Као први и једноставнији концепт надоградње система била би примена ардуино и “RaspberryPi” платформи са сензорским мрежама и “RFID” односно “NFC” таговима (Mingxing et al., 2012). Идеја је да се датим платформама повежу сензори на систем, како из сигурносних разлога (детекција пожара рецимо, или детекција упада у току ноћи), тако и из разлога једноставности коришћења система, однсно праћења робе која улази у складиште, напушта складиште, и где се налази у складишту. За неке од ових задатака систем имплементира скенер, који користи камеру, применом ових технологија би овај процес додатно био аутоматизован.

Још један концепт применом којег би систем добио на аутоматизацији и/или једноставности примене, концепт машинског учења. Као пример примене концепта су рецимо примена за обавештавање запослених о року трајања робе у складишту, однсно продавници. Такође концепт би био примењив и на систему учења о карактеристикама продаје робе, када је роба пристигла, након којег времена је продата, како би продаја могла бити оптимизована, однсно аутоматско генерисање акција продаје, аутоматско генерисање плана маркетинга, на основу набавке, продаје и стања (Pulungman, 2013). Као

и аутоматског генерисања извештаја о набавци и добављању робе продавницама. Оваквом технологијом се посао запослених радника, пребацује на сам систем, предности су једноставнос одржавања, затим константност и квалитет обављених задатака, све ово доприноси велике финансијске предности.

Применом оваквог система, отварају се нове могућности са проширењем истог. Приликом коришћења система се може до закључака шта је и како потребно систему, приликом коришћења у пракси. Применом система на једном примеру добио на значају, а затим би константним радом на систему могао бити проширен у неком од наведених праваца, по потреби. Проширењем система, или применом нових технологија.

7. Литература

1. Feinbube Frank, (2011), "*Embedded OS-Android*", Hasso-Plattner-Institut, Potsdam.

2. Holovaty Adrian, (2007), "*Django's History*", Chicago, [Интернет] Доступно на: https://web.archive.org/web/20150702070736/http://www.djangobook.com/en/2.0/chapter01.html [Приступљено: 20.07.2019].

3. Masse Mark, (2012), "*RestAPI Design Rulebook*", O'Reilly Media, Sebastopol.

4. Mingxing Deng, Jian Mao, Xingwen Gan, (2016), "*Development of Automated Warehouse Management System*", Shanghai University of Engineering Science, Shanghai. [Интернет] Доступно на: https://www.researchgate.net/publication/329036247_Development_of_Automated_Warehouse_Management_System [Приступљено: 20.07.2019].

5. Pulungman Reza, (2013), "*Design of an Intelligent Warehouse Management System"*, ISICO. [Интернет] Доступно на: https://www.researchgate.net/publication/259558571_Design_of_An_Intelligent_Warehouse_Management_System [Приступљено: 29.07.2019].

6. Šukalo Adi, (2016), "*Android Operativni Sistemi*", Travnik. [Интернет] Доступно на: https://edoc.pub/android-operativni-sistem-seminarski-rad-pdf-free.html [Приступљено: 20.07.2019].

7. Tan Jin Soon, (2008), "*QR code*", Synthesis journal, [Интернет] Доступно на: https://foxdesignsstudio.com/uploads/pdf/Three_QR_Code.pdf [Приступљено: 20.07.2019].

8. Wang Jingna, Guowei Hua, (2019), "*Design of Android Warehouse Management Software based on Web Service*", IOP Conf. Ser.: Earth Environ. Sci. 252 042009. [Интернет] Доступно на: https://iopscience.iop.org/article/10.1088/1755-1315/252/4/042009/pdf [Приступљено: 20.07.2019].

9. Војиновић Светлана, (2012), "*Primer operativnog sistema – Android*", Београд.

10. Ервин Пепић, (2016), "*Напредни алати и технологије за развој веб апликација*", Нови Пазар. [Интернет] Доступно на: http://d.uninp.edu.rs/file.php/81/DIPL_RADOVI/Diplomski_rad_-_Ervin_Pepic_2016.compressed.pdf [Приступљено: 20.07.2019].

Интернет садржај:

11. Android zvanična stranica, https://developer.android.com [Приступљено: 20.07.2019].

12. Barkod skeneri, (2009), https://www.explainthatstuff.com/barcodescanners.html [Приступљено: 20.07.2019].

13. Django Software Foundation, (2019), "*Django*", Lawrence, Kansas. [Интернет] Доступно на: https://djangoproject.com [Приступљено: 20.07.2019].

14. GIT stranica biblioteke "gson", https://github.com/google/gson [Приступљено: 13.07.2019].

15. GIT stranica biblioteke "zxing", https://github.com/zxing/zxing [Приступљено: 13.07.2019].

16. Microservices, (2019), https://searchmicroservices.techtarget.com/ [Приступљено: 10.07.2019].

17. Struktura QR koda , (2019), [Интернет] Доступно на: https://www.keyence.com/ss/products/auto_id/barcode_lecture/basic_2d/qr/ [Приступљено: 20.07.2019].

18. Trygve Reenskaug, James O. Coplien, (2009), *"MVC arhitektura"*, [Интернет] Доступно на: https://www.artima.com/articles/dci_vision.html [Приступљено: 20.07.2019].

19. Vincent James, (2015), *"Broj android korisnika, novinski članak"*, [Интернет] Доступно на: http://www.theverge.com/2015/9/29/9409071/google-android-stats-users-downloads-sales [Приступљено: 20.07.2019].

20. WEB framework, (2014), https://web.archive.org/web/20150723163302/http://docforge.com/wiki/Web_application_framework [Приступљено: 20.07.2019].

21. Zvanična stranica "SQLite", https://www.sqlite.org/about.html [Приступљено: 12.07.2019].