

***Sistem za kupovinu u drogerijama***

(Projektni zadatak)

PREDMET: SE211 – Konstruisanje softvera

Profesor: Student:

**Milena Bogdanović Bojana Stajić 4596**

Asistent:

**Sara Nikolić**

Sadržaj

[1. Opšti opis problema 4](#_Toc105439013)

[1.1 Korisnici sistema 4](#_Toc105439014)

[1.2 Funkcije sistema 4](#_Toc105439015)

[2. Korisnički zahtevi 4](#_Toc105439016)

[2.1. Funkcionalni zahtevi 4](#_Toc105439017)

[2.2 Nefunkcionalni zahtevi 5](#_Toc105439018)

[3. Arhitektura sistema 6](#_Toc105439019)

[3.1 Opis arhitekture sistema 6](#_Toc105439020)

[4. Procesni model 6](#_Toc105439021)

[4.1 Procesni model za običnog korisnika 7](#_Toc105439022)

[4.2 Procesni model za administratora 7](#_Toc105439023)

[5. Klasni dijagram i dijagrami modula 8](#_Toc105439024)

[5.1 Dijagram klasa celog sistema 9](#_Toc105439025)

[5.2 Klasni moduli 9](#_Toc105439026)

[5.2.1 Klasni modul administratora 9](#_Toc105439027)

[5.2.2 Klasni modul kupovina 10](#_Toc105439028)

[5.2.3 Klasni modul dekorator 11](#_Toc105439029)

[5.2.4 Klasni modul korisničke kupovine 12](#_Toc105439030)

[5.3 Modelovanje klasnih metoda pomoću pseudokoda (PPP) 12](#_Toc105439031)

[5.1.1 Metoda prva 12](#_Toc105439032)

[5.1.2 Metoda druga 13](#_Toc105439033)

[5.1.3 Metoda treća 13](#_Toc105439034)

[5.1.4 Metoda četvrta 13](#_Toc105439035)

[5.1.5 Metoda peta 14](#_Toc105439036)

[5.1.6 Metoda šesta 14](#_Toc105439037)

[5.1.8 Metoda osma 15](#_Toc105439038)

[5.1.9 Metoda deveta 15](#_Toc105439039)

[5.1.10 Metoda deseta 15](#_Toc105439040)

[5.1.11 Metoda jedanaesta 16](#_Toc105439041)

[5.1.12 Metoda dvanaest 16](#_Toc105439042)

[5.1.13 Metoda trinaesta 17](#_Toc105439043)

[5.1.14 Metoda četrnaesta 17](#_Toc105439044)

[5.1.15 Metoda petnaesta 18](#_Toc105439045)

[5.1.16 Metoda šesnaesta 18](#_Toc105439046)

[5.1.17 Metoda sedamnesta 19](#_Toc105439047)

[5.1.18 Metoda osamnaesta 19](#_Toc105439048)

[5.1.19 Metoda devetneasta 20](#_Toc105439049)

[5.1.20 Metoda dvadeseta 20](#_Toc105439050)

[5.1.21 Metoda dvadeset prva 21](#_Toc105439051)

[5.1.22 Metoda dvadeset druga 21](#_Toc105439052)

[5.1.23 Metoda dvadeset treća 21](#_Toc105439053)

[5.1.24 Metoda dvadeset četvrta 21](#_Toc105439054)

[5.1.25 Metoda dvadeset peta 22](#_Toc105439055)

[5.1.26 Metoda dvadeset šesta 22](#_Toc105439056)

[5.1.27 Metoda dvadeset sedma 22](#_Toc105439057)

[5.1.28 Metoda dvadeset osma 22](#_Toc105439058)

[5.1.29 Metoda dvadeset deveta 23](#_Toc105439059)

[5.1.30 Metoda trideseta 23](#_Toc105439060)

[5.1.31 Metoda trideset prva 23](#_Toc105439061)

[5.1.31 Metoda trideset druga 23](#_Toc105439062)

[5.1.32 Interfejs Dekorisi 24](#_Toc105439063)

[6. Sekvencijalni dijagram 24](#_Toc105439064)

[6.1 Sekvencijalni dijagram kupovine 24](#_Toc105439065)

[6.2 Sekvencijalni dijagram uvida u korisnike 25](#_Toc105439066)

[6.3 Sekvencijalni dijagram opcije filtriranja 26](#_Toc105439067)

[7. Implementacija u programskom jeziku – Java 28](#_Toc105439068)

[7.1 Prikaz aplikacije 28](#_Toc105439069)

[7.2 Prikaz koda aplikacije 32](#_Toc105439070)

[7.3 Prikaz alata FindBugs 33](#_Toc105439071)

[7.4 Debagovanje aplikacije 34](#_Toc105439072)

[8. Testiranje 35](#_Toc105439073)

[8.1 JUnit testiranje- jedinično testiranje 35](#_Toc105439074)

[8.2 Determinističko testiranje 37](#_Toc105439075)

[8.3 Integraciono testiranje 39](#_Toc105439076)

[9. Zaključak 41](#_Toc105439077)

[10. Literatura 41](#_Toc105439078)

# 1. Opšti opis problema

Cilj i ideja ovog projekta jeste kreiranje jedne aplikacije koja bi omogućila kupovinu u jednoj drogeriji. Zadatak ove aplikacije jeste jednostavna i brza kupovina proizvoda sa fizičkog mesta drogerija i namenjena je korisnicima koji nisu u mogućnosti da kupe sa fizičkog mesta ili im je ovako lakše. Funkcionalnosti ovakve aplikacije bi bili odabir željenog proizvoda, njegovo filtriranje po različitim parametrima u cilju lakšeg pronalaženja proizvoda, i finalno kupovina proizvoda. U zavisnosti od funkcije: admin ili korisnik kupac neke funkcionalnosti će biti ograničene, tako će admin moći da upravlja proizvodima na stanju, da ih briše,menja ili dodaje. Platforma ove aplikacije bi bio Windows/Linux ili Mac operativni sistem, odnosno u pitanju bi bila jedna desktop aplikacija. Izrada i realizacija ove aplikacije podrazumevela bi sva pravila konstrukcije softvera i zadatke ovog projektnog zadatka.

## 1.1 Korisnici sistema

Postoje dve vrste korisnika ovog softverskog sistema. To su:

1. Korisnik/Kupac
2. Admin

Prvi korak pri pristupanju sistema je proces autentifikacije odnosno registracija/login putem korisničkog imena i šifre. U zavisnosti od uloge osobe koja pristupa sistemu prikazaće se različite funkcionalnosti. Korisnik, odnosno kupac ima mogućnost odabira proizvoda iz liste, pretrage proizvoda i konačno, kupovine gde unosi metod plaćanja. Admin, osim istih mogućnosti kao kupac korisnik ima i mogućnost dodavanja, brisanja i ažuriranja proizvoda koji su u trenutnoj ponudi drogerija.

## 1.2 Funkcije sistema

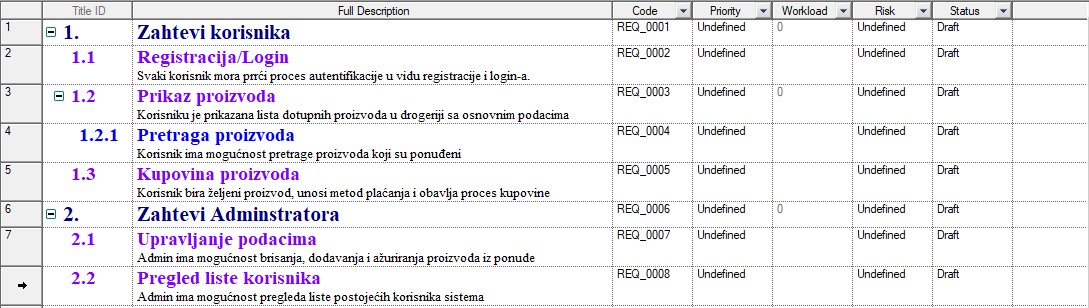
Funkcije sistema su:

* Autentifikacija korisnika
* Pretraga proizvoda
* Odabir proizvoda
* Kupovina proizvoda
* Ažuriranje, brisanje i dodavanje proizvoda

# 2. Korisnički zahtevi

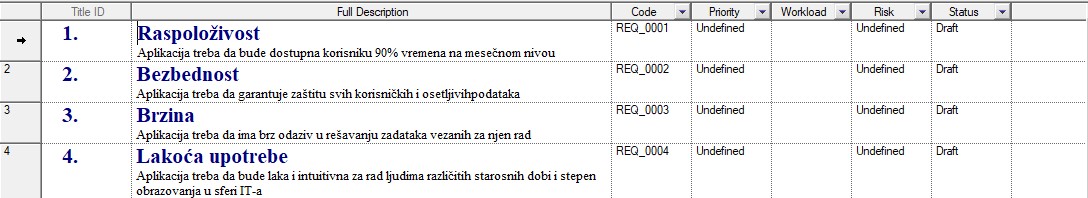
## 2.1. Funkcionalni zahtevi

U ovom poglavlju prikazani su i opisani svi funkcionalni zahtevi koje sistem treba da zadovolji. Zahtevi su izrađeni u Requirements modelu generisanom u alatu PowerDesigner. Zahtevi zavise od tipa korisnika sistema.



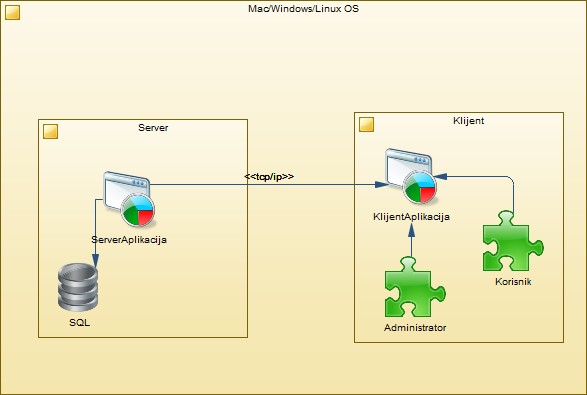
***Slika 1.*** *Prikaz Funkcionalnih zahteva*

## 2.2 Nefunkcionalni zahtevi



***Slika2.*** *Prikaz Nefunkcionalnih zahteva*

# Arhitektura sistema



***Slika3.*** *Dijagram arhitekture*

## Opis arhitekture sistema

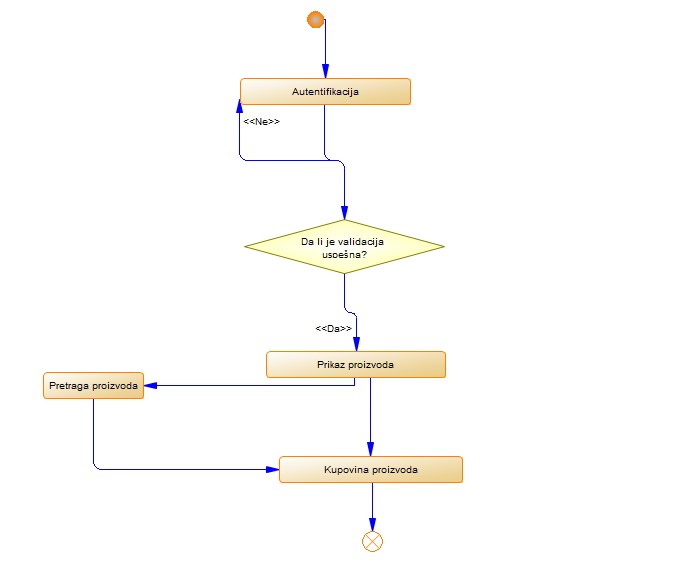
Šablon koji je upotrebljen za ovaj projektni zadatak je Klijent-server model. Klijent-server arhitektura je arhitektura računarske mreže u kojoj mnogi klijenti (udaljeni procesori) zahtevaju i primaju uslugu od centralizovanog servera (host računar).

Aplikacija se pokreće sa Desktop-a, odnosno na prikazanim operativnim sistemima. Sama aplikacija, odnosno klijentska strana je ista i za admina i za običnog korisnika. Zatim u zavisnosti od uloge server deo komunicira sa klijentom kao i sa bazom podataka, i tim putem se izvršavaju potrebne operacije bez redudantnosti.

Ovaj šablon je odabran kako je on najčešće korišćen model u ovakvom tipu aplikacija i njegova logika ja intuitivna i fleksibilna tako da odgovara ovom projektu.

# Procesni model

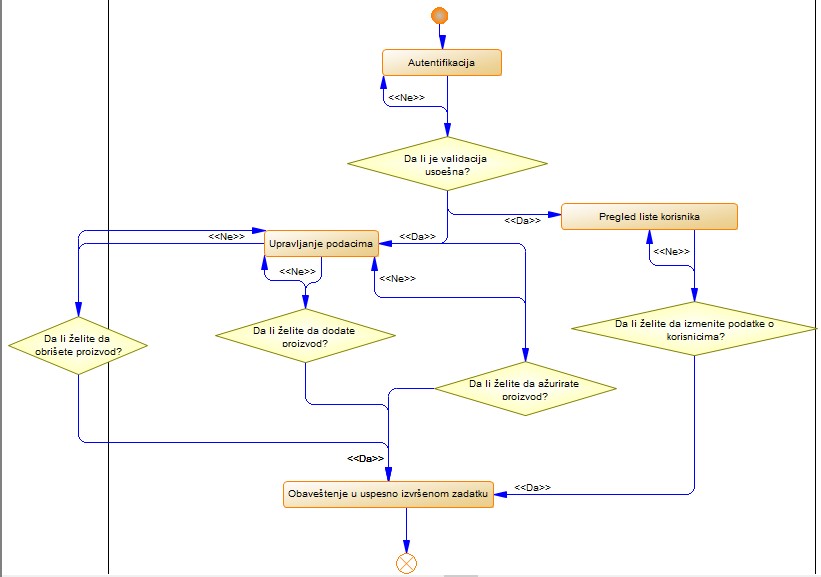
## 4.1 Procesni model za običnog korisnika



***Slika 4.***Prikaz procesnog modela za običnog korisnika

Pokretanjem aplikacije, sistem od korisnika zahteva prijavljivanje/registraciju. Ukoliko korisnik nema postojeći profil, može ga napraviti. Nakon uspešnog proces logovanja/registracije, korisniku se otvara pregled proizvoda u vidu tabele, a korisnik ima mogućnost pretrage proizvoda, kao i kupovine proizvoda, koja je ujedno i finalna akcija korisnika.(***Slika 4.***)

## 4.2 Procesni model za administratora



***Slika 5.***Prikaz procesnog modela za administratora

Aktivnosti administratora na početku dijagrama su iste kao kod običnog korisnika, jer mora prođe proces autentifikacije, a nakon toga, admin ima mogućnost pregleda liste korisnika i izmene podataka o istim, a isto tako ima mogućnost upravljanja podacima, gde razlikujemo procese brisanja, ažuriranja i dodavanja proizvoda. (***Slika 5.***)

# Klasni dijagram i dijagrami modula

## 5.1 Dijagram klasa celog sistema

Diagram

Description automatically generated

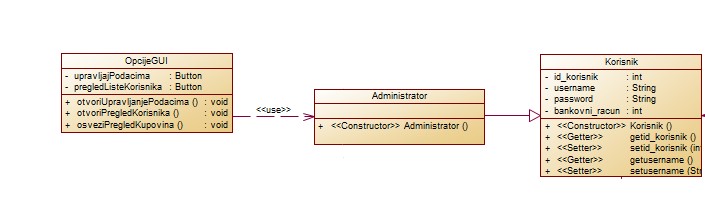
***Slika 6.*** *Dijagram klasa celog sistema*

Na prikazanom klasnom dijagramu nalaze se svi potrebni podaci o klasama koje učestvuju i čine sam softver, kao i njihove međusobne veze.

## 5.2 Klasni moduli

U nastavku sistem je prikazan po modulima, što će pomoći boljem razumevanju načina rada sistema pri razvoju.

### 5.2.1 Klasni modul administratora



***Slika 7.*** *modul administratora*

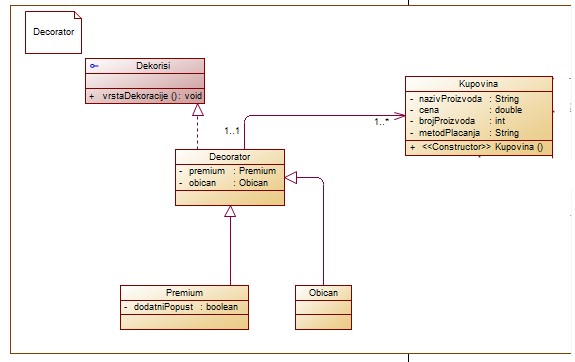
Modul administratora sastavljen je od klasa **Korisnik, Administrator** i **OpcijeGUI** a zadužen je za deo aplikacije koji se tiče admina sistema i njegovih odgovarajućih zadataka koje on može izvršiti nad sistemom. Nasleđuje klasu Korisnik kako od nje preuzima osnovne atribute koji su im zajednički.

### 5.2.2 Klasni modul kupovina

***Slika 8.*** *modul kupovina*

Modul kupovina izgrađen je od klasa **Kupovina, KupovinaBuilder i ProdajniObjekat**. Ovaj modul zajedno sa Builder klasom koja predstavlja upotrebljeni šablon ovog sistema čine kupovinu kao celinu kao i metode za njeno kreiranje.

### 5.2.3 Klasni modul dekorator



***Slika 9.*** *modul dekorator*

Modul dekorator izgrađen je od klasa **Decorator, Premium, Obican i Kupovina** kao i od interfejsa **Dekorisi.** U okviru ovog modula implementiran je Decorator šablon koji malo bolje definiše klasu Kupovinu dajući joj pomoću interfejsa dodatnu opciju odabira programa kupovine u smislu premim i obična vrsta kupovine.

### 5.2.4 Klasni modul korisničke kupovine

Diagram

Description automatically generated

***Slika 10.*** *modul korisničke kupovine*

Modul korisničke kupovine izgrađen je od klasa **Korisnik, KupovinaGUI i DBKontroler**. Ovaj modul zadužen je za definiciju GUI za korisničku stranu kupovine kao i metode za proveru korisničkog odabira i realizacije same kupovine.

## 5.3 Modelovanje klasnih metoda pomoću pseudokoda (PPP)

### 5.1.1 Metoda prva

Naziv metode: otvoriUpravljanjePodacima();

Ulaz: /.

Izlaz: /

BEGIN

//sistemska metoda otvara Upravljanje Podacima u okviru aplikacije i poziva se na objekat

o.otvoriUpravljanjePodacima();

END

### 5.1.2 Metoda druga

Naziv metode: zapocniKonekciju();

Ulaz: /.

Izlaz: Objekat klase Connection

*Globalna promenljiva*

Connection kon = null;

BEGIN

*kon = (Connection) DriverManager.getConnection("jdbc:mysql://localhost:3306/se211\_pz", "root", "");*

END

### 5.1.3 Metoda treća

Naziv metode: prekiniKonekciju();

Ulaz: /.

Izlaz: /

*Globalna promenljiva*

Connection kon = null;

BEGIN

*kon.close();*

END

### 5.1.4 Metoda četvrta

Naziv metode: otvoriPregledKorisnika();

Ulaz: /.

Izlaz: /

BEGIN

//sistemska metoda otvara Upravljanje Korisnikom u okviru aplikacije i poziva se na objekat

o.otvoriPregledKorisnika ();

END

### 5.1.5 Metoda peta

Naziv metode: otvoriPregledKupovina();

Ulaz: /.

Izlaz: /

BEGIN

//sistemska metoda otvara Upravljanje Kupovinama u okviru aplikacije i poziva se na objekat

o.otvoriPregledKupovina ();

END

### 5.1.6 Metoda šesta

Naziv metode: *filtrijaj(String grad, String naziv, String adresa)*;

Ulaz: String grad, String naziv, String adresa

Izlaz: ArrayList<ProdajniObjekat>

BEGIN

upit = "SELECT \* FROM `prodajniObjekat` WHERE grad LIKE '" + grad + "%' "; PreparedStatement ps = kon.prepareStatement(upit);

ResultSet rs = ps.executeQuery();

ArrayList<ProdajniObjekat> lista = new ArrayList<>();

while (rs.next())

lista.add(new ProdajniObjekat(rs.getInt(1), rs.getString(2), rs.getString(3), rs.getString(4), rs.getInt(5)));

END

### 5.1.8 Metoda osma

Naziv metode: proveraStanja(ProdajniObjekat o, int brojProizvoda);

Ulaz: ProdajniObjekat o, int brojProizvoda

Izlaz: boolean - stanje

BEGIN

if (brojProizvoda <= (o.getLagerKolicina() - dbKontr.ukupnoKupljenjo(o)) && brojProizvoda > 0)

System.out.println((o.getLagerKolicina() - dbKontr.ukupnoKupljenjo(o)));

return true;

else

return false;

END

### 5.1.9 Metoda deveta

Naziv metode: dodajKorisnika(Korisnik k);

Ulaz: Korisnik k (objekat klase Korisnik)

Izlaz:/.

BEGIN

upit = "INSERT INTO korisnik (korisnik\_username, korisnik\_password,korisnik\_adresa, bankovni\_racun) VALUES (?, ?, ?, ?)"

PreparedStatement ps = kon.prepareStatement(upit);

ps.setString(1, k.getUsername());

ps.setString(2, k.getPassword());

ps.setString(3, k.getAdresa());

ps.setString(4, k.getBankovni\_racun());

ps.executeUpdate();

END

### 5.1.10 Metoda deseta

Naziv metode: listaKorisnika();

Ulaz: /

Izlaz: ArrayList<Korisnik>

BEGIN

upit = "SELECT \* FROM `korisnik` ";

PreparedStatement ps = kon.prepareStatement(upit);

ResultSet rs = ps.executeQuery();

ArrayList<Korisnik> lista = new ArrayList<>();

while (rs.next())

lista.add(new Korisnik(rs.getString(2), rs.getString(3), rs.getString(4), rs.getString(5)));

END

### 5.1.11 Metoda jedanaesta

Naziv metode: obrisiKorisnika(String username);

Ulaz: String username

Izlaz: /

BEGIN

upit = "Delete FROM korisnik where korisnik\_username = ? ";

PreparedStatement ps = kon.prepareStatement(upit);

ArrayList<Korisnik> lista = listaKorisnika();

ps.setString(1, username);

ps.executeUpdate();

END

### 5.1.12 Metoda dvanaest

Naziv metode: izmeniKorisnika (Korisnik k, String username)

Ulaz: Korisnik k, String username

Izlaz:/.

BEGIN

upit = "UPDATE korisnik SET korisnik\_username = ?, korisnik\_password = ?,korisnik\_adresa = ?, bankovni\_racun = ? WHERE korisnik\_username = '" + username + "';"

PreparedStatement ps = kon.prepareStatement(upit);

ps.setString(1, k.getUsername());

ps.setString(2, k.getPassword());

ps.setString(3, k.getAdresa());

ps.setString(4, k.getBankovni\_racun());

ps.executeUpdate();

END

### 5.1.13 Metoda trinaesta

Naziv metode: prijavljeniKorisnik(String username, String password);

Ulaz: String username, String password

Izlaz:ArrayList<Korisnik>

BEGIN

upit = "SELECT \* FROM `korisnik` WHERE korisnik\_username = '" + username + "' AND korisnik\_password = '" + password + "'";

PreparedStatement ps = kon.prepareStatement(upit);

ResultSet rs = ps.executeQuery();

ArrayList<Korisnik> lista = new ArrayList<>();

while (rs.next())

lista.add(new Korisnik(rs.getInt(1), rs.getString(2), rs.getString(3), rs.getString(4), rs.getString(5), rs.getInt(6)));

END

### 5.1.14 Metoda četrnaesta

Naziv metode: usernameProvera(String username);

Ulaz: String username

Izlaz: boolean provera

BEGIN

upit = "SELECT korisnik\_username,korisnik\_password FROM `korisnik` WHERE korisnik\_username = '" + username + "'";

PreparedStatement ps = kon.prepareStatement(upit);

ResultSet rs = ps.executeQuery();

ArrayList<Korisnik> lista = new ArrayList<>();

while (rs.next())

lista.add(new Korisnik(rs.getString(1), rs.getString(2)));

boolean provera = false;

for (int i = 0; i < lista.size(); i++)

if (username.equals(lista.get(i).getUsername().replaceAll("\\s", "")))

provera = true;

END

### 5.1.15 Metoda petnaesta

Naziv metode: dodajKupovinu(Kupovina r);

Ulaz: Kupovina r

Izlaz:/.

BEGIN

upit = "INSERT INTO kupovina (korisnikId, objekatId,naziv\_proizvoda,metod\_placanja,broj\_proizvoda) VALUES (?, ?, ?, ?, ?)"; PreparedStatement ps = kon.prepareStatement(upit);

ps.setInt(1, r.getKorisnik().getKorisnik\_id());

ps.setInt(2, r.getPo().getObjekat\_id());

ps.setString(3, r.getNazivProizvoda());

ps.setString(4, r.getMetodPlacanja());

ps.setInt(5, r.getBrojProizvoda());

ps.executeUpdate();

END

### 5.1.16 Metoda šesnaesta

Naziv metode: listaKupovina(Korisnik kor);

Ulaz: Korisnik kor

Izlaz: ArrayList<Kupovina>

BEGIN

upit = "SELECT \* FROM kupovina JOIN korisnik ON kupovina.korisnikId = korisnik.korisnikId JOIN prodajniObjekat ON kupovina.objekatId = prodajniObjekat.objekatId WHERE korisnik.korisnikId = '" + kor.getKorisnik\_id() + "' ORDER BY kupovina.kupovinaId ASC";

PreparedStatement ps = kon.prepareStatement(upit);

ResultSet rs = ps.executeQuery();

ArrayList<Kupovina> lista = new ArrayList<>();

while (rs.next())

ProdajniObjekat o = new ProdajniObjekat(rs.getInt(14), rs.getString(15), rs.getString(16), rs.getString(17), rs.getInt(18));

Korisnik k = new Korisnik(rs.getInt(8), rs.getString(9), rs.getString(10), rs.getString(11), rs.getString(12), rs.getInt(13));

lista.add(new KupovinaBuilder()

.kupovina\_id(rs.getInt(1))

.Korisnik(k)

.ProdajniObjekat(o)

.naziv(rs.getString(4))

.cena(rs.getDouble(5))

.brojProizvoda(rs.getInt(7))

.metodPlacanja(rs.getString(6))

.build());

END

### 5.1.17 Metoda sedamnesta

Naziv metode: listaProdajnihObjekata();

Ulaz: /

Izlaz:ArrayList<ProdajniObjekat>

BEGIN

upit = "SELECT \* FROM `prodajniObjekat` ";

PreparedStatement ps = kon.prepareStatement(upit);

ResultSet rs = ps.executeQuery();

ArrayList<ProdajniObjekat> lista = new ArrayList<>();

while (rs.next())

lista.add(new ProdajniObjekat(rs.getString(2), rs.getString(3), rs.getString(4), rs.getInt(5)));

END

### 5.1.18 Metoda osamnaesta

Naziv metode: ukupnoKupljeno(ProdajniObjekat o);

Ulaz: ProdajniObjekat o

Izlaz: int ukupno

BEGIN

upit = "SELECT SUM(prodajniObjekat.lagerKolicina) as ukupno FROM kupovina INNER JOIN prodajniObjekat ON kupovina.objekatId = prodajniObjekat.objekatId WHERE prodajniObjekat.objekatId = '" + o.getObjekat\_id() + "' ORDER BY kupovina.kupovinaId ASC";

PreparedStatement ps = kon.prepareStatement(upit);

ResultSet rs = ps.executeQuery();

int ukupno = 0;

while (rs.next())

ukupno = rs.getInt(1);

END

### 5.1.19 Metoda devetneasta

Naziv metode: KorisnikTabel(TableView tv);

Ulaz: TableView tv

Izlaz: /

BEGIN

//Metod pravi tabelu putem JavaFX elemenata

END

### 5.1.20 Metoda dvadeseta

Naziv metode: ProdajniObjekatTabel(TableView tv);

Ulaz: TableView tv

Izlaz: /

BEGIN

//Metod pravi tabelu putem JavaFX elemenata

END

### 5.1.21 Metoda dvadeset prva

Naziv metode: KupovinaTabel(TableView tv, Korisnik kor);

Ulaz: TableView tv, Korisnik kor

Izlaz: /

BEGIN

//Metod pravi tabelu putem JavaFX elemenata

END

### 5.1.22 Metoda dvadeset druga

Naziv metode: kupovina\_id(int kupovina\_id);

Ulaz: int kupovina\_id

Izlaz: instanca klase KupovinaBuilder

private Kupovina kupovina;

BEGIN

this.kupovina.setKupovina\_id(kupovina\_id);

END

### 5.1.23 Metoda dvadeset treća

Naziv metode: naziv(String naziv);

Ulaz: String naziv

Izlaz: instanca klase KupovinaBuilder

private Kupovina kupovina;

BEGIN

this.kupovina.setNazivProizvoda(naziv);

END

### 5.1.24 Metoda dvadeset četvrta

Naziv metode: cena(double cena);

Ulaz: double cena

Izlaz: instanca klase KupovinaBuilder

private Kupovina kupovina;

BEGIN

this.kupovina.setCena(cena);

END

### 5.1.25 Metoda dvadeset peta

Naziv metode: Korisnik(Korisnik k);

Ulaz: Korisnik k

Izlaz: instanca klase KupovinaBuilder

private Kupovina kupovina;

BEGIN

this.kupovina.setKorisnik(korisnik);

END

### 5.1.26 Metoda dvadeset šesta

Naziv metode: Objekat(ProdajniObjekat o);

Ulaz: ProdajniObjekat o

Izlaz: instanca klase KupovinaBuilder

private Kupovina kupovina;

BEGIN

this.kupovina.setPo(objekat);

END

### 5.1.27 Metoda dvadeset sedma

Naziv metode: build ( );

Ulaz: /

Izlaz: instanca klase Kupovina

private Kupovina kupovina;

BEGIN

return this.kupovina;

END

### 5.1.28 Metoda dvadeset osma

Naziv metode: brojProizvoda(int brojProizvoda);

Ulaz: int brojProizvoda

Izlaz: instanca klase KupovinaBuilder

private Kupovina kupovina;

BEGIN

this.kupovina.setBrojProizvoda(brojProizvoda);

END

### 5.1.29 Metoda dvadeset deveta

Naziv metode: metodPlacanja(String placanje);

Ulaz: String placanje

Izlaz: instanca klase KupovinaBuilder

private Kupovina kupovina;

BEGIN

this.kupovina.setMetodPlacanja(placanje);

END

### 5.1.30 Metoda trideseta

Naziv metode: GUIMeni(Korisnik prijvaljeniKor);

Ulaz: Korisnik prijvaljeniKor

Izlaz: /

BEGIN

//Metoda definiše JavaFX elemente menija

END

### 5.1.31 Metoda trideset prva

Naziv metode: KupovinaGUI(Korisnik prijvaljeniKor, ProdajniObjekat o);

Ulaz: Korisnik prijvaljeniKor, ProdajniObjekat o

Izlaz: /

BEGIN

//Metoda definiše JavaFX elemente za kupovinu proizvoda i tabelarno dodaje novu kupovinu u tabelu i u bazu podataka

END

### 5.1.31 Metoda trideset druga

Naziv metode: PodKlasa();

Ulaz: /

Izlaz: /

BEGIN

//Metoda definiše JavaFX elemente za registraciju novog korisnika u vidu forme

END

### 5.1.32 Interfejs Dekorisi

Naziv metode: vrstaDekoracije();

Ulaz: /

Izlaz: /

BEGIN

//Apstraktna metoda interfejsa namenjena da bude implementirana u drugim klasama

END

# Sekvencijalni dijagram

## Sekvencijalni dijagram kupovine

Na detaljnom sekvencijalnom dijagramu za kupovinu, prikazan je proces kupovine iz perspektive korisnika. Najpre korisnik unosi podatke u formi zatim se oni proveravaju i ukoliko je sve kako treba kreira se nov objekat klase Rezervacija i korisnik se obaveštava o uspešnoj kupovini u drogeriji.( slika 11.)

Timeline

Description automatically generated with medium confidence

***Slika 11.*** *Detaljni sekvencijalni dijagram kupovine*

## 6.2 Sekvencijalni dijagram uvida u korisnike

***Diagram

Description automatically generated***

***Slika 12.*** *Detaljni sekvencijalni dijagram uvida u korisnike i njihove podatke*

Na slici 12. prikazan je detaljni sekvencijalni dijagram uvida u korisnike i njihove podatke iz perspektive admina. Ono što je značajno dodati, jeste da admin na isti način može da ima uvid u prodajne objekte i rezervacije, ali rezervacije ne može da izmeni niti briše, ali nisu napravljeni sekvencijalni dijagrami za taj scenario jer bi izgledali isto kao i ovaj uz minimalne izmene, a to bi stvorilo nepreglednost.

## Sekvencijalni dijagram opcije filtriranja

A picture containing diagram

Description automatically generated

***Slika 13.*** *Detaljni dijagram filtriranja*

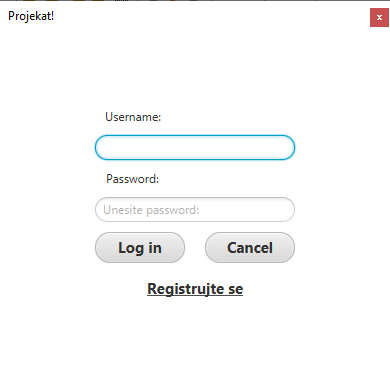
Na slici 13. prikazan je detaljni sekvencijalni dijagram filtriranja iz korisnikove perspektive. Naime, korisnik ima mogućnosti filtriranja/pretrage prodajnih objekata po parametrima, konkretno po gradu, adresi i nazivu prodajnog objekta.

# 7. Implementacija u programskom jeziku – Java

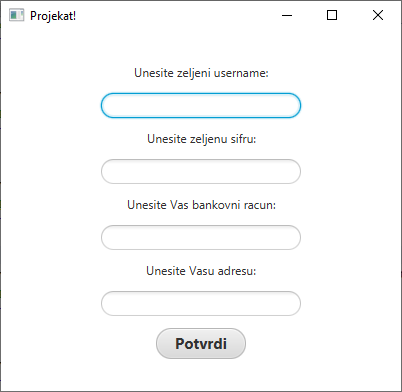
## 7.1 Prikaz aplikacije

U ovom poglavlju ukratko je prikazan i opisan korisnički interfejs sa kojim će se susresti korisnici ove aplikacije.

Korisnik se prvo susreće login formom kojom pristupa aplikaciji sa svojim postojećim validnim podacima.(***Slika 14.***)

******

***Slika 14. Početna strana za logovanje***



***Slika 15. Strana za registraciju korisnika***

**

***Slika 16. Početna strana admin korisnika(Kod korisnika izgleda isto sem što nema opcija List kupovina i Lista korisnika)***

***Graphical user interface, table

Description automatically generated***

***Slika 17. Opcija Lista Korisnika kod admin korisnika***

Graphical user interface, application, table

Description automatically generated

***Slika 18. Opcija lista kupovina kod admin korisnika***

***Graphical user interface, table

Description automatically generated***

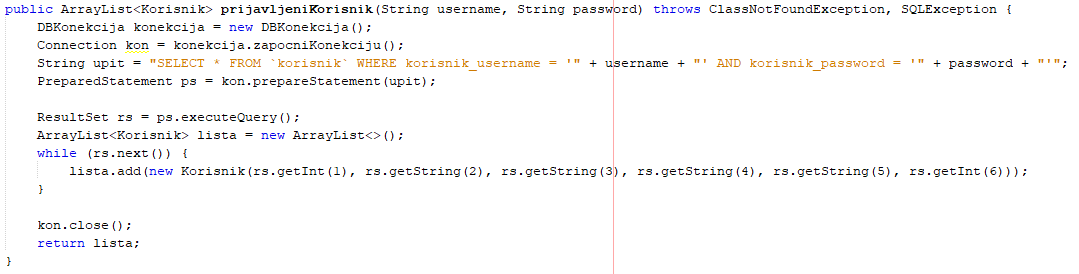
***Slika 19. Opcija Lista prodajnih objekata***

***Graphical user interface

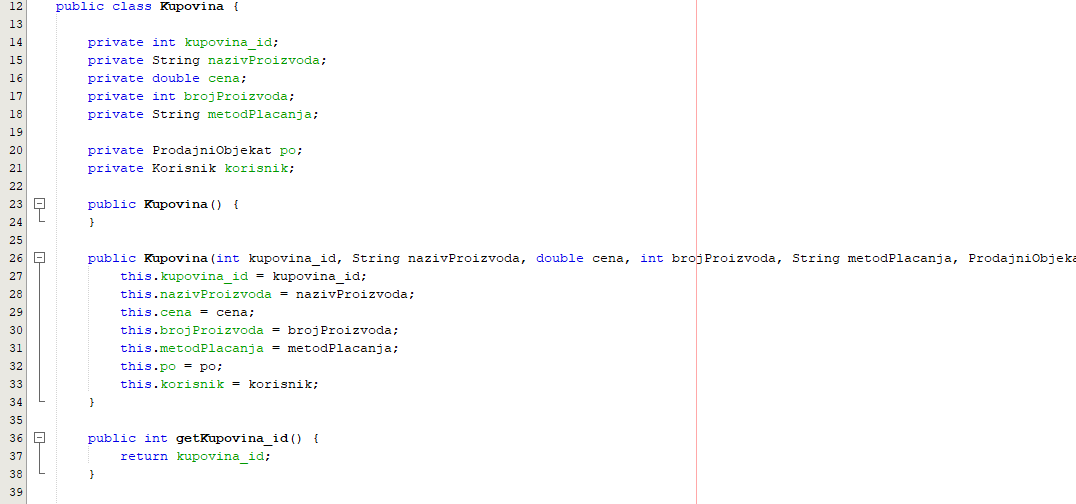
Description automatically generated***

***Slika 20. Forma za realizaciju kupovine***

## 7.2 Prikaz koda aplikacije



***Slika 21. metod za proveru korisnika prilikom login-a***



***Slika 22. Klasa Kupovina***

***Slika 23. Klasa KupovinaGUI***

## 7.3 Prikaz alata FindBugs

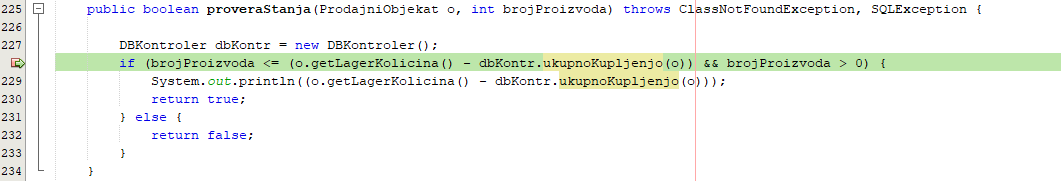
Text

Description automatically generated

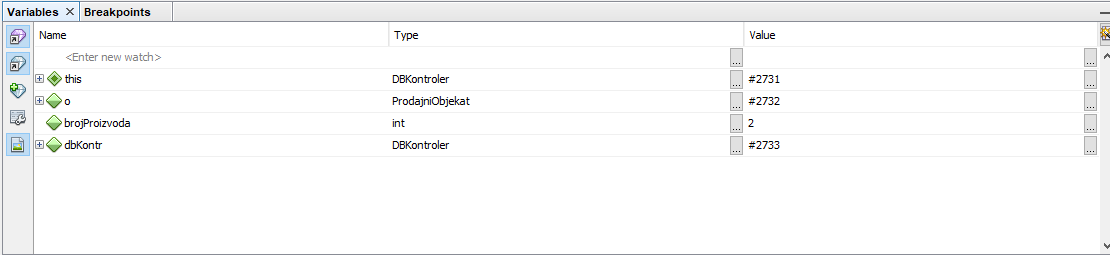
***Slika 24. Primena alata FindBugs na projekat***

Slika iznad prikazuje rad aplikacije FindBugs koja pronalazi bagove u kodu i daje sugestije za njihovo rešavanje. Na ostalim slikama prikazana je funkcionalnost aplikacije iz perspektive admina koji ima kontrolu nad svim podacima, dok bi se u korisničkom GUI pokazala samo tabela prodajnih objekata i mogućnost kupovine.

## 7.4 Debagovanje aplikacije



***Slika 25. Postavljanje breakpoint-a***



***Slika 26. Rezultat debugging-a***

Metoda služi za proveru raspoloživog stanja na lageru kada korisnik unese željeni broj proizvoda koji želi da kupi pa upoređuje te dve vrednosti da bi videla ima li željeni broj proizvoda na stanju. Na slici linija 228 predstavlja breakpoint. Breakpoint je signal debug alatu koji mu govori da provera počinje baš od te linije. Pokrenemo alat za debug u okviru razvojnog okruženja. Prolaskom kroz metodu u konzoli pratimo vrednost svih promenljivih kao što je prikazano na slikama iznad.

## 8. [Testiranje](#_Toc515048898)

## [8.1 JUnit testiranje- jedinično testiranje](#_Toc515048899)

Jediničnim testiranjem ćemo proveravamo da li pojedinačne komponente pravilo rade sa različitim ulazima. Definišemo unapred skup podataka koji ćemo koristiti u skladu sa scenarijem testiranja i rezultata koje očekujemo. Metod se testira tako što se pokrene i otvori se prozor testiranja u kome pratimo rezultate testova na osnovu datih ulaza.

U ovom konkretnom projektu, testiraćemo metodu koja proverava stanje na lageru kada korisnik želi da kupi određeni proizvod upoređujući količinu koju korisnik želi da kupi sa količinom koja je dostupna na lageru, a ta informacija se nalazi u bazi podataka. Testovima, kojih ima 4, predviđamo rezultate u normalnim okolnostima i ako je program ispravan i pravilno reaguje, sve testove će proći. U toku 4 testa, 1 je namenjen da namerno bude oboren, odnosno da padne u svrhu pokazatelja samog jediničnog testiranja. Kao što vidimo na slikama dole, testovi su se pokazali uspešnim i potvrdili iznete tvrdnje o sistemu i njegovoj pouzdanosti, pa čak i onaj koji je namenski oboren. Svaki test koji ne odgovara predviđenim Graphical user interface, application, Word

Description automatically generatedrezultatima i vrednostima biće oboren.

***Slika 27.(iznad) Prvi i namenski oboren test***

Namenski oboren test, gde je očekivan rezultat bio true, a zapravo je false, ovaj test je namenski oboren a u nastavku slede slike preostala 3 testa koja su uspešno prošla

Graphical user interface, text, application

Description automatically generated

***Slika 28. Uspešni testovi 2 i 3***

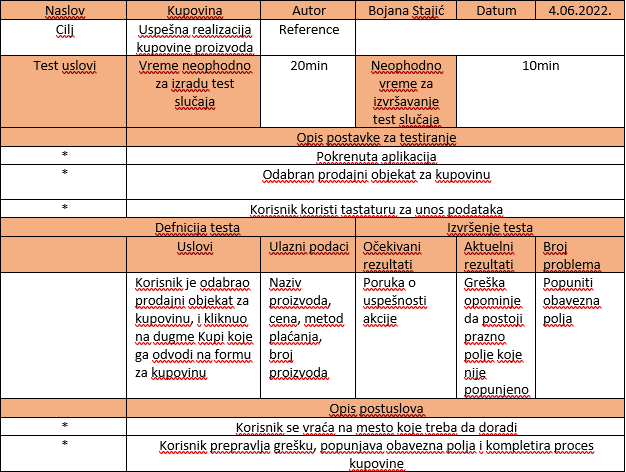
Graphical user interface, text, application

Description automatically generated

***Slika 29. Uspešni 4. test***

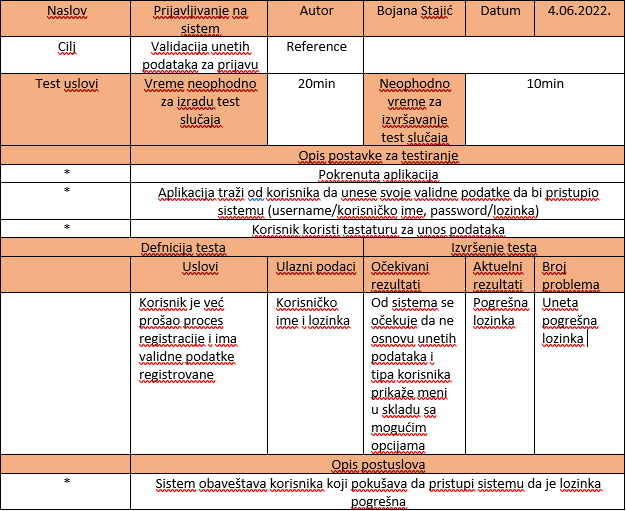
## 8.2 Determinističko testiranje

Determinističko testiranje predstavljamo putem tabele koja od podataka sadrži test slučaj kao i dobijeni rezultat testiranja.



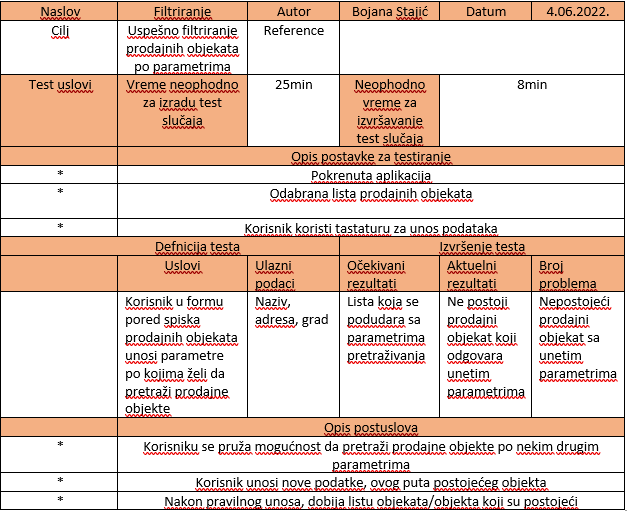
***Slika 30.*** *Kupovina*

Cilj ovog testiranja jeste uspešna realizacija proces kupovine. Preduslov je pokrenuta aplikacija, pristup nalogu kao i odabran prodajni objekat za kupovinu. Greška koja se javlja jeste da postoji prazno obavezno polje koje treba popuniti. Sistem će obavestiti korisnika i vratiti ga na mesto dorade.



***Slika 31.*** *Prijavljivanje na sistem*

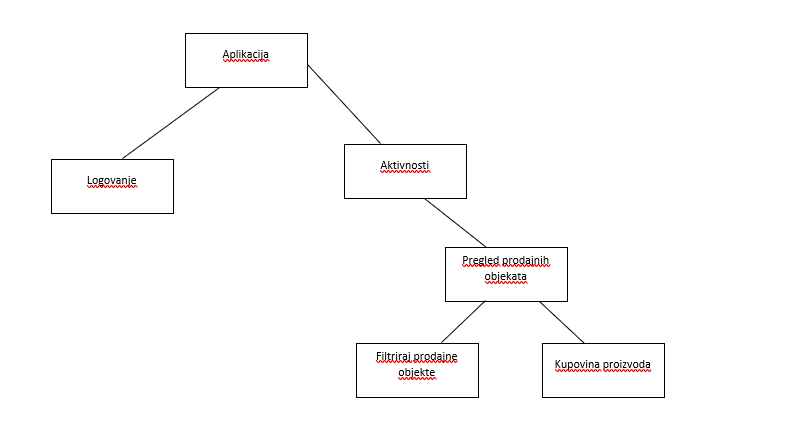
Cilj ovog testiranje jeste validacija prijave korisnika. Do toga dolazi kada korisnik pokrene aplikaciju. Zatim uz pomoć tastature unosi tražene podatke. Potrebno je da unese validno korisničko ime i šifru. U ovakvom scenariju može se dogoditi greška neispravne lozinke , te će korisnik biti obavešten o tome , i akcija će se ponavljati dok korisnik ne unese ispravan traženi podatak.



***Slika 32.*** *Filtriranje*

Cilj ovog testiranja jeste uspešno filtriranje/pretraga prodajnih objekata po ponuđenim parametrima koji su grad, naziv i adresa koje korisnik unosi u formu kraj liste prodajnih objekata. Ukoliko postoji objekat, korisnik će je dobiti kao rezultat u listi. Greška koja može da se dogodi u ovom scenariju je da korisnik unese podatke koji ne odgovaraju postojećim prodajnim objektima iz baze podataka ovog projekta, pa će dobiti praznu listu. Korisniku se pruža mogućnost da unese nove parametre ponovo u formu i tako dođe konačno do liste nekih prodajnih objekata koji odgovaraju unetim parametrima.

## 8.3 Integraciono testiranje



***Slika 33.*** *prikaz Bottom-up tehnike*

Integraciono testiranje vršimo nakon jediničnog testiranja. Ovde ćemo kombinovati različite jedinice i cilj nam je da otkrijemo defekte koji su izazvani integracijom modula. Još jedan cilj ovog testiranja jeste da se obezbedi da moduli rade kako se od njih očekuje nakon integracije.

Tehnika koju koristimo je "bottom-up" , i sa njom prvo vršimo testiranje najnižih modula / jedinica aplikacije, korak po korak, odozdo na gore. Prednost ove tehnike u odnosu na "top-down" tehniku jeste lakše detektovanje grešaka na nižim nivoima. Glavnu aplikaciju razbijamo na 6 modula prikazanih na slici, a vršimo 2 integraciona testiranja. Prvo testiramo modul "Filtriraj prodajne objekte" i "Kupovina proizvoda" putem drajvera odvojeno a zatim testiramo integrisano u "Pregled prodajnih objekata", testirano tako da prvo testiramo jedan sa glavnim modulom pa drugi, i onda ukoliko dođe do neke greške znamo da se nalazi u nekom od tih modula. Zatim taj modul integrišemo u "Aktivnosti" i tako integrisano bi se onda testiralo. Nakon toga bi se testirali iz ostali moduli iz tog nivoa "Logovanje". "Logovanje se integriše u samu aplikaciju gde bi se svaki modul ponaosob testirao sa konačnim sistemom i ukoliko bi došlo do neke greške znamo tačno o kom modulu je reč. Ovim završavamo celokupno integraciono testiranje.

# 9. Zaključak

Za realizaciju ovog projetka primenjeno je znanje koje je stečeno u okviru ovog predmeta a i onih koji su mu prethodili. Projekat efektivno prolazi kroz sve faze razvoja jednog softverskog sistema. Sam program je funkcionalan, i može se u budućnosti unaprediti funkcijama kao što su korpa, gde korisnik ima pregledniju sliku o svojoj porudžbini, odnosno kupovini, kao i boljoj i unapređenoj formi u kojoj mogu biti slike ili padajuće liste.

# 10. Literatura

1. Code Complete: A practical handbook of software construction, by S. McConnell, Microsoft Press, ISBN 0-7356-1967-0
2. Theory and Problems of Software Engineering – Schaum’s Outline Series, by David Gustafson, ISBN 0-07-137794-8