

OBJEKTNO ORIJENTISANA ANALIZA I DIZAJN

SPECIFIKACIJA PROJEKTA

NAZIV: 2nd TIME



ČLANOVI TIMA:

- Begović Amila
 - Kaleta Senija
 - Leka Elma
 - Panjeta Eldar
-

SADRŽAJ

SADRŽAJ	2
DETALJNA SPECIFIKACIJA PROGRAMA	5
OPIS	5
FUNKCIONALNOSTI	5
NEFUNKCIONALNI ZAHTJEVI	5
FUNKCIONALNI ZAHTJEVI	6
AKTERI	6
FUNKCIONALNOSTI AKTERA	7
ADMINISTRATOR	7
REGISTROVANI KORISNIK	7
NEREGISTROVANI KORISNIK (GOST)	8
VANJSKI SISTEM UPLATE	8
DOSTAVLJAČKA SLUŽBA	8
DIJAGRAM SLUČAJEVA UPOTREBE	9
SCENARIJ ZA SLUČAJEVE UPOTREBE	10
UPRAVLJANJE KORPOM	10
SCENARIJ	10
TOK DOGAĐAJA	11
DIJAGRAM AKTIVNOSTI	13
UPRAVLJANJE ARTIKLIMA	15
SCENARIJ	15
TOK DOGAĐAJA	16
DIJAGRAM AKTIVNOSTI	18
PRETRAGA ARTIKALA	19
SCENARIJ	19
TOK DOGAĐAJA	20
DIJAGRAM AKTIVNOSTI	23
REGISTRACIJA RAČUNA	24
SCENARIJ	24
TOK DOGAĐAJA	25
DIJAGRAM AKTIVNOSTI	26
LOG IN	27
SCENARIJ	27
TOK DOGAĐAJA	28
DIJAGRAM AKTIVNOSTI	29

USER INTERFACE	30
SOLID PRINCIPI	31
SINGLE RESPONSIBILITY PRINCIPLE - PRINCIP POJEDINAČNE ODGOVORNOSTI	31
OPEN CLOSED PRINCIPLE - OTVORENO ZATVOREN PRINCIP	31
LISKOV SUBSTITUTION PRINCIPLE - LISKOV PRINCIP ZAMJENE	32
INTERFACE SEGREGATION PRINCIPLE - PRINCIP IZOLIRANJA INTERFEJSA	32
DEPENDENCY INVERSION PRINCIPLE - PRINCIP INVERZIJE OVISNOSTI	32
CLASS DIJAGRAM	33
DIJAGRAM OBJEKATA	34
MVC DIJAGRAM	35
STRUKTURALNI PATERNI	36
ADAPTER PATERN	36
FAÇADE PATERN	37
DECORATOR PATERN	37
BRIDGE PATERN	38
COMPOSITE PATERN	39
PROXY PATERN	41
FLYWEIGHT PATERN	42
DIJAGRAM KLASE S STRUKTURALnim PATERNIMA	43
KREACIJSKI PATERNI	44
SINGLETON PATERN	44
FACTORY METHOD PATERN	44
BUILDER PATERN	45
PROTOTYPE PATERN	45
ABSTRACT FACTORY PATERN	46
DIJAGRAM KLASE S KREACIJSKIM PATERNIMA	47
PATERNI PONAŠANJA	48
STRATEGY PATTERN	48
STATE PATTERN	48
TEMPLATE METHOD PATTERN	49
ITERATOR PATTERN	49
OBSERVER PATTERN	50
CHAIN OF RESPONSIBILITY PATTERN	50
MEDIATOR PATTERN	51
DIJAGRAM KLASE S PATERNIMA PONAŠANJA	52

DIJAGRAM KLASE S PATERNIMA	53
ENTITY RELATIONSHIP DIJAGRAM	54
DIJAGRAM SEKVENCI	55
REGISTRACIJA RAČUNA	55
KREIRANJE NOVOG ARTIKLA	56
KUPOVINA	57
PRETRAGA	58
DIJAGRAM KOMPONENTI	59
DIJAGRAM PAKETA	59
DIJAGRAM RASPOREĐIVANJA	60
FINALNI KLAS I ER DIJAGRAM	61

DETALJNA SPECIFIKACIJA PROGRAMA

OPIS

"2nd TIME" je softver koji omogućava njegovim korisnicima da prodaju vlastitu odjeću/obuću, ali i da kupuju garderobu po veoma niskim cijenama. U našoj zemlji "second hand" prodavnice nisu najbolje organizovane, te imaju veoma mali izbor garderobe, dok u drugim zemljama "trifting" je postao veoma popularan način kupovine odjeće koji je jeftiniji i bolji za okoliš od kupovine nove garderobe. Kupnja jednog korištenog predmeta smanjuje njegov udio ugljika, otpada i vode za 82%. Uz sve pogodnosti za okoliš, ovim softverom će ljudi moći zaraditi i riješiti se garderobe koja im zauzima mjesto i stvara nerед.

FUNKCIONALNOSTI

Pri pokretanju softvera bit će omogućeno ulogovanje ili kreacija novog korisničkog računa u cilju kupovine i prodaje artikala, dok za pregled svih artikala logovanje neće biti obavezno. Na aplikaciji će se nalaziti preglednik s detaljnim filterima za lakše pretraživanje. Prilikom kreiranja korisničkog računa potrebno je dodati sve tačne informacije (ime, prezime, e-mail, grad, sliku) i onda će korisniku biti dostupna vlastita korpa. Registrovani korisnik će moći sam objavljivati vlastitu odjeću/obuću koju će moći detaljno opisati (vrsta, veličina, boja, marka...) pri čemu će biti neophodan upload slike sa vanjskog uređaja. Ukoliko namjerava da kupi neki ponuđeni artikal potrebno je da isti "smjesti" u korpu iz koje će se dalje vršiti kupovina. Svaki korisnik će imati vlastitu ocjenu, kao i recenzije koje su pisali drugi korisnici programa. Administrator će imati mogućnost da deaktivacije neprimjerenih korisničkih računa ili brisanje artikala.

NEFUNKCIONALNI ZAHTJEVI

- Korisnik mora imati više od 18 godina
- Postoje obavezni dijelovi koji se moraju unijeti pri kreaciji artikla
- Zaštita privatnosti korisničkih informacija
- Jednostavno korištenje aplikacije

FUNKCIONALNI ZAHTJEVI

- Registracija i uređivanje vlastitog korisničkog profila
- Mogućnost pregleda svih artikala
- Mogućnost sortiranja i filtriranja artikala prilikom pregleda
- Kreacija i uređivanje artikala s detaljnim opisima
- Pisanje recenzija za druge korisnike
- Prijavljivanje drugih korisničkih računa
- Kupovina se obavlja preko kartičnog plaćanja
- Kada se obavi kupovina, vlasniku artikla se šalje mail sa informacijama kupca i on dalje šalje pošiljku.
- Postojat će i progress bar za kreaciju korisničkog računa i kreaciju artikla
- Korištenje vanjskog uređaja za upload slike pri registraciji ili kreaciji artikla

AKTERI

- Administrator
- Registrovani korisnik
- Gost / neregistrovani korisnik
- Vanjski sistem uplate
- Dostavljач

FUNKCIONALNOSTI AKTERA

ADMINISTRATOR

Administrator će imati sljedeće dozvole:

- Pregled žalbi
- Mogućnost da deaktivira korisničke račune koje su prijavili drugi korisnici i/ili korisnike koji objavljaju neprimjereno sadržaj
- Brisanje artikala koji predstavljaju neprimjereno sadržaj
- Uklanjanje komentara

REGISTROVANI KORISNIK

Registrovani korisnik će imati sljedeće dozvole:

- Pregled i uređivanje vlastitog korisničkog računa
- Objavljanje vlastitih artikala uz detaljan opis, te prodaju istih. Ukoliko korisnik namjera da kreira artikal potrebno je da doda sljedeće informacije o njemu:
 - Naziv artikla
 - Sliku artikla
 - Spol za koji je artikal namijenjen (M, Ž)
 - Dob za koju je artikal namijenjen (za bebe, djecu, odrasle..)
 - Vrstu artikla (majica, hlače, obuća..)
 - Boja artikla (crna, crvena, zelena..)
 - Veličina (za odjeću - S, M, L.. za obuću - 36, 37, 38..)
 - Količina

-
- Opis
 - Imat će pristup cijelom pregledniku artikala
 - Imat će vlastitu korpu u koju će postavljati articke koje namjerava kupiti i iz koje će se dalje vršiti kupovina
 - Bit će mu omogućeno ocjenjivanje i pisanje recenzija za druge korisnike

NEREGISTROVANI KORISNIK (GOST)

- Neregistrovani korisnik će imati mogućnost pretrage i pregleda svih artikala na stranici, ali za kupovinu i prodaju će morati napraviti vlastiti korisnički račun. Pri kreiranju korisničkog računa potrebno je da se unese:
 - Ime i prezime
 - Korisničko ime
 - Lozinku
 - E-mail
 - Dob
 - Broj telefona
 - Adresa
 - Slika
 - Broj kreditne kartice/PayPal račun

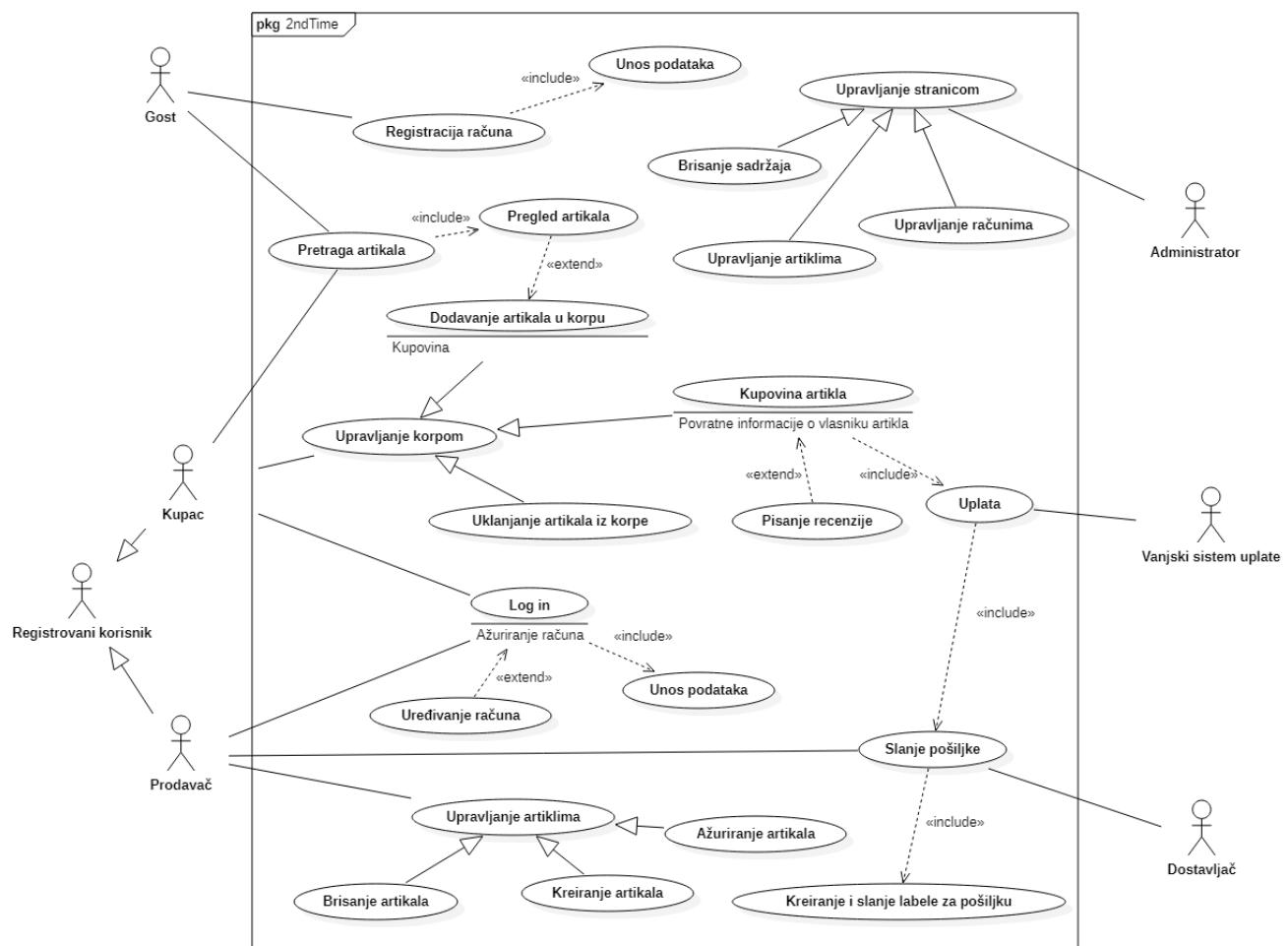
VANJSKI SISTEM UPLATE

- Aplikacija podržava vanjski princip uplate: PayPal, plaćanje karticom - sistem koji odobrava ili odbija transakciju u slučaju da kupac odabere kupovinu artikla

DOSTAVLJAČKA SLUŽBA

- Dostavljач – sistem koji nakon odobravanja narudžbe preuzima osnovne informacije o kupcu potrebne za dostavu narudžbe i jedinstven identifikacioni broj narudžbe po kojem treba izvršiti dostavu.

DIJAGRAM SLUČAJEVA UPOTREBE



SCENARIJ ZA SLUČAJEVE UPOTREBE

Najkompleksnije funkcionalnosti sistema:

- Upravljanje korpom
- Upravljanje artiklima
- Pretraga artikala
- Registracija
- Log-in

UPRAVLJANJE KORPOM

SCENARIJ

Naziv	Upravljanje korpom
Opis	Korisnik koji posjeduje registrovani račun može da upravlja svojom potrosačkom korpom (pregled artikala, uklanjanje artikala, dodavanje artikla ili da se odluči za kupnju istih)
Vezani zahtjevi	
Preduslovi	Posjedovanje korisnickog racuna
Posljedice – uspješan završetak	Pristup korpi i njenim mogućnostima
Posljedice – neuspješan završetak	Odbijen pristup korpi uslijed greske ili ako je korisnik prijavljen kao gost
Primarni akteri	Registrovani korisnik (kupac)
Ostali akteri	Registrovani korisnik (prodavac)
Glavni tok	Prijavljeni korisnik ima pristup svim mogućnostima korpe
Proširenja/Alternative	<ul style="list-style-type: none">• Brisanje artikala• Dodavanje novih artikala• Kupanja artikala iz korpe

TOK DOGAĐAJA

1. Uspješni završetak

Kupac	2nd Time
1. Registrovani korisnik pristupa interface-u korpe	2. Prikaz stanja korpe
3. Pregled stanja korpe	

2. Alternativni tok 1: Korisnik zeli da izbaci neke artikle iz korpe

Kupac	2nd Time
1. Uklanjanje artikala	2. Sistem traži potvrdu da bi uklonio artikal
3. Prihvatanje/odbijanje zahtjeva	

3. Alternativni tok 2: Korisnik zeli da stavi nove artikle u korpe

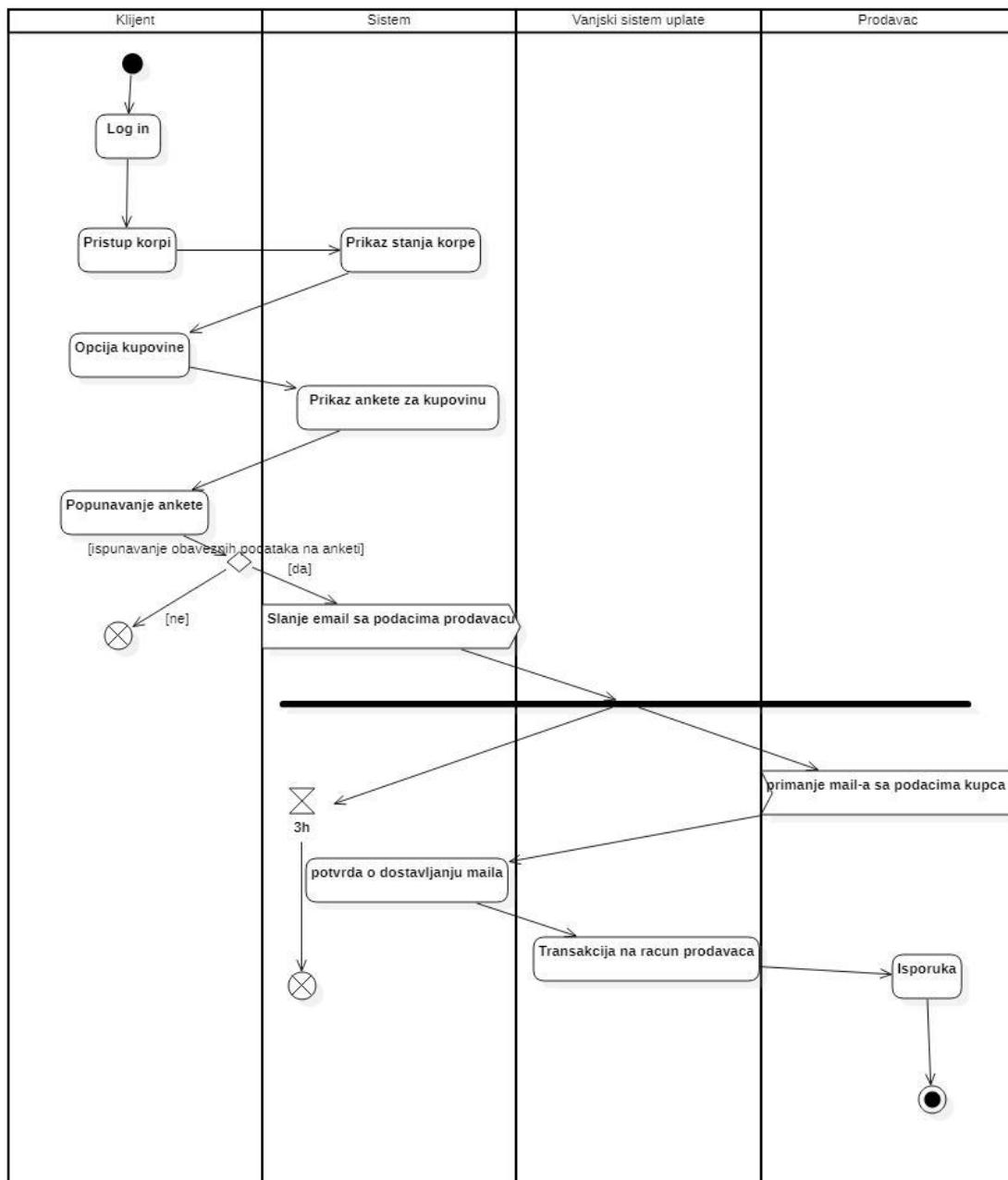
Kupac	2nd Time
1. Unos filtera za pretragu artikla	2. Prikaz artikala koji odgovaraju filteru
3. Korisnik bira opciju "stavi u korpu"	

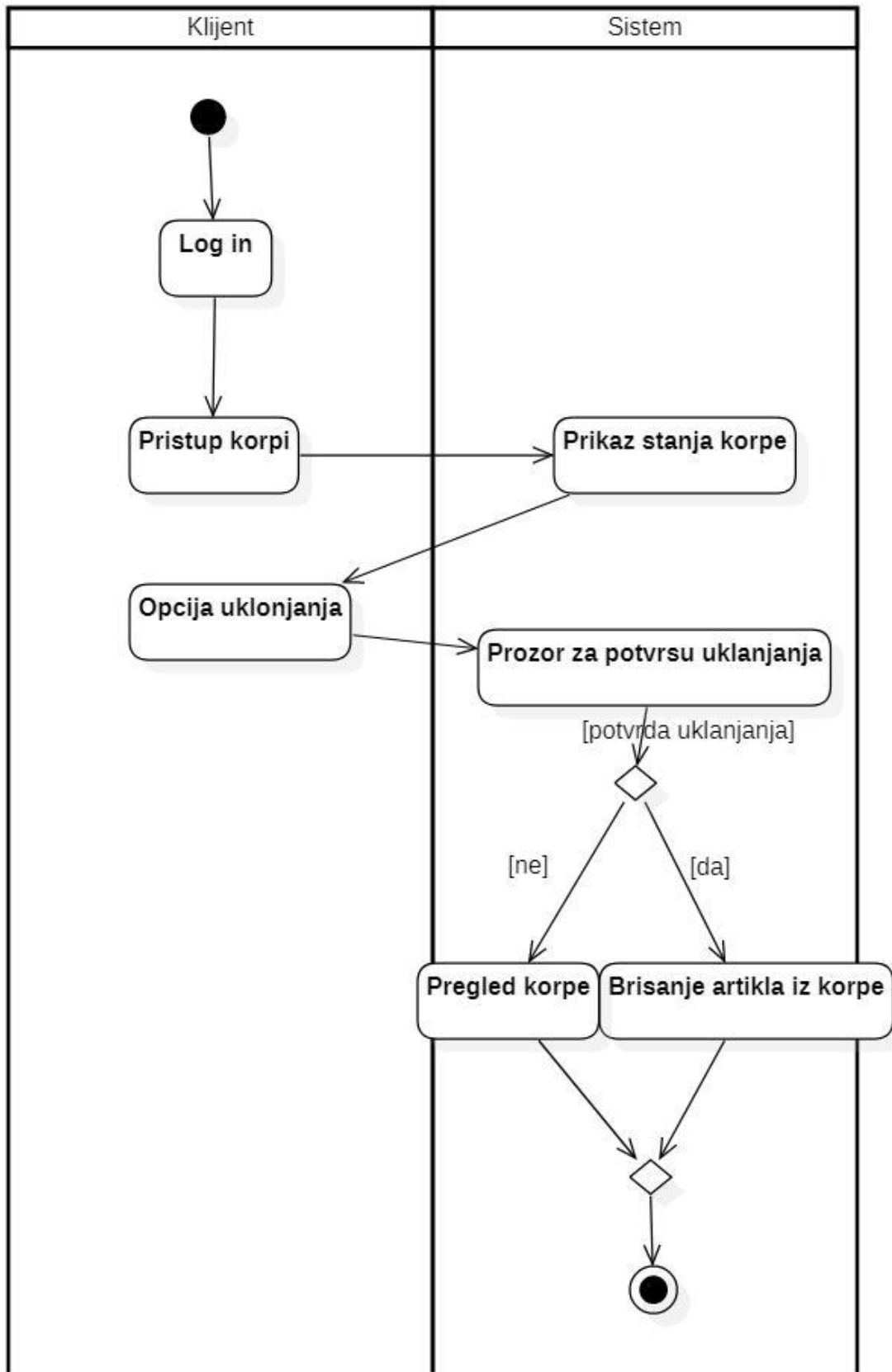
4. Alternativni tok 3: Korisnik zeli da kupi artikal iz korpe

Kupac	2nd Time	Vanjski sistem uplate	Prodavac
1. Za odabrani artikal bira opciju kupnje	2. Otvara prozor za kupnju gdje se traži da kupac popuni		

	anketu gdje je obavezno unijeti adresu na koju se želi obaviti isporuka i broj kartice		
3. Popunjavanje ankete vezane za kupnju	4. Podaci kupca se salju prodavaču artikla	5. Transakcija novca sa računa kupca na račun prodavača	6. Isporuka pošiljke

DIJAGRAM AKTIVNOSTI





UPRAVLJANJE ARTIKLIMA

SCENARIJ

Naziv	Upravljanje artikloma
Opis	Registrovani korisnik ima opciju da objavljuje ili uređuje svoje sopstvene articke i nudi ih na prodaju ostalim korisnicima. Kad god kreira neki artikal moći će ga detaljno opisati i naknadno ažurirati. Korisnik će moći izbrisati sve do tada kreirane articke.
Vezani zahtjevi	
Preduslovi	Korisnik mora biti registrovan na sajt. Gost nema prava za objavljivanje artikala.
Posljedice – uspješan završetak	Korisnik je uspješno unio sve podatke vezane za artikal i artikal se nalazi na sajtu. Korisnik je uspješno obrisao artikal.
Posljedice – neuspješan završetak	Prilikom objavljivanja, brisanja ili uređivanja artikla došlo je do greške i akcija nije završena.
Primarni akteri	Registrovani korisnik - prodavač
Ostali akteri	<ul style="list-style-type: none">• Registrovani korisnik - kupac• Neregistrirani korisnik - gost• Administrator
Glavni tok	Registrovani korisnik je unio podatke, te uspješno kreirao, ažurirao ili obrisao artikal.
Proširenja/Alternative	<ul style="list-style-type: none">• Kreiranje artikla• Brisanje artikla• Ažuriranje artikla

TOK DOGAĐAJA

1. Uspješni završetak

Registrirani korisnik	Sistem
1. Registrirani korisnik pristupa svojim artiklima	2. Prikaz artikala
3. Pregled artikala	

2. Uspješan (alternativni) završetak - dodavanje artikla

Registrirani korisnik	Sistem
1. Kreiranje novog artikla	2. Prikaz interfejsa za kreaciju artikla
3. Popunjavanje podataka za artikal	4. Artikal uspješno kreiran i nalazi se na sajtu

3. Uspješan (alternativni) završetak - brisanje artikla

Registrirani korisnik	Sistem
1. Brisanje postojećeg artikla	2. Sistem traži potvrdu da bi uklonio artikal
3. Prihvatanje/odbijanje zahtjeva	4. Artikal obrisan sa sajta

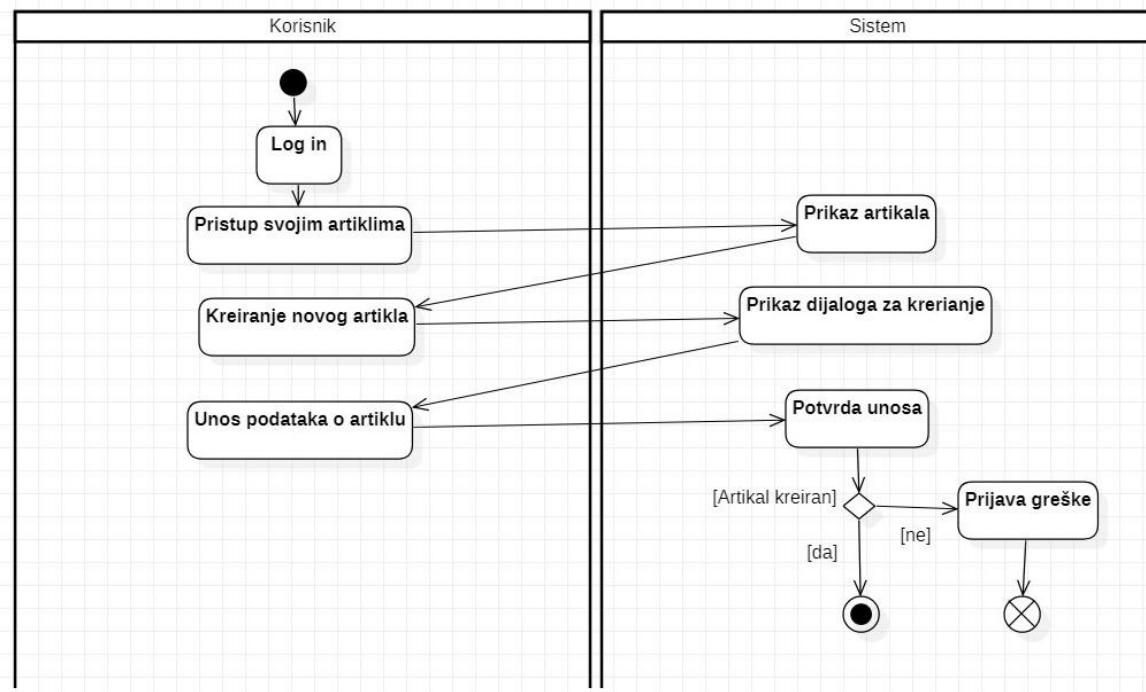
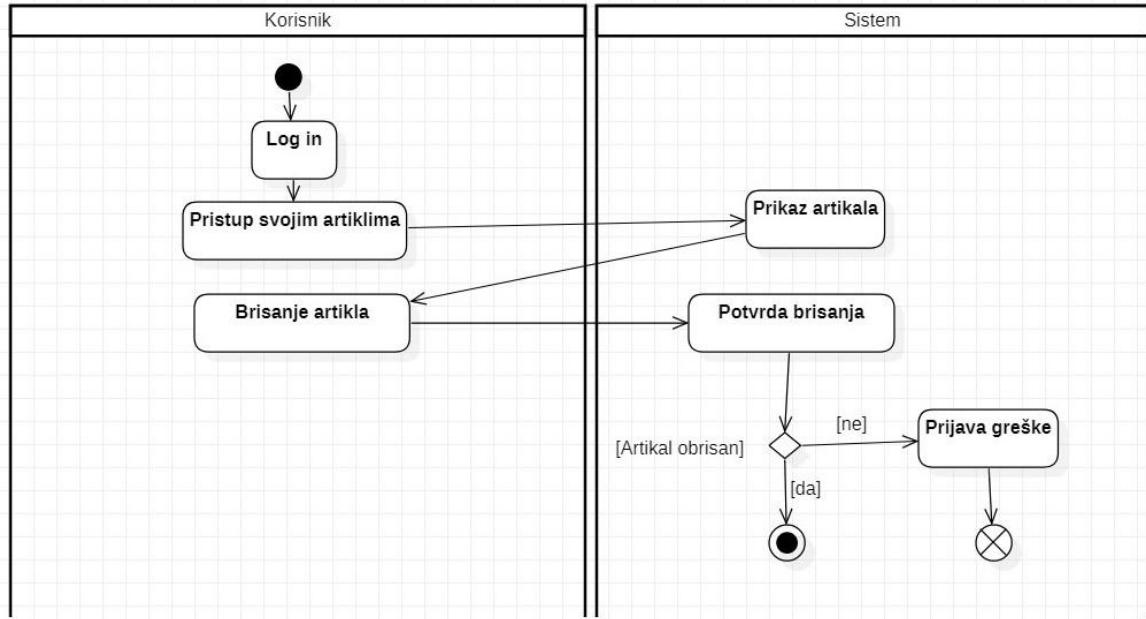
4. Uspješan (alternativni) završetak - ažuriranje artikla

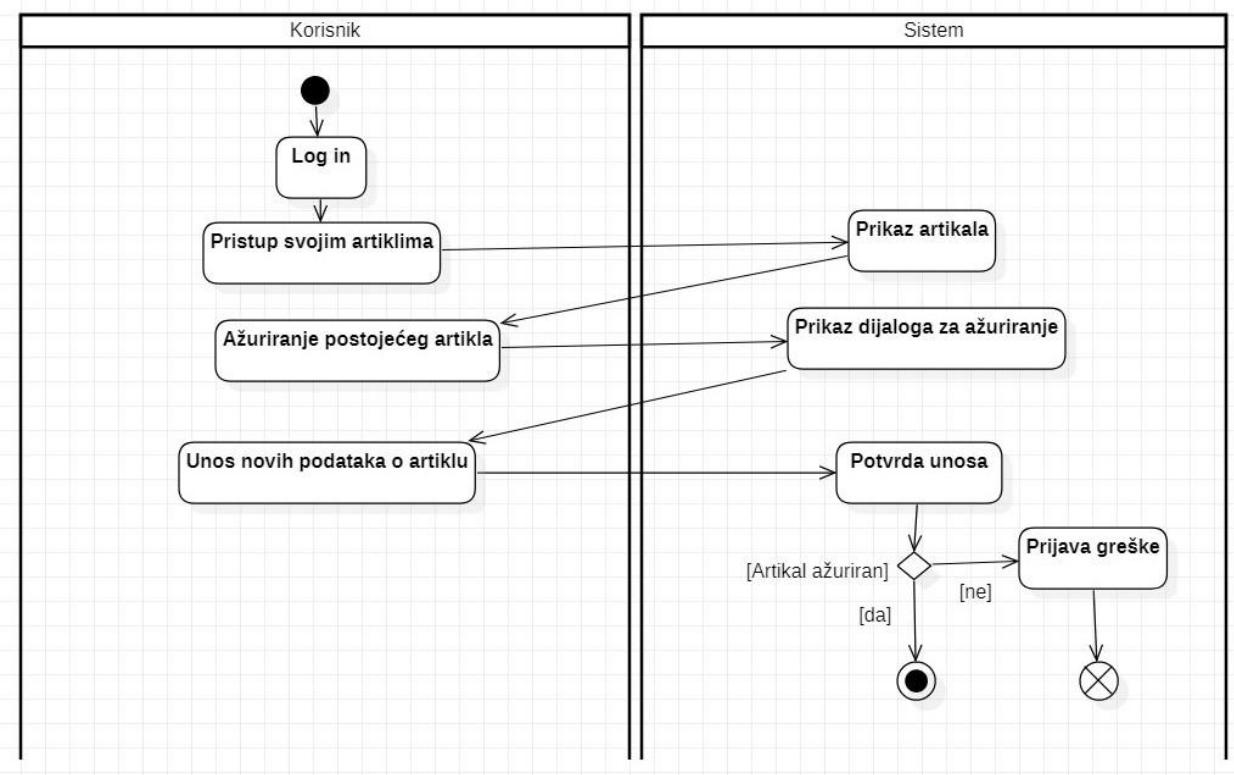
Registrirani korisnik	Sistem
1. Ažuriranje postojećeg artikla	2. Prikaz dijaloga za ažuriranje artikla
3. Popunjavanje podataka za artikal	4. Artikal uspješno ažuriran

5. Neuspješan završetak - neuspješna kreacija artikla

Registrirani korisnik	Sistem
1. Kreiranje novog artikla	2. Prikaz interfejsa za kreaciju artikla
3. Popunjavanje podataka za artikal	4. Nastala je greška. Artikal nije kreiran

DIJAGRAM AKTIVNOSTI





PRETRAGA ARTIKALA

SCENARIJ

Naziv	Pretraga artikala
Opis	Korisnik I gost pretražuju articke u sistemu
Vezani zahtjevi	
Preduslovi	
Posljedice – uspješan završetak	Prikazuje se lista svih articala koji zadovoljavaju uslove korisnika
Posljedice – neuspješan završetak	Došlo je do greške prilikom pretrage, dodavanja u korpu ili prijave articla te akcija nije završena
Primarni akteri	Registrirani korisnik (kupac) i gost

Ostali akteri	Administrator
Glavni tok	Korisnik (registrovani i neregistrovani) može da pretražuje sve articke u sistemu na način da ih filtrira po određenim ili svim kategorijama (spol, dob, veličina, vrsta artikla, boja, cijena) nakon čega može da pregledava svaki artikal detaljnije
Proširenja/Alternative	<ul style="list-style-type: none"> • Dodavanje željenog artikla u korpu • Prijava artikla ili korisnika

TOK DOGAĐAJA

1. Uspješni završetak

Korisnik	2nd Time
1. Korisnik bira da li želi muški ili ženski artikal	
2. Korisnik bira koju vrstu artikala želi pretraživati	
3. Korisnik filtrira po ostalim kriterijima koje želi	
	4. Izlistava sve articke koje zadovoljavaju kriterije korisnika
5. Korisnik bira način sortiranja za prikazane articke	
	6. Sortira articke po odabranom kriteriju
7. Korisnik može da izabere neki artikal kako bi ga detaljno pogledao	
	8. Detaljni prikaz odabranog artikla

2. Uspješni (alternativni) završetak

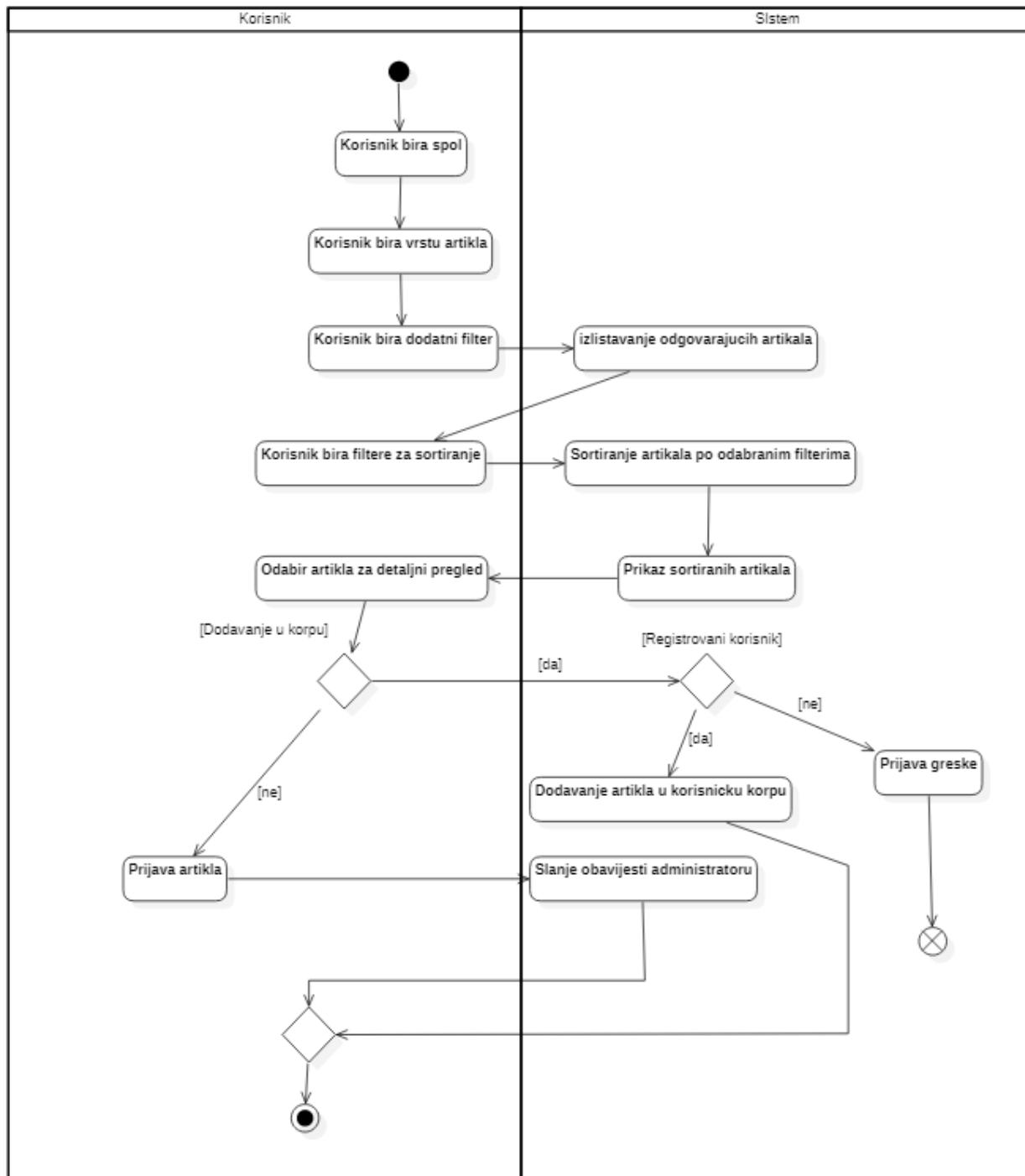
Korisnik	2nd Time
1. Korisnik bira da li želi muški ili ženski artikal	
2. Korisnik bira koju vrstu artikala želi pretraživati	
3. Korisnik filtrira po ostalim kriterijima koje želi	
	4. Izlistava sve artikle koje zadovoljavaju kriterije korisnika
5. Korisnik bira način sortiranja za prikazane articke	
	6. Sortira articke po odabranom kriteriju
7. Korisnik može da izabere neki artikal kako bi ga detaljno pogledao	
	8. Detaljni prikaz odabranog artikla
9. Korisnik bira da doda artikal u svoju korpu	
	10. Dodaje odabrani artikal u korpu korisnika

Tok dogadjaja 1.3:

Korisnik	2nd Time
1. Korisnik bira da li želi muški ili ženski artikal	
2. Korisnik bira koju vrstu artikala želi pretraživati	

3. Korisnik filtrira po ostalim kriterijima koje želi	
	4. Izlistava sve articke koje zadovoljavaju kriterije korisnika
5. Korisnik bira način sortiranja za prikazane articke	
	6. Sortira articke po odabranom kriteriju
7. Korisnik može da izabere neki artikal kako bi ga detaljno pogledao	
	8. Detaljni prikaz odabranog artikla
9. Korisnik prijavljuje artikal ili korisnika te popunjava razlog prijave	
	10. Registruje prijavu i šalje obavijest administratoru da pregleda prijavu

DIJAGRAM AKTIVNOSTI



REGISTRACIJA RAČUNA

SCENARIJ

Naziv	Registracija računa
Opis	Ukoliko je osoba prijavljena kao gost, ali želi pristupiti kupovini ili uploadu artikala no nema prethodno kreiran korisničku račun potrebno je da kreira isti sa neophodnim informacijama kao što su: ime, prezime, mail, šifra, dob, broj telefona, adresu te broj kreditne kartice. Ukoliko su uneseni podaci validni osoba je uspješno kreirala račun.
Vezani zahtjevi	/
Preduslovi	Korisnik ne posjeduje korisnički račun.
Posljedice – uspješan završetak	Korisnik je uspješno unio podatke koji su uspješno validirani te ima pristup svim pogodnostima registrovanog korisnika.
Posljedice – neuspješan završetak	Korisnik je neuspješno unio svoje podatke ili je došlo do neke druge greške prilikom registriranja i samim time nije kreiran korisnički račun.
Primarni akteri	Neregistrovani korisnik/Gost
Ostali akteri	Registrovani korisnik
Glavni tok	Korisnik nakon uspješnog registrovanja u sistem ima sve beneficije registrovanog korisnika (kupca i prodavača).
Proširenja/Alternative	

TOK DOGAĐAJA

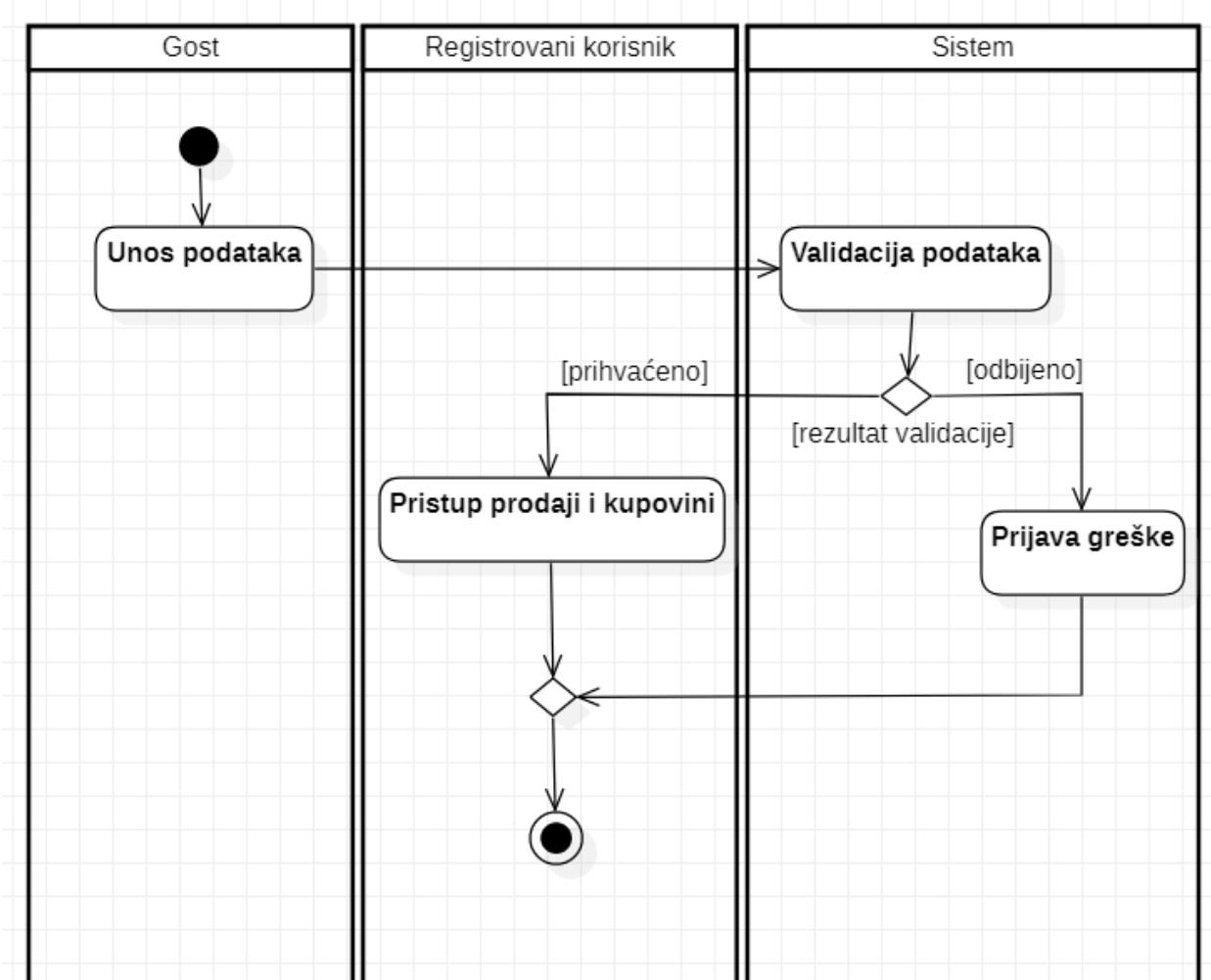
1. Uspješni završetak

Gost	Sistem	Registrovani korisnik
1. Popunjavanje podataka o korisniku	2. Provjera podataka	
		3. Korisnik je registrovan, ima pristup beneficijama registrovanog računa

2. Neuspješan završetak

Gost	Sistem
1. Popunjavanje podataka okorisniku	2. Validacija podataka
	3. Doslo je do greške prilikom validacije, pristup računu nije odobren.

DIJAGRAM AKTIVNOSTI



LOG IN

SCENARIJ

Naziv	Log in
Opis	Ukoliko korisnik namjerava da pristupi pretrazi i pregledu artikala s namjerom da obavi kupovinu odabralih artikala ili ukoliko namjerava da objavi svoj artikal na prodaju, prvo je potrebno da se loguje na svoj postojeći korisnički račun. Log in će se obavljati preko jedinstvenog mail-a/korisničkog imena i lozinke.
Vezani zahtjevi	/
Preduslovi	Korisnik posjeduje registrovani račun. Korisnik mora unijeti tačnu mail adresu ili korisničko ime i tačnu lozinku za svoj račun.
Posljedice – uspješan završetak	Korisnik je uspješno unio podatke za svoj račun i time je pristupio svom računu te dobio beneficije koje posjeduje registrovani račun.
Posljedice – neuspješan završetak	Korisnik je neuspješno unio svoje podatke ili je došlo do neke druge greške prilikom logovanja i samim time nije omogućen pristup korisničkom računu i njegovim pogodnostima.
Primarni akteri	Gost, Registrovani korisnik
Ostali akteri	/
Glavni tok	Korisnik nakon uspješnog logiranja u sistem ima sve beneficije registrovanog korisnika (kupca i prodavača).
Proširenja/Alternative	Uređivanje računa

TOK DOGAĐAJA

1. Uspješni završetak

Gost	Sistem	Registrirani korisnik
1. Popunjavanje podataka o računu	2. Validacija podataka	
		3. Pristup beneficijama registrovanog računa

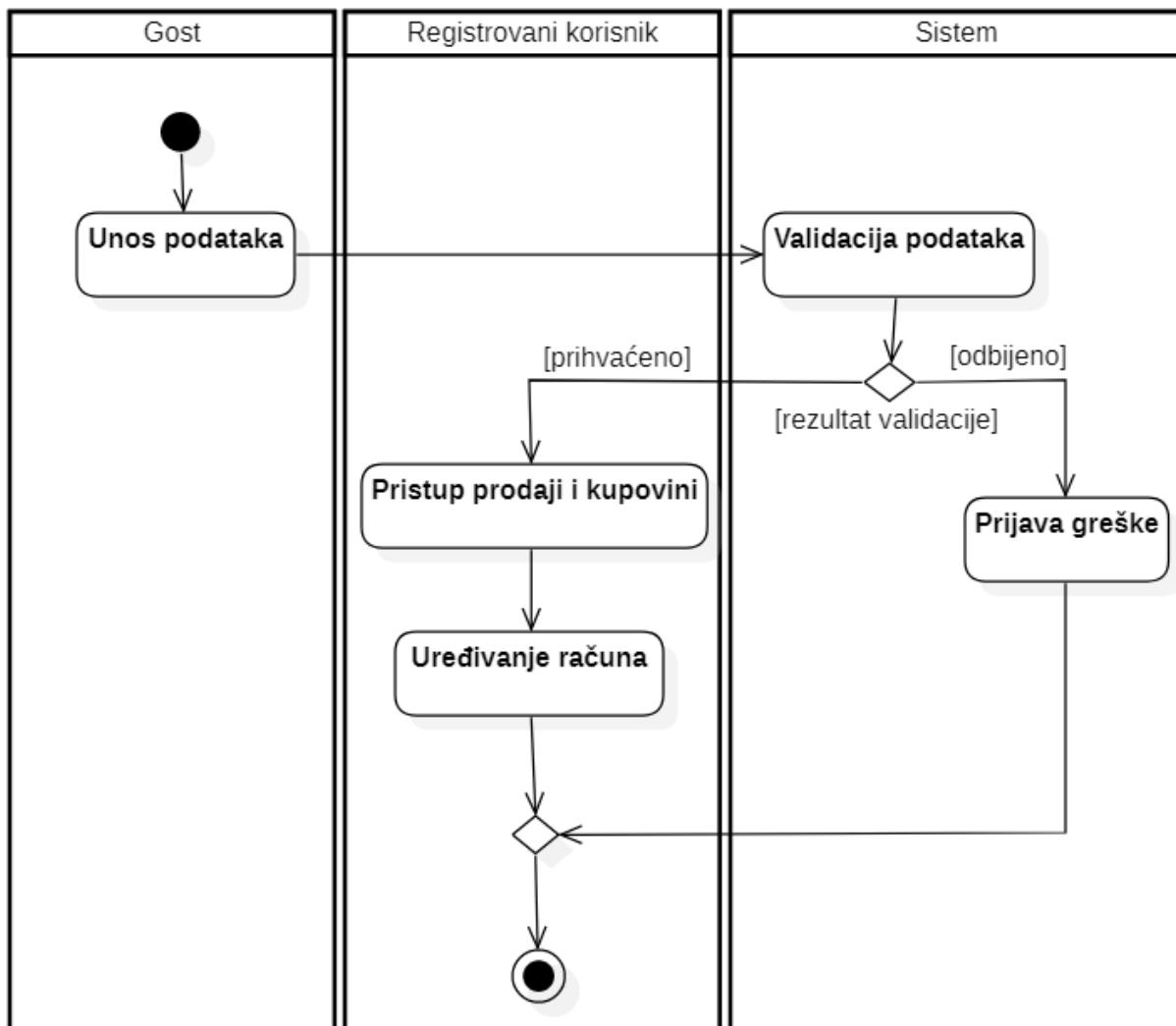
2. Uspješan (alternativni) završetak

Gost	Sistem	Registrirani korisnik
1. Popunjavanje podataka o računu	2. Validacija podataka	
		3. Uređivanje računa
		4. Pristup beneficijama registrovanog računa

3. Neuspješan završetak

Gost	Sistem
1. Popunjavanje podataka o računu	2. Validacija podataka
	3. Doslo je do greske prilikom validacije, pristup racunu nije odobren.

DIJAGRAM AKTIVNOSTI



USER INTERFACE



Više na sljedećem [linku](#).

SOLID PRINCIPI

SINGLE RESPONSIBILITY PRINCIPLE - PRINCIP POJEDINAČNE ODGOVORNOSTI

Princip pojedinačne odgovornosti zahtijeva da svaka klasa ima samo jednu odgovornost, odnosno da klasa vrši samo jedan tip akcija kako ne bi ovisila o velikom broju konkretnih implementacija.

Ukoliko pogledamo dijagram klasa vidimo da većina klase implementira samo gettere i settere pripadajućih atributa ili radi sa listama koje posjeduje kao attribute što omogućava da ukoliko se promjene i dese one se dešavaju na jednom mjestu. Uzmimo primjer klasu User. Svi atributi ove klase su geteri i seteri te dodavanje nove mogućnosti neće zahtijevati izmjenu nigdje osim u toj klasi, što nam govori da je ovaj princip ispunjen.

OPEN CLOSED PRINCIPLE - OTVORENO ZATVOREN PRINCIP

Otvoreno zatvoren princip zahtijeva da klasa koja koristi neku drugu klasu ne treba biti modificirana pri uvodenju novih funkcionalnosti, ili pri potrebi za mijenjanjem druge klase

Sve klase imaju veliki broj metoda te dodavanje novih neće uticati na promjenu same klase i logike koja stoji iza nje odnosno dodavanje metoda neće zahtijevati izmjenu čitave klase. Većina klase sadrži kolekcije drugih klasa tako da izmjena u nekoj klasi neće implicirati promjenu druge.

Primjer ovog principa možemo vidjeti kroz klasu Cart koja sadrži listu Product koja je u biti apstraktna klasa iz koje su izvedene klase Accessorie, Clothing i Shoes. Ukoliko bismo odlučili dodati novu klasu koja će biti izvedena iz Product npr. SportsWear nikakve promjene ne bi bile potrebne u klasi Cart. Nasuprot tome da Product nije apstraktna klasa te da u Cart imamo kolekcije Shoes, Accessories i Clothing, pa ukoliko kreiramo novu klasu npr. SportsWear neophodna bi bila i izmjena u Cart što bi narušilo open-close princip.

LISKOV SUBSTITUTION PRINCIPLE - LISKOV PRINCIP ZAMJENE

Liskov princip zamjene zahtijeva da nasljeđivanje bude ispravno implementirano, odnosno da je na svim mjestima na kojima se koristi osnovni objekat moguće iskoristiti i izvedeni objekat a da takvo nešto ima smisla.

Dijagram posjeduje samo jednu apstraktnu klasu, a to je Product iz koje su izvedene klase Shoes, Accessories i Clothing. Jasno je da se na svakom mjestu, gdje je potrebno koristiti neku od ove tri klase, koristi apstraktna klasa iz koje su iste izvedene i da je upotreba bilo koje od izvedenih klasa na mjestu apstraktne moguće.

Primjer upotrebe apstraktne klase jeste recimo u klasi Cart gdje se koristi lista Products. U toj listi je moguće čuvati bilo koju od izvedenih klasa zbog čega je ovaj princip i ispunjen.

INTERFACE SEGREGATION PRINCIPLE - PRINCIP IZOLIRANJA INTERFEJSA

Princip izoliranja interfejsa zahtijeva da i svi interfejsi zadovoljavaju princip pojedinačne odgovornosti, odnosno da svaki interfejs obavlja samo jednu vrstu akcija.

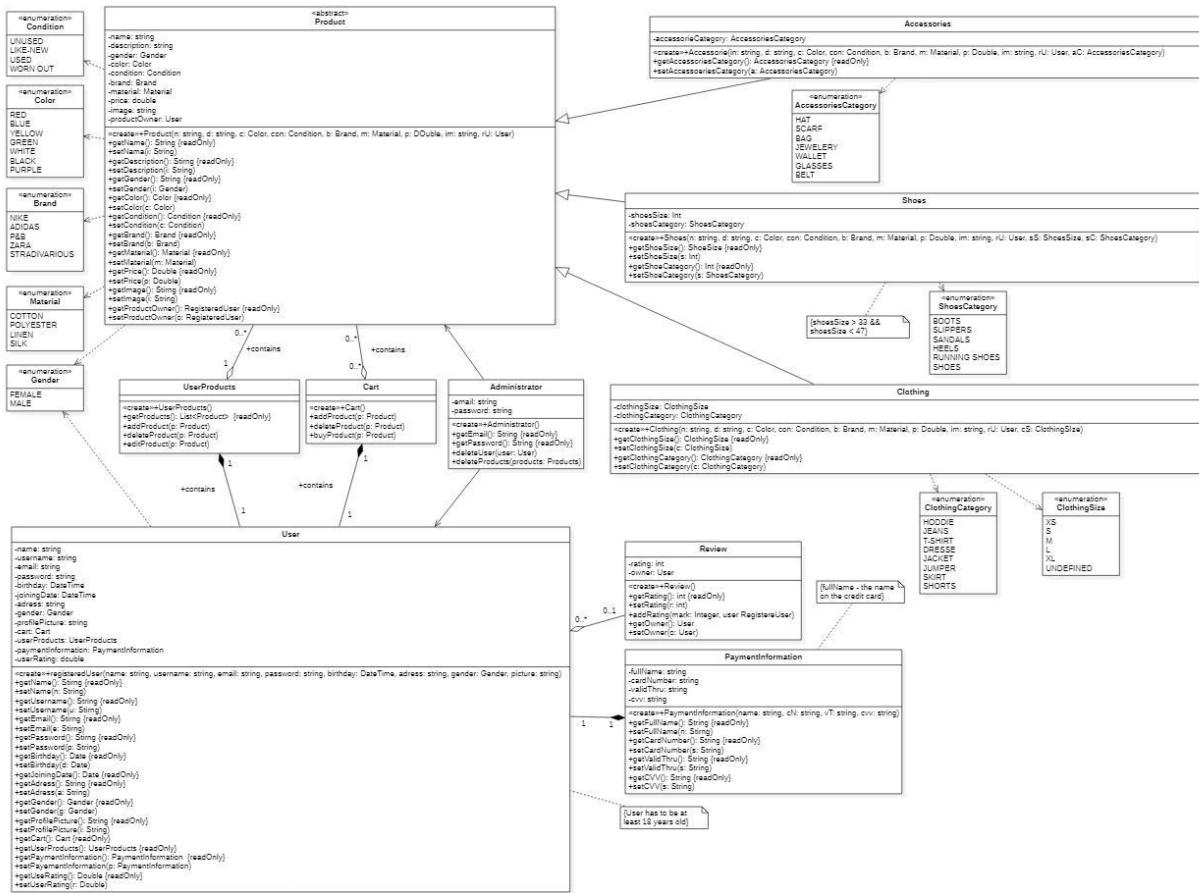
U našem sistemu ne postoje interfejsi, o ovome principu ne možemo ni diskutovati te smatramo da je ispoštovan.

DEPENDENCY INVERSION PRINCIPLE - PRINCIP INVERZIJE OVISNOSTI

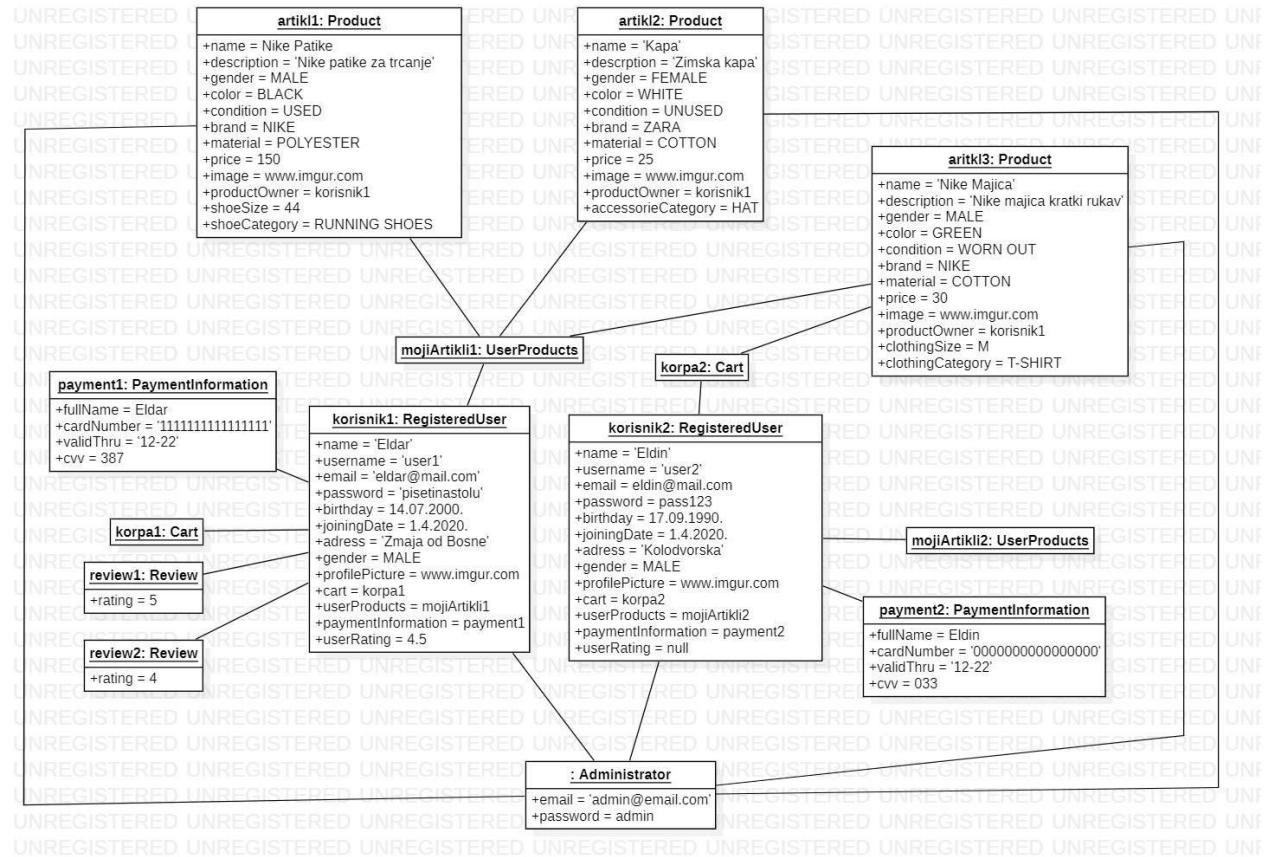
Princip inverzije ovisnosti zahtijeva da pri nasljeđivanju od strane više klase bazna klasa uvijek bude apstraktna. Razlog za ovo je što je teško koordinisati veliki broj nasljeđenih klasa i konkretnu baznu klasu ukoliko ista nije apstraktna, a da pritom kod bude čitak i jednostavan za razumijevanje.

Jedina klasa kod koje se javlja nasljeđivanje je klasa Product. Ona je navedena kao abstract odnosno nemoguće ju je instancirati. Iz nje su izvedene klase Shoes, Accessories i Clothing koje je moguće instancirati, stoga se smatra da je ovaj princip ispoštovan.

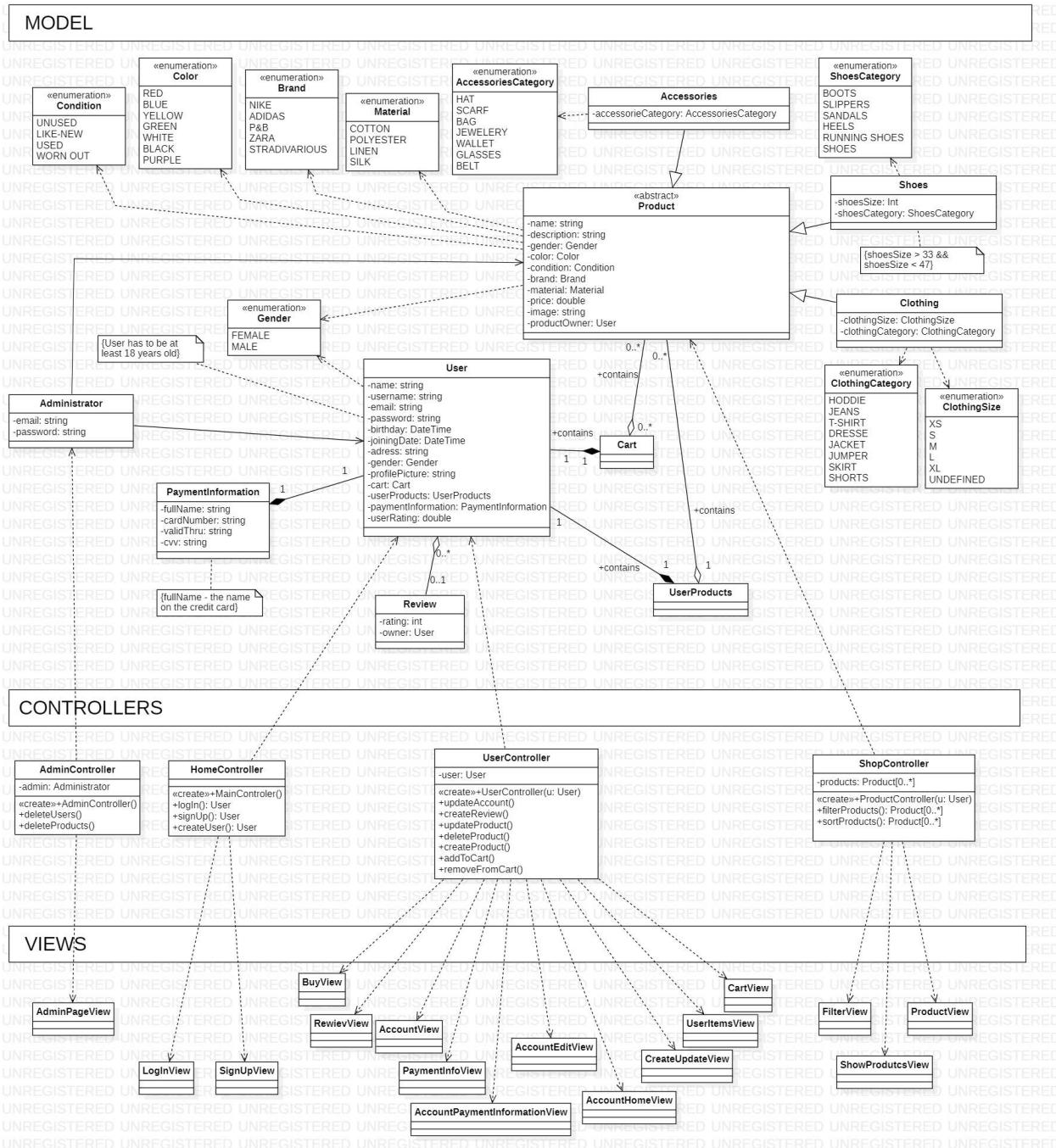
CLASS DIJAGRAM



DIJAGRAM OBJEKATA



MVC DIJAGRAM



STRUKTURALNI PATERNI

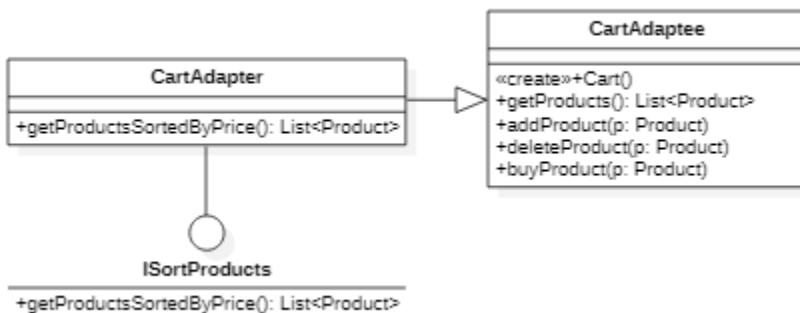
ADAPTER PATTERN

Adapter pattern služi da se postojeći objekat prilagodi za korištenje na neki novi način u odnosu na postojeći rad, bez mijenjanja same definicije objekta. Na taj način obezbeđuje se da će se objekti i dalje moći upotrebljavati na način kako su se dosad upotrebljavali, a u isto vrijeme će se omogućiti njihovo prilagođavanje novim uslovima.

U ovom projektu Adapter pattern ćemo omogućiti tako što ćemo u klasi Cart omogućiti prikaz Products sortiranih po cijeni. Tačnije omogućit ćemo da metoda getProducts() vraća sve proizvode ali sortirane po cijeni.

Klasu Cart preimenovat ćemo u CartAdaptee te ćemo dodati klasu CartAdapter koja će implementirati interfejs ISortProducts koja posjeduje metodu getProductsSortedByPrice(). Ova metoda će u sebi pozivati metodu klase CartAdaptee getProducts() međutim mijenjat će je tako da vraća sortiranu listu po cijeni proizvoda.

Ovim smo postigli da je klasa Cart adaptirana i nadograđena. Klasa Cart nije promijenjena samo je unapriđena novim interfejsom. Ovo ne mora biti jedina nadogradnja. Možemo omogućiti sortiranje Products po raznim kriterijima kao što su condition, color i slično. U nastavku je prikazano kako je zamišljena primjena ovog paterna na naš dijagram klase.



FACADE PATERN

Fasadni patern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema. Klijenti vide samo fasadu, odnosno krajnji izgled objekta, dok je njegova unutrašnja struktura skrivena. Na ovaj način smanjuje se mogućnost pojavljivanja grešaka jer klijenti ne moraju dobro poznavati sistem kako bi ga mogli koristiti.

Facade Patern nećemo implementirati međutim kada bi smo to željeli odabrali bi dio sistema koji je komplikovan no korisnik ne treba znati šta se dešava "ispod haube". Jedan od najkomplikovanijih dijelova ovog projekta jeste dobavljanje informacija iz baze podataka. To zahtjeva veći broj upita koji odgovaraju željenom pozivu. Kako korisnik nema potrebe da zna kako izgledaju ti upiti nego ga samo interesuje krajnji rezultat to je idealno mjesto za korištenje ovog paterna. Primjer kako bi to izgledalo u kodu je dato u nastavku:

```
public class FasadaProduct
{
    public List<Product> getUserProductsForUser(email: String, username: String);
    public List<Products> getCartProductsForUser(email: String, username: String);
    public List<Review> getReviewsForUser(email: String, username: String);
    ...
}
```

DECORATOR PATERN

Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu). Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti različito dekoriranje objekata (tj. dodavanje različitih detalja), te se na taj način pojednostavljuje i rukovanje objektima klijentima, i samo implementiranje modela objekata.

Ovaj Pattern nećemo implementirati no možemo pričati o hipotetskoj implementaciji. Naime bilo bi veoma zgodno iskoristiti ovaj patern na klasi Product da bi se omogućilo editovanje slika Producta koje korisnik želi postaviti. Da bi to omogućili potrebno je izdvojiti klasu Slika kako bi ona mogla implementirati interfejs ISlika koji posjeduje metode dajSliku

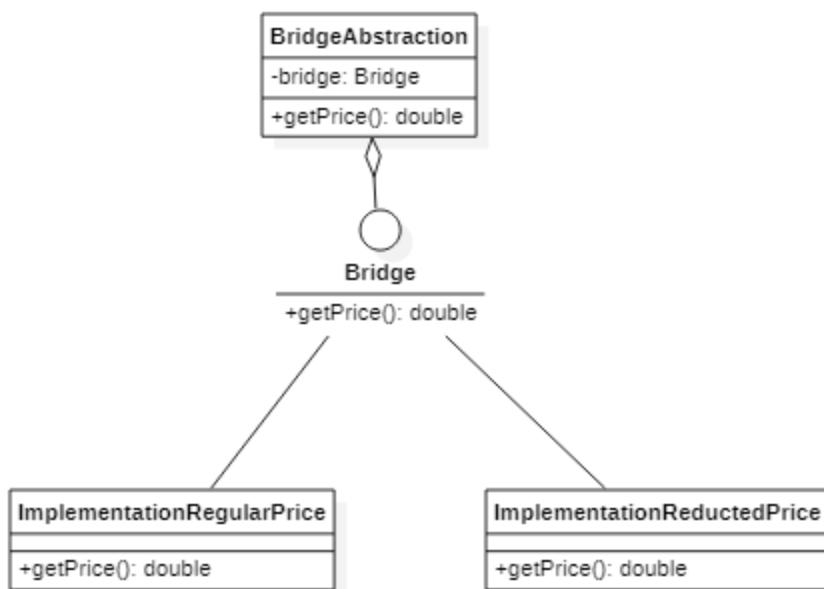
koja vraca Sliku te postaviSliku. U ovom slučaju će klasa Slika biti Component dok bi Decorator klasa mogla biti klasa SlikaFilteri koja će omogućiti da se slika edituje primjenom nekog filtera. Takoder vrlo bi se lahko moglo omogućiti i druge vrste manipulacije nad slikom kao što su rotiranje, rezanje i slično.

BRIDGE PATERN

Bridge patern služi kako bi se apstrakcija nekog objekta odvojila od njegove implementacije. Ovaj patern veoma je važan jer omogućava ispunjavanje Open-Closed SOLID principa, odnosno uz poštivanje ovog paterna omogućava se nadogradnja modela klase u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama.

Ovaj Pattern nećemo implementirati no ukoliko bismo se odlučili na implementaciju najbolje mjesto za iskoristiti ovaj patern jeste na klasi Product. Ukoliko bi željeli dati određeni popust na Product to bismo uradili preko Bridge paterna.

Da bi ovo bilo izvodivo odnosno da bi mogli nesmetano izračunati novu cijenu nakon sniženja potrebno je dodati interfejs Bridge koji ima metodu getCijena(). Osim toga omogućili bi dvije implementacije. Jedna bi vraćala regularnu cijenu Product, dok bi druga implementacija vraćala cijenu sniženu u zavisnosti od popusta. Na taj način smo osigurali da će već postojeći sistem regularno raditi.



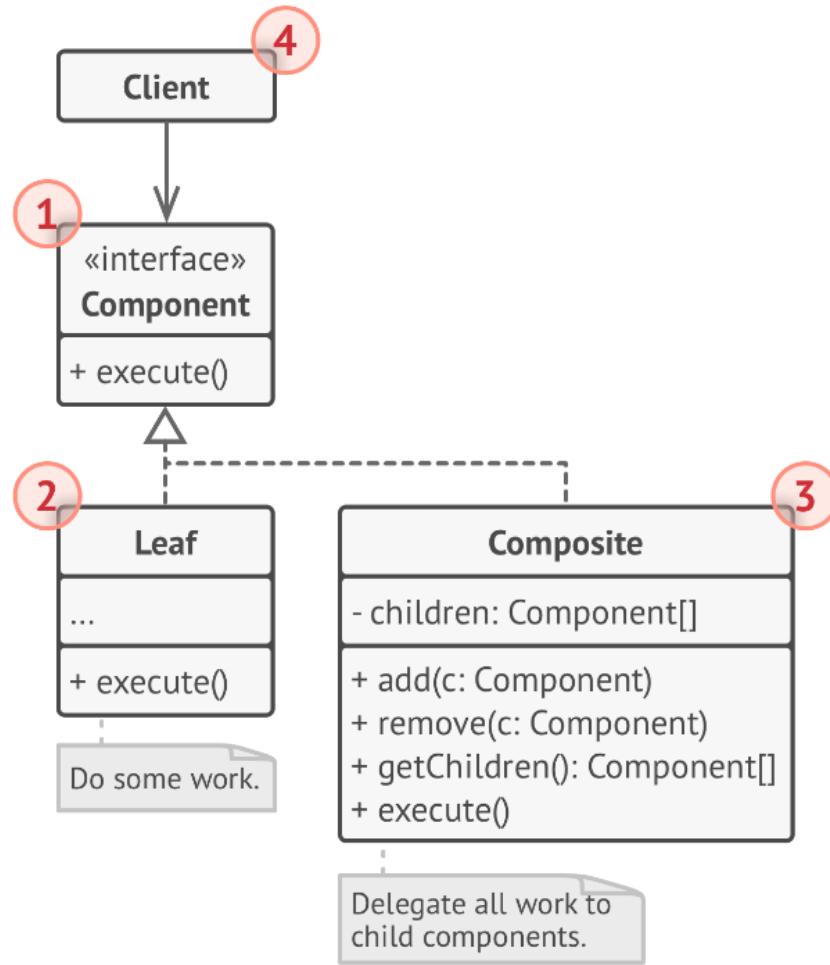
COMPOSITE PATERN

Composite patern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija.

Korištenje Composite paterna ima smisla jedino da aplikacija može biti predstavljena kao stablo. On pruža dva osnovna tipa elemenata koja dijele zajednički interfejs: listovi i složeni kontejneri. Kontejner se može sastojati i od listova od drugih kontejnera. To omogućuje da se konstruiše ugniježdene rekurzivne strukture objekata koja podsjećaju na stablo.

U našem projektu je moguće primijeniti ovaj patern na klasu Cart. Ta klasa je kontejnerske koja posjeduje Products, a iz te klase su izvedene klase Shoes, Clothing i Accessories pa je moguće u interface izdvojiti metodu za računanje cijene ili uklanjanje iz korpe (jer je to isto bez obzira na vrstu Product-a). Ako gledamo donji grafik Composite klasa bi bila naša klasa Cart . Leaf su pojedinačni elementi što predstavlja Product klasu. Component interface bi sadržavao metode za računanje cijene ili uklanjanje iz korpe, a Client klasa bi mogla biti neka klasa za obračun finalne (ukupne) cijene korpe.

Structure



PROXY PATTERN

Proxy pattern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe.

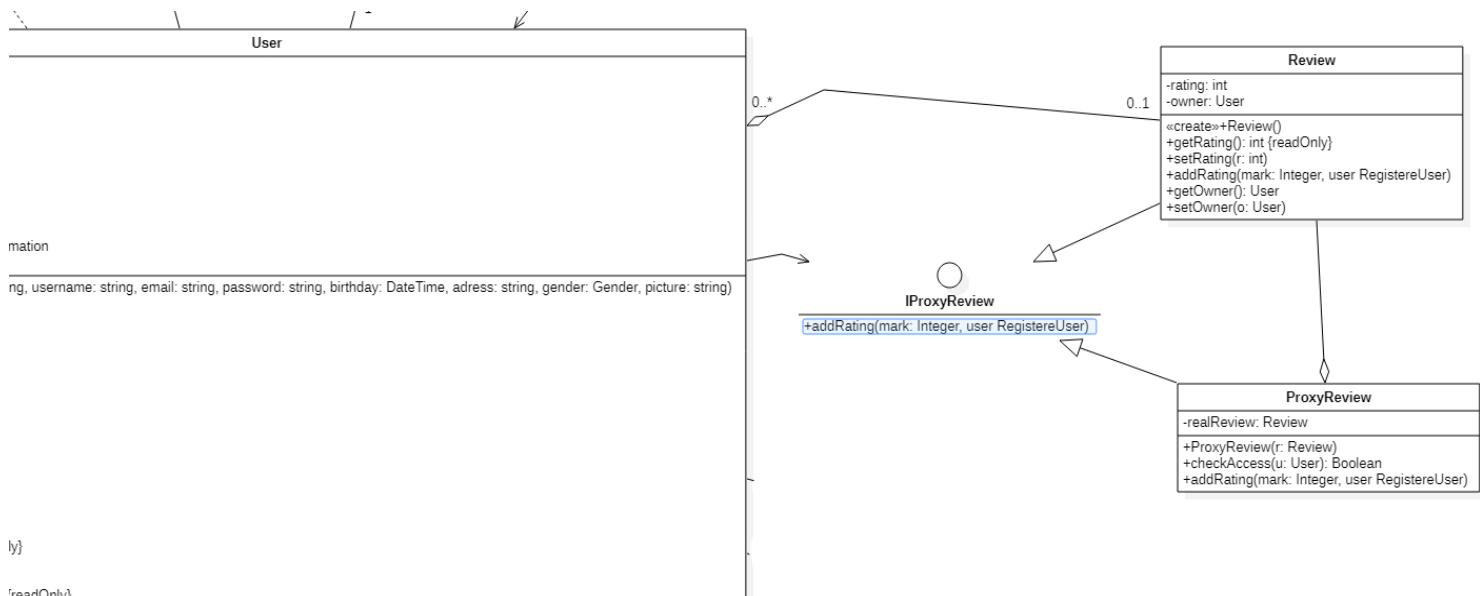
Primjenom ovog paterna omogućava se kontrola pristupa objektima, te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu.

Primjena u praksi:

Zaštita pristupa resursima

Ubrzavanje pristupa korištenjem cache objekata

Mi ćemo se u svojoj aplikaciji ograničiti na zaštitu pristupa resursima. U sistemu postoje klase User i Review. Recenzije želimo da ograničimo tako da je moguće ocijeniti samo User-a sa kojim je kupac poslova. Za tu akciju potrebno je dodati interface IProxyReview, koji sadrži metodu za ocjenjivanje, i klasu ProxyReview koja će provjeriti da li je moguće dodijeliti recenziju odabranom korisniku.

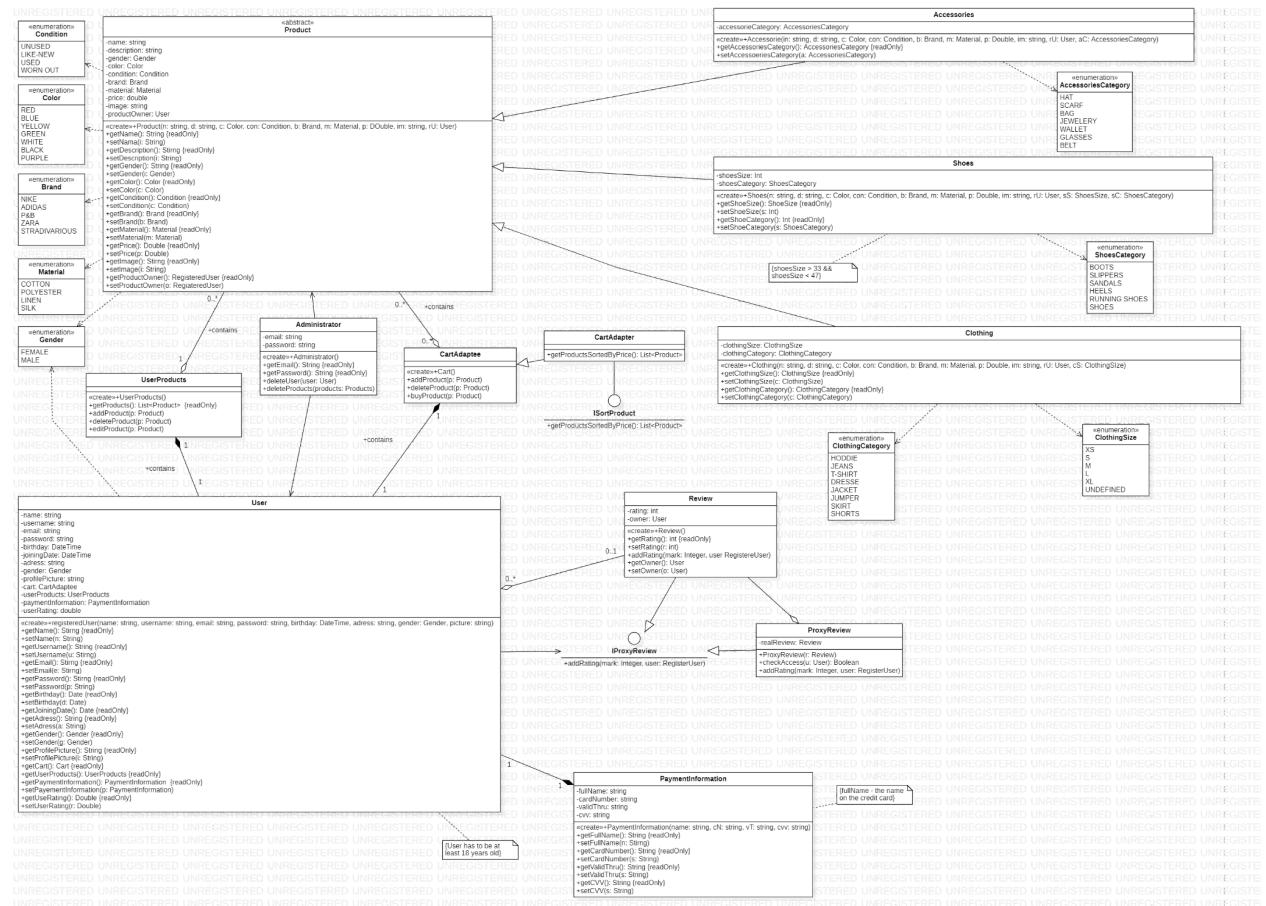


FLYWEIGHT PATERN

Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama (tzv. specifično stanje), vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instanca objekta (tzv. bezlično stanje).

Flyweight je dizajn patern koji omogućuje da "stavimo" više objekata u dostupnu količinu RAM memorije dijeljenjem zajedničkih dijelova između više objekata, umjesto da čuvate sve podatke u svakom objektu. Dakle, ideja je da se iskoristi ono što je zajedničko za više objekata i izdvoji gdje će biti dostupno za korištenje svim objektima koji su sačinjeni od tih elemenata (atribura). Pošto ovaj patern čini kod nečitljivim mi ga nećemo implementirati u svom projektu, al on bi se dobro morao iskoristiti npr za klasu gost gdje bi uvijek imali samo jednu instancu tog objekta, ili bi se moglo izdvojiti neki atributi iz Cloatnig, Accessories i Shoes klase koji bi bili svima zajednički (npr atribut boja se može ovako izdvojiti jer je zajednička za sve 3 klase).

DIJAGRAM KLASE S STRUKTURALNIM PATERNIMA



KREACIJSKI PATERNI

SINGLETON PATERN

Singleton patern osigurava da se klasa može instancirati samo jednom i da postoji globalni pristup kreiranoj instanci klase. Ovaj patern se koristi za objekate koje je potrebno samo jednom instancirati i nad kojim je potrebna jedinstvena kontrola pristupa.

Ovaj pattern se može primijeniti pri registraciji korisnika na sistem. Zamišljeno je da se pri tome izvrši jedna konekcija na bazu te da se svi upiti pišu u jednoj klasi te da se kroz nju dobavljaju podaci iz baze. Ovo je najbolje obaviti kroz Logger klasu. Ona će nam osigurati singleton konekciju na bazu i globalni pristup istoj, odnosno bilo koja klasa će moći pristupiti instanci te Logger klase i preko nje dobiti sve željene informacije.

FACTORY METHOD PATERN

Factory Method patern omogućava kreiranje objekata na način da podklase odluče koju klasu instancirati. Različite podklase mogu na različite načine implementirati interfejs. Factory Method instancira odgovarajuću podklasu(izvedenu klasu) preko posebne metode na osnovu informacije od strane klijenta ili na osnovu tekućeg stanja.

Factory Method patern možemo iskoristiti pri kreiranju instanca klase Product odnosno kada moramo odlučiti da li će kreirani Product biti Accessories, Shoes ili Clothing. Imat ćemo klasu ProductCreator koja posjeduje metodu productCreationMethod koja će kreirati odgovarajući tip Product u zavisnosti od parametra koji je proslijeđen koji predstavlja odabrani tip Producta koji korisnik želi kreirati.

U kodu bi navedeni pater izgledao ovako:

```
ProductCreator pc = new ProductCreator();
Product product = pc.productCreationMethod(productType : String);
```

BUILDER PATERN

Builder patern služi za apstrakciju procesa konstrukcije objekta, kako bi se kao rezultat moglo dobiti različite specifikacije objekta koristeći isti proces konstrukcije. Uloga Builder paterna je odvajanje specifikacije kompleksnih objekata od njihove stvarne konstrukcije. Isti konstrukcijski proces može kreirati različite reprezentacije.

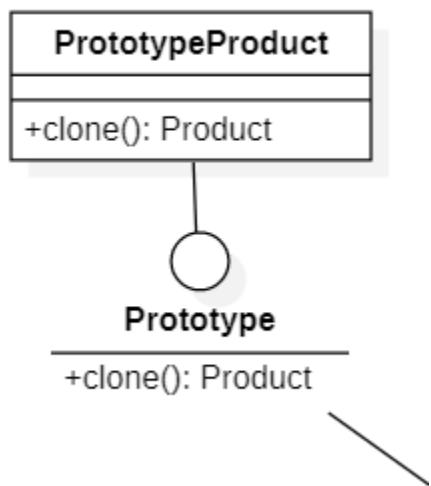
Pošto konstruktor klase User sadrži veliki broj parametara, builder patern možemo iskoristiti za kreiranje objekata tog tipa. Koristeći ovaj patern kreiranje objekta bi obavljali u klasi UserBuilder, pa bi i sve validacije podataka (email, korisničko ime, datum rođenja, informacije o plaćanju), koje bi se inače dešavale u konstruktoru klase User i odgovorajućim seterima za atribute, sada smjestili u tu klasu. Ovo olakšava i dodavanje novih atributa u klasu User.

PROTOTYPE PATERN

Prototype dizajn patern dozvoljava da se kreiraju prilagođeni objekti bez poznavanja njihove klase ili detalja kako je objekat kreiran. Ovaj patern prepušta proces kloniranja samom objektu koji se klonira preko zajedničkog interface-a koji implementiraju sve klase koje podržavaju kloniranje. Obično taj interface sadrži samo jednu metodu - clone(). Clone metoda vrlo je slična u svim klasama. Metoda stvara objekt trenutne klase i prenosi sve vrijednosti atributa starog objekta u novi. Objekt koji podržava kloniranje naziva se prototip (prototype). Kada zatreba objekat koji je već konfigurisan umjesto da se pravi novi objekat od nule samo se klonira prototip.

Prototype patern je korisno ugraditi u projekte s obzirom da je često postrebno više istih objekata. U našem projektu bi kloniranje podrzavala klasa Products kao i klase koje je

nasljeđuju. Klasa User nema smisla da implementira ovaj interface jer nije dozvoljeno da postoje dva identična korisnika (user-a).

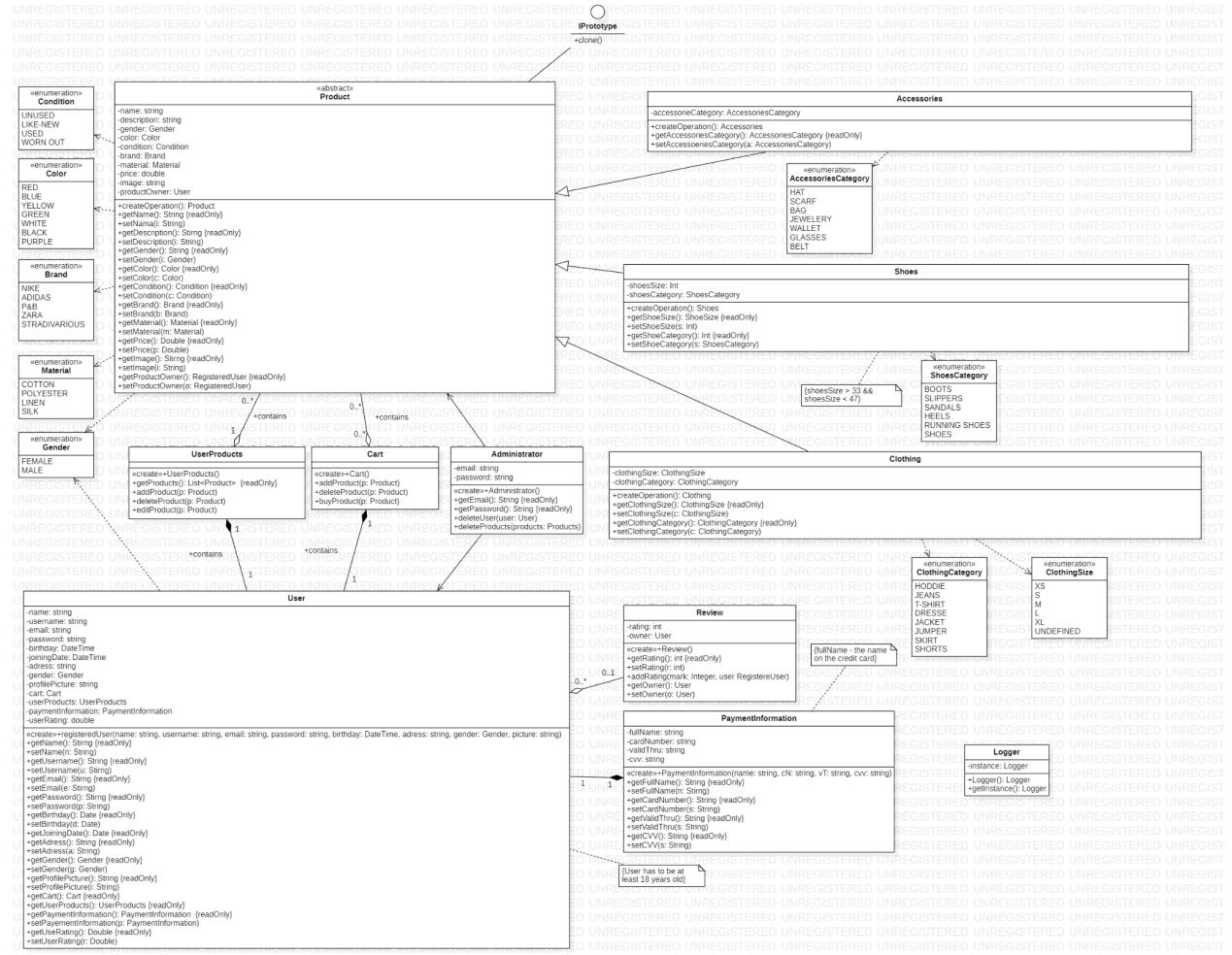


ABSTRACT FACTORY PATTERN

Abstract Factory patern nam omogućava da se kreiraju familije povezanih objekata odnosno produkata. Na osnovu apstraktne familije produkata kreiraju se konkretne fabrike produkata različitih tipova i različitih kombinacija a pošto patern odvaja definiciju klase produkata od klijenta, familije produkata je moguće jednostavno prikazivati, mijenjati i ažurirati.

Abstract Factory patern možemo iskoristiti kada korisnik bude vršio filtriranje proizvoda tokom pretrage. Na osnovu korisničkih filtera kreiraće se i fabrika produkata različitih tipova artikala kao i različitih kombinacija. Ovim bi se prikaz artikala učinio dosta efikasnijim s obzirom da Abstract Factory patern upravlja familijama produkata i čuva njihove detalje neovisnim od klijenata.

DIJAGRAM KLASE S KREACIJSKIM PATERNIMA



PATERNI PONAŠANJA

STRATEGY PATTERN

Ovaj patern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti primjenjivi algoritmi (strategije) za neki problem. Strategy patern omogućava klijentu izbor jednog od algoritma iz familije algoritama za korištenje. Algoritmi su neovisni od klijenata koji ih koriste.

U našem sistemu postoji idealno mjesto za iskorištavanje ovog paterna. Naime kada želimo sortirati Product-e po zahtjevu korisnika (recimo da korisnik želi sortirati Pruduct-e po cijeni od najniže ka najvišoj) mi možemo ponuditi razne "strategije" sortiranja, odnosno možemo osigurati različite algoritme sortiranja (recimo da su to Quick sort, Bubble sort i Selection sort). Svaki od ovih algoritama pruža različito vrijeme izvršavanja no u konačnici svi će imati isti rezultat, odnosno sortiranu listu Product-a.

STATE PATTERN

State patern predstavlja samo dinamička verzija Strategy paterna. Objekat mijenja način ponašanja na osnovu trenutnog stanja. To se postiže promjenom podklase unutar hijerarhije klasa.

Ovaj pattern bi mogli hipotetski iskoristiti pri ocjenjivanju korisnika odnosno u klasi Review. Recimo da želimo imati informaciju da li smo već ocijenili tog korisnika i da se u zavisnosti od toga može ili ne može željeni korisnik ocijeniti (recimo da ukoliko se ne može ponovo ocijeniti možemo vidjeti ocjenu koju smo prethodno dali). Potrebno je kreirati interfejs IOcjena koji će implementirati dvije klase Ocjenjen i Neocjenjen. Da bi se dinamički odredilo koja će klasa biti pozvana potrebno je provjeriti u bazi da li postoji review za odabranog korisnika koji je trenutni korisnik ostavio.

TEMPLATE METHOD PATTERN

Omogućava izdvajanje određenih koraka algoritma u odvojene podklase. Struktura algoritma se ne mijenja odnosno samo mali dijelovi operacija se izdvajaju i ti se dijelovi mogu implementirati različito.

Ovaj pattern je moguće i poželjno iskoristiti u našem sistemu. Naime potrebno je omogućiti korisniku da vrši filtriranje i sortiranje proizvoda na osnovu nekog kriterija. Da bi izbjegli duplicitiranje koda i komplikacije koristit ćemo ovaj pattern jer pristupamo istim Products iz baze te je njih potrebno sortirati i/ili filtrirati. Da bi se sve zamišljeno omogućilo kreirati ćemo klasu Sort koja sadrži listu Product-a te implementira metode sort() te condition() koja će biti apstraktna i njena implementacija ce se vršiti u izvedenim klasama. Recimo da imamo izvedene klase PriceSort, ConditionSort, GenderSort. Svaka od ovih klasa sortira listu Product-a po logičkom kriteriju (PriceSort po cijeni, ConditionSort po condition, GenderSort po gender).

ITERATOR PATTERN

Ovaj patern se koristi za prolazjenje kroz elemente kolekcije bez izlaganja strukture te kolekcije. Odnosno pruža nam pristup elementima, bez obzira na to kako je kolekcija strukturirana, da bi se oni mogli koristiti u ostalim dijelovima koda.

Iterator pattern možemo iskoristiti na bilo kojem mjestu gdje se javlja lista Product-a (klasa UserProducts i Cart). Kako je Product apstraktna klasa na njenom mjestu se može nalaziti bilo koja klasa izvedena iz nje što nam nudi idelno mjesto za iterator pattern, jer mi želimo biti u mogućnosti prolaziti kroz listu bez obzira koja je konkretna instanca klase u listi. Da bi ovaj pattern bio moguć potrebno je kreirati interfejs ICollection i Iterator. Klasa ProductCollection ce implementirati interfejs ICollection dok ce klasa ProductIterator implementirati interfejs Iterator. ICollection ce posjedovati metodu createIterator() dok će Iterator posjedovati metode koje omogućuju prolazak kroz Collection, te neke dodatne kao što su hasNext() i/ili hasMore().

OBSERVER PATTERN

Uloga ovog patterna je da uspostavi relaciju između objekata tako kada jedan objekat promijeni stanje drugi zainteresirani objekti se obavještavaju.

Ovaj patern mi nećemo implementirati u našem sistemu ali zgodna upotreba ovog paterna jeste da kada se ocijeni korisnik dobije obavijest o toj ocjeni (moguće je obavijestiti o ocjeni i o korisniku koji je ocijenio). Pri izvedbi ovog paterna potrebno je kreirati interfejs IObserve. Potrebno je da organizujemo mehanizam koji će obavještavati User-a. Kako je ovo u potpunosti nepotrebno u sistemu, nema potrebe da se dalje razrađuje ideja ovog paterna.

CHAIN OF RESPONSIBILITY PATTERN

Ovaj pattern omogućava objektu da pošalje naredbu bez znanja koji će objekt primiti i rukovati njime. Zahtjev se šalje s jednog objekta na drugi čineći ih dijelovima lanca i svaki objekt u ovom lancu može obraditi naredbu, proslijediti je ili učiniti oboje.

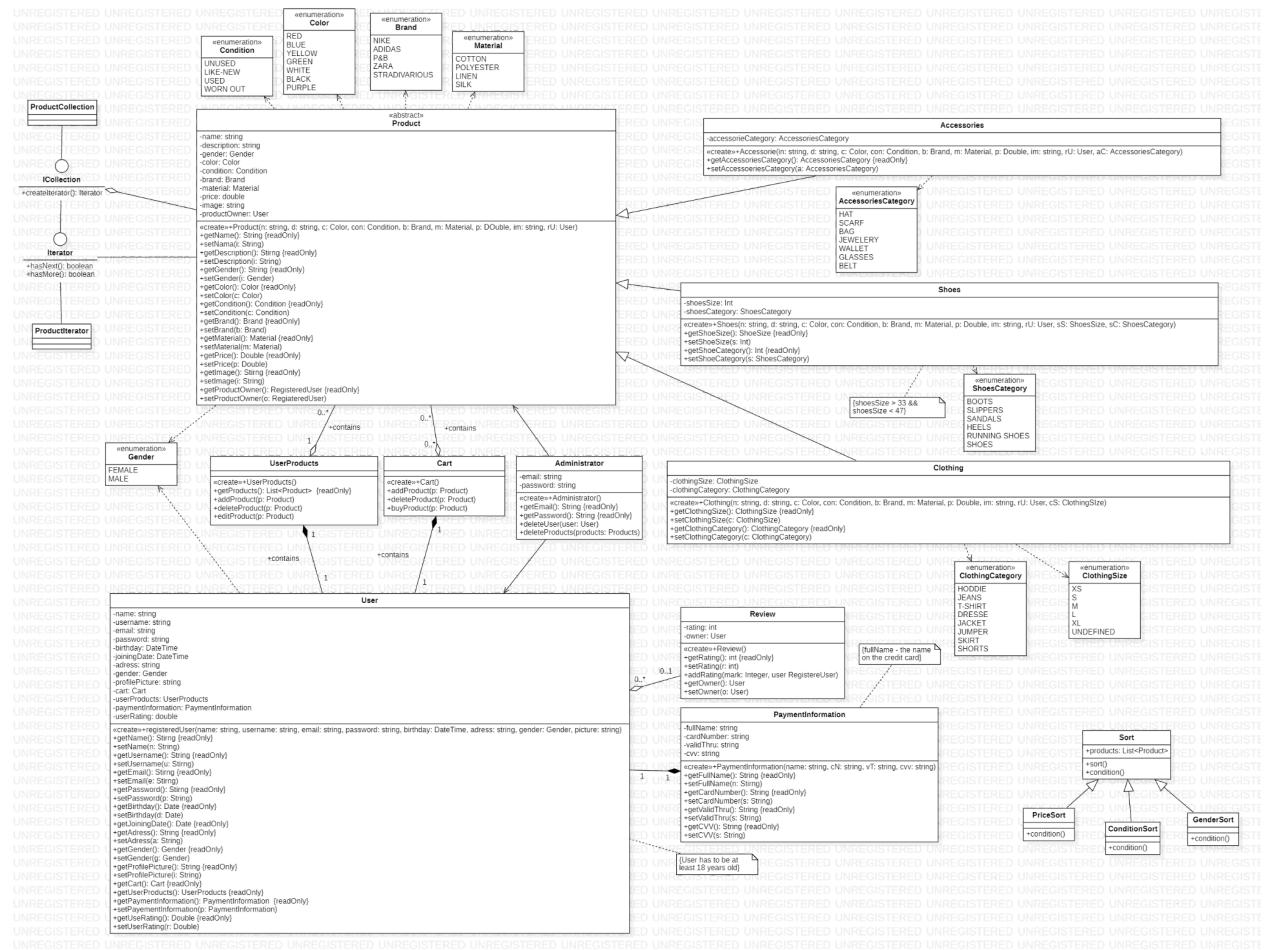
Ovaj patern je najlakše objasniti na primjeru GUI-a. Naime pri kliku na interfejs propagiraju se informacije handlerima da bi se izvršila željena akcija koja je inicirana klikom. Različiti Handleri izvršavaju različite akcije (tipa Edit, Remove i slično).

MEDIATOR PATTERN

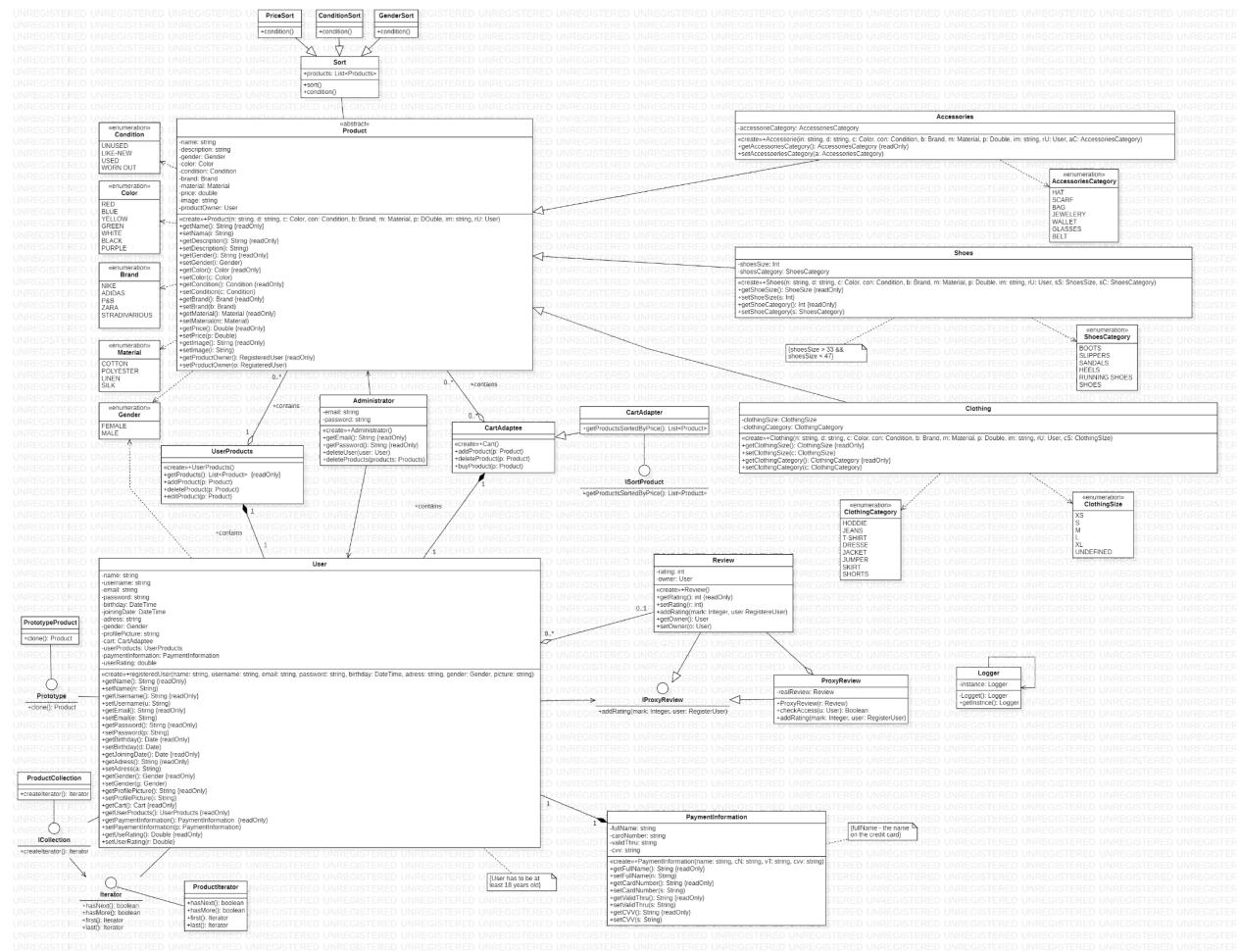
Mediator pattern koristimo da bismo izbjegli tijesno povezane frameworks, potreban nam je mehanizam koji će olakšati interakciju između objekata na način da objekti nisu svjesni postojanja drugih objekata.

Jedan od primjera predstavljaju Dialog klase u okvirima GUI aplikacija. Prozor dijaloga je kolekcija grafičkih kontrola. Klasa Dialog pruža mehanizam koji olakšava interakciju između kontrola. Na primjer, kada je nova vrijednost odabrana iz objekta ComboBox, oznaka mora prikazati novu vrijednost. I ComboBox i Label nisu svjesni međusobne strukture i svom interakcijom upravlja objekt Dialog. Svaka kontrola nije svjesna postojanja drugih kontrola.

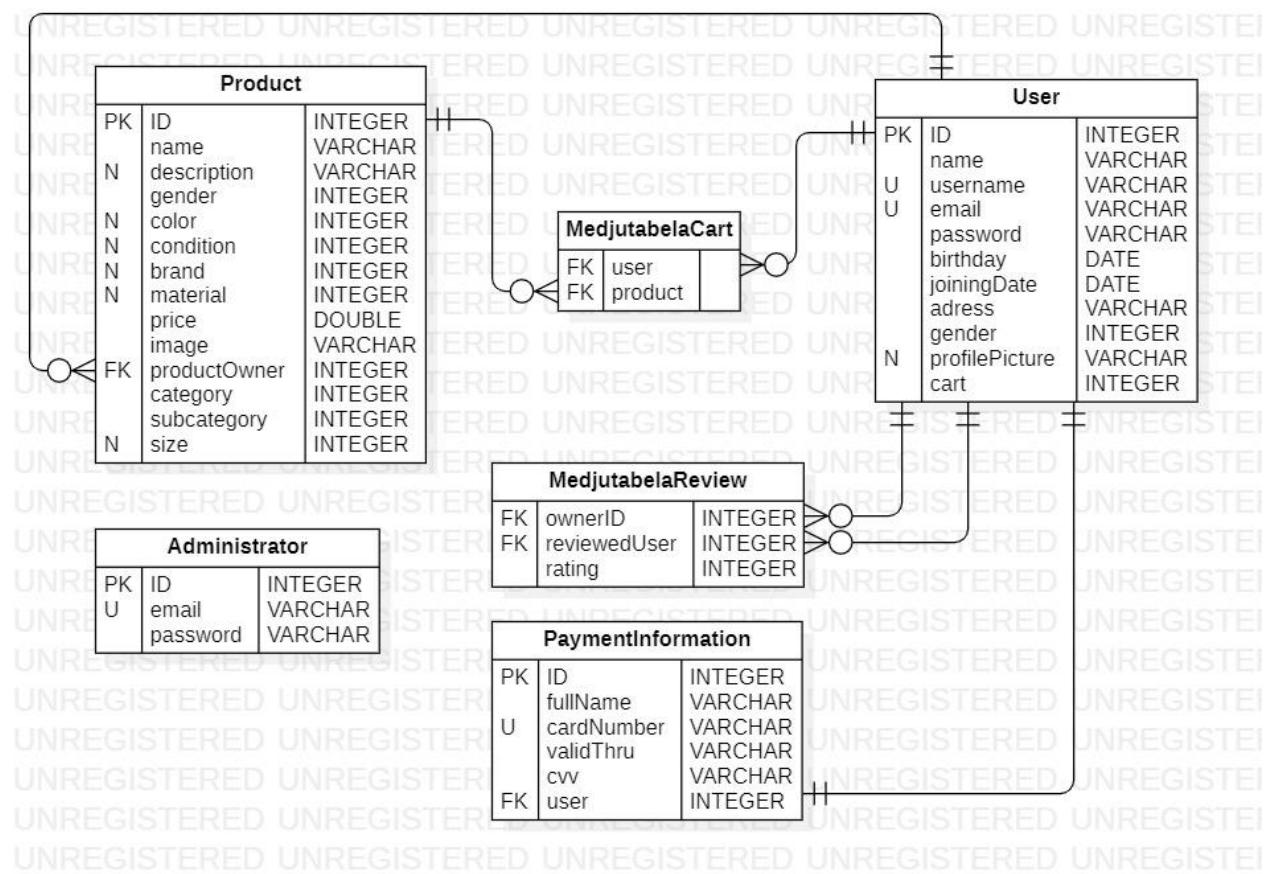
DIJAGRAM KLASE S PATERNIMA PONAŠANJA



DIJAGRAM KLASE S PATERNIMA

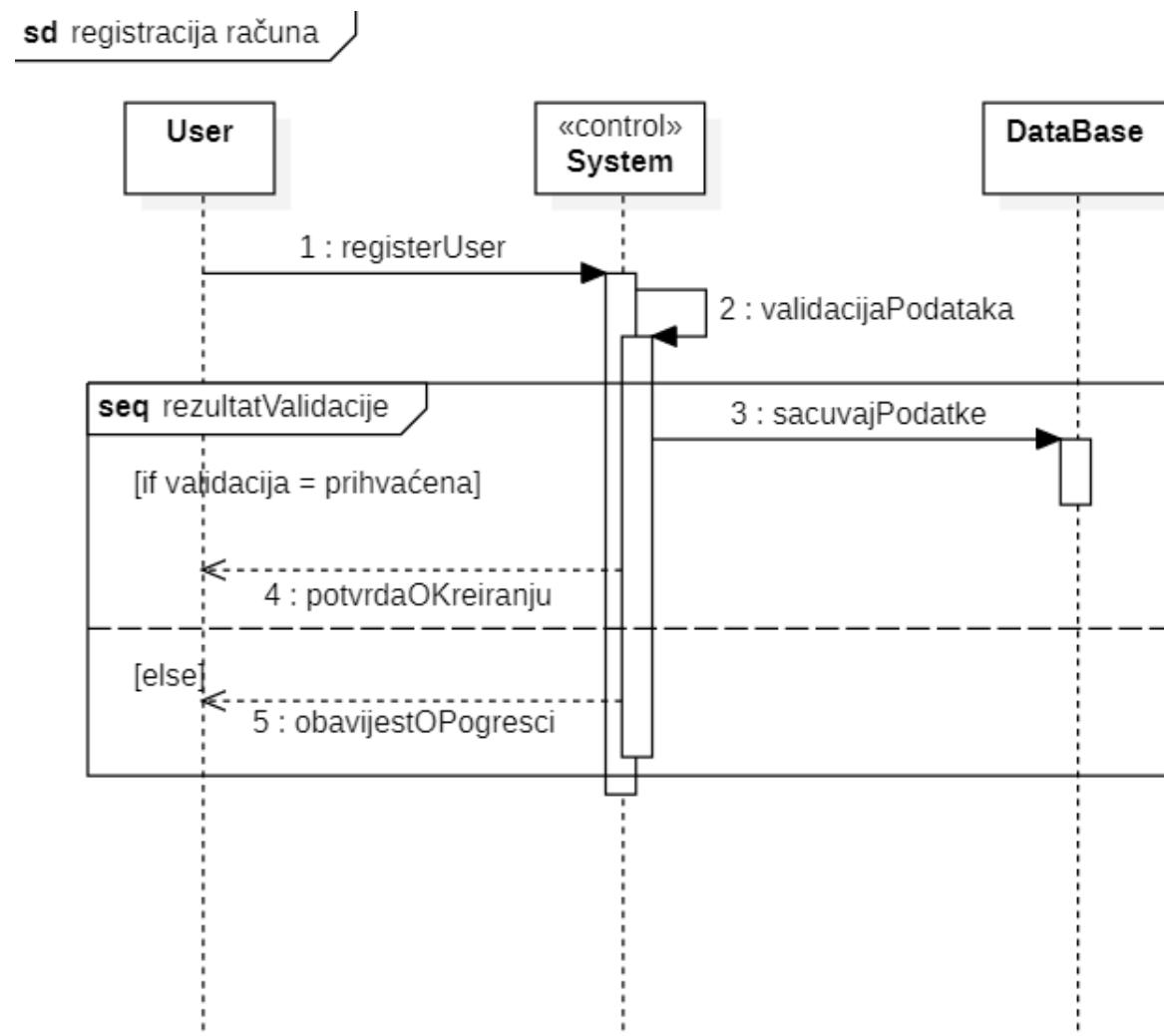


ENTITY RELATIONSHIP DIJAGRAM

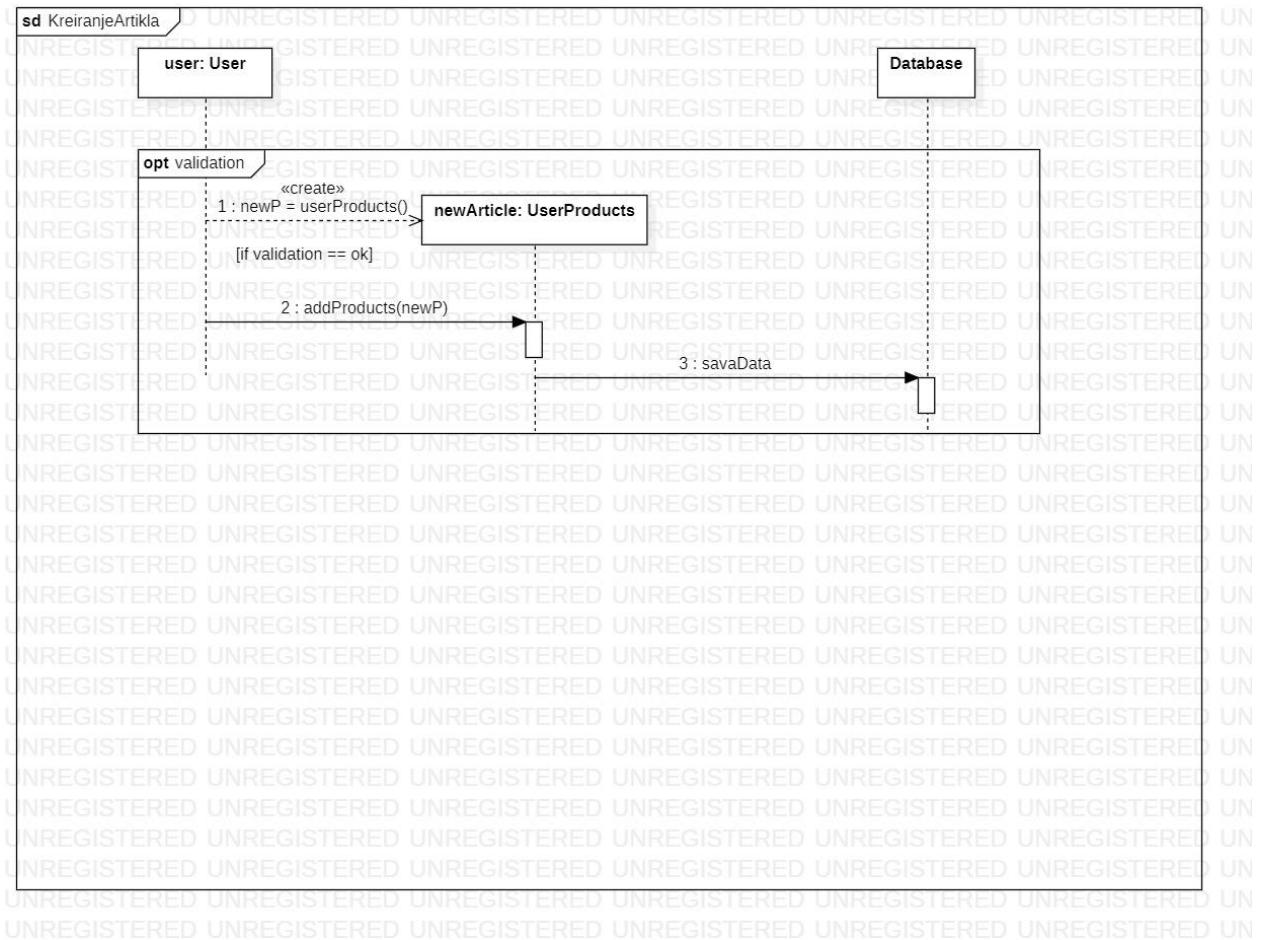


DIJAGRAM SEKVENCI

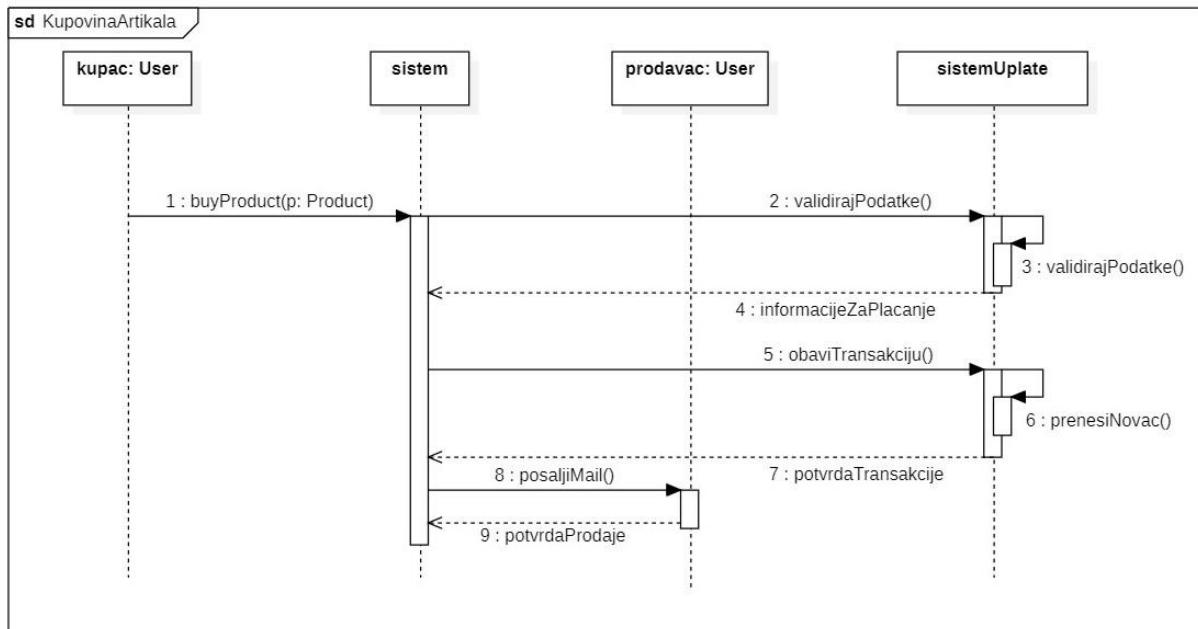
REGISTRACIJA RAČUNA



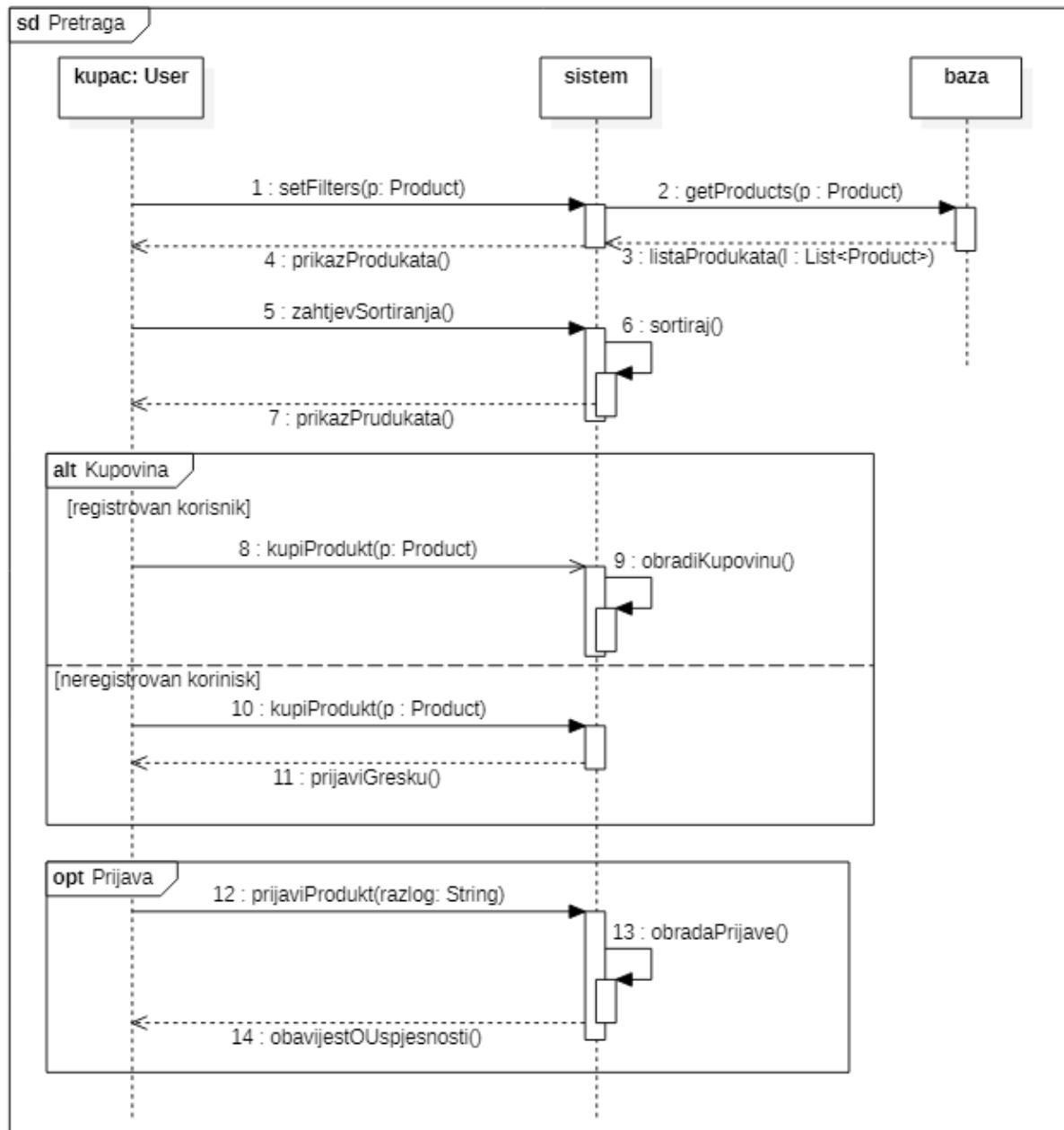
KREIRANJE NOVOG ARTIKLA



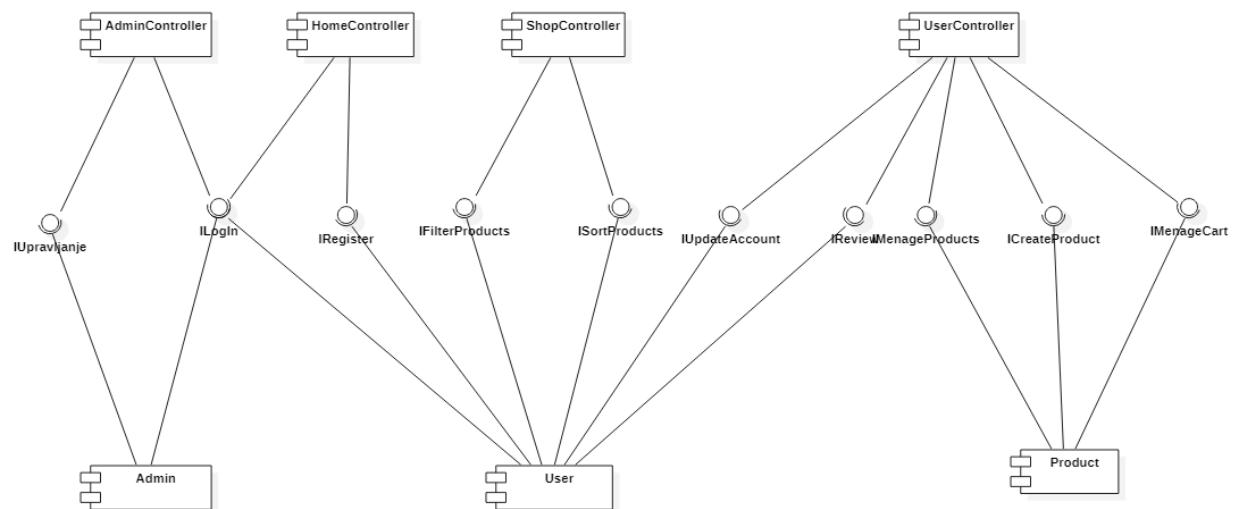
KUPOVINA



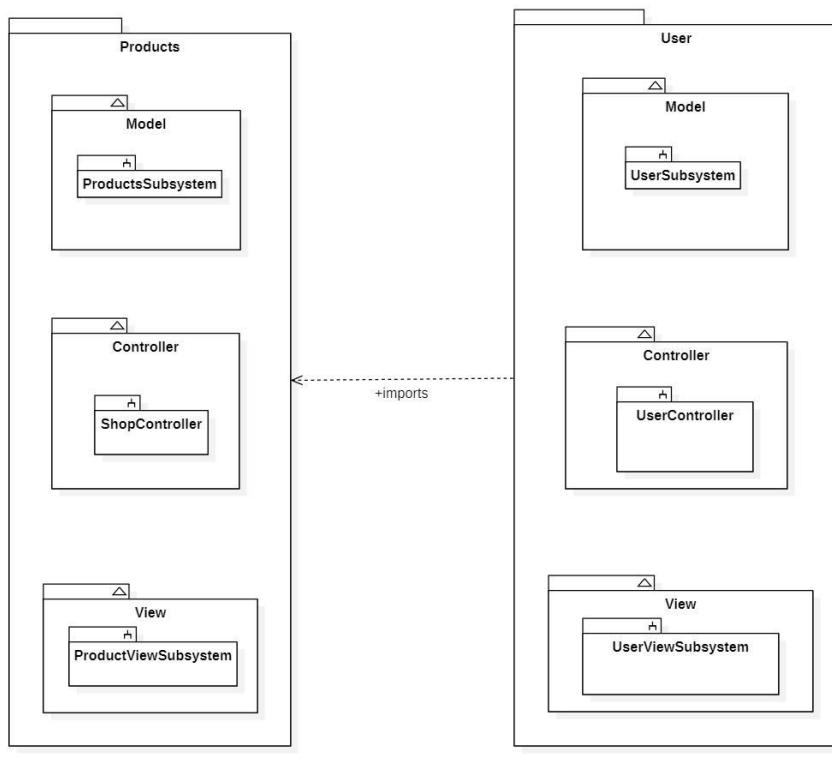
PRETRAGA



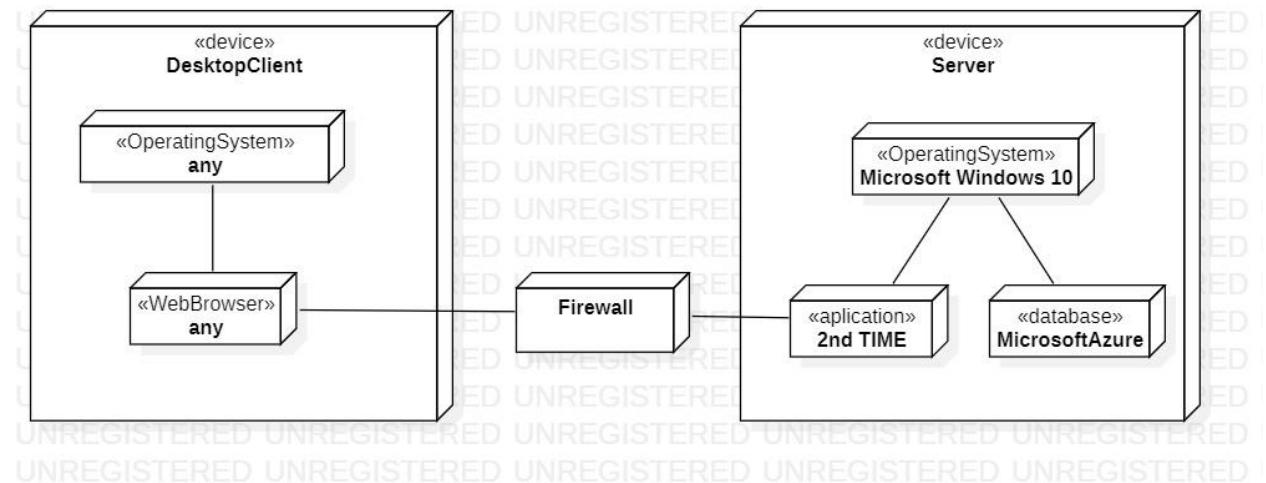
DIJAGRAM KOMPONENTI



DIJAGRAM PAKETA



DIJAGRAM RASPOREĐIVANJA



FINALNI KLASIČNI DIJAGRAM

