

Univerzitet u Sarajevu  
Elektrotehnički fakultet Sarajevo  
Računarstvo i informatika

## ITShop

### Dokumentacija modela

Predmet: Objektno orijentisana analiza i dizajn

Članovi grupe: Ahmed Pašić, Sabahudin Spahić, Elma Šeremet

Juni, 2020.

## Opis teme

ITShop je prvi korak prema "One-Stop-Shop" za sve vrste proizvoda i usluga koje pruža odjel za informatičke usluge. U svom trenutnom razvojnom studiju omogućuje korisnicima da lako kupuju hardver potreban za rad, studije, istraživanje i proučavanje. Korisnik će imati opciju da sam "sastavi" svoj idealni računar. Ako to ne želi, za odabrani iznos može dobiti računar sastavljen od strane naših uposlenika. Korisnik nam saopšti svoje potrebe, a mi pravimo računar idealan za navedenu svrhu. Kupovina je moguća samo ukoliko je korisnik prijavljen, a uz to, ukoliko je u pitanju student, dajemo pogodnosti prilikom kupovine hardvera.

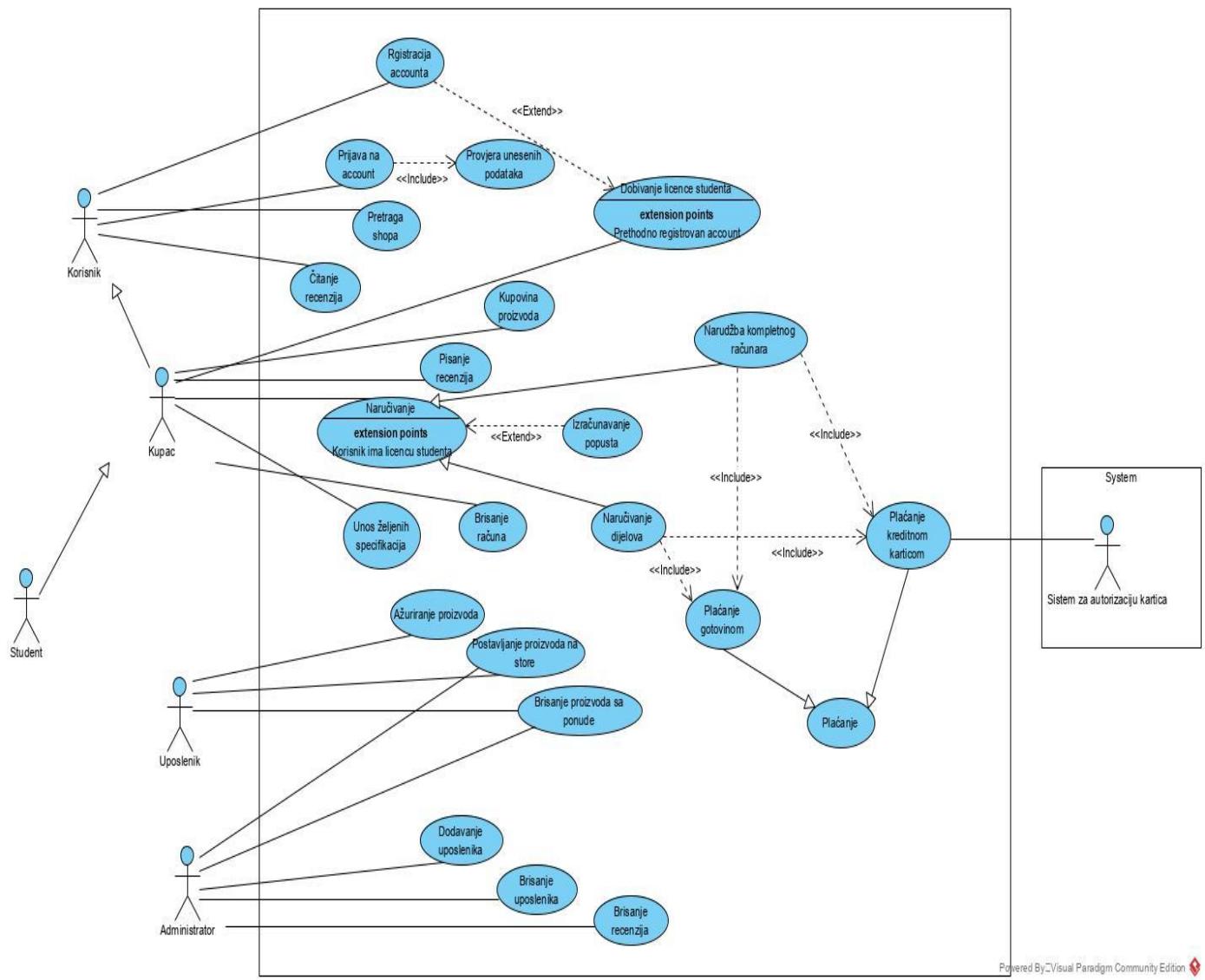
## Funkcionalnosti

- Kreiranje/brisanje korisničkog računa
- Prijava na račun
- Modifikacija vlastitog profila
- Pregled ponuđenih proizvoda po kategorijama
- Kupovina pojedinačnih proizvoda
- Sastavljanje "idealnog" računara
- Ostavljanje recenzija
- Upravljanje proizvodima (dodavanje/brisanje)
- Modifikacija proizvoda
- Jednostavno i brzo regulisanje plaćanja odabralih proizvoda

## Akteri

- Korisnik
- Kupac
- Student
- Uposlenik
- Administrator

## Dijagram slučajeva upotrebe (Use case diagram)



Powered By Visual Paradigm Community Edition

## Scenariji

U nastavku su prikazana 3 scenarija za naš program. Prvi scenarij se odnosi na prijavu

### Scenario 1

<b>Naziv</b>	<b>Login korisnika na svoj račun</b>
<b>Opis</b>	Registrovani korisnik unosi pristupne podatke i prijavljuje se svojim profilom
<b>Vezani zahtjevi</b>	
<b>Preduvjeti</b>	Korisnik ima pristup UI-u
<b>Posljedice-uspješan završetak</b>	Korisnik je prijavljen na svoj račun i ima sve privilegije koje nudi posjedovanje računa
<b>Posljedice-neuspješan završetak</b>	Sistem obavještava korisnika da nisu validni uneseni podaci, te moli korisnika za ponovni unos.
<b>Primarni akteri</b>	Korisnik
<b>Ostali akteri</b>	
<b>Glavni tok</b>	Korisnik se uspješno prijavljuje na svoj profil nakon unosa pristupnih podataka.
<b>Proširenja/Alternative</b>	

Tok događaja je prikazan sljedećom tabelom:

<b>Korisnik</b>	<b>Sistem ITShop</b>
1. Pristupanje interfejsu	2. Prikaz početnog prozora koji daje korisniku opciju prijave, registracije ili nastavka u ulozi gosta
3. Izbor prijave na račun	4. Otvaranje forme za unos pristupnih podataka za račun
5. Unošenje podataka	
6. Potvrda unesenih podataka	7. Provjera tačnosti unesenih podataka
	8. Prikaz odgovarajućeg prozora prijavljenom korisniku

Ponovo je u pitanju scenario prijave, pri čemu korisnik unese pogrešne podatke. Dakle, u pitanju je tok događaja kada imamo neuspješan završetak, odnosno, neispravnu prijavu:

<b>Korisnik</b>	<b>Sistem ITShop</b>
1. Pristupanje interfejsu	2. Prikaz početnog prozora koji daje korisniku opciju prijave, registracije ili nastavka u ulozi gosta
3. Izbor prijave na račun	4. Otvaranje forme za unos pristupnih podataka za račun
5. Unošenje podataka	
6. Potvrda unesenih podataka	7. Provjera tačnosti unesenih podataka
	8. Signaliziranje korisniku da su uneseni podaci neispravni
9. Korisnik ponovo unosi podatke ili izlazi iz prozora za prijavu	

---

Drugi scenarij koji smo obradili se odnosi na pisanje recenzija za neki proizvod:

### Scenarij 2

Naziv	Ostavljanje recenzija na odgovarajući proizvod
Opis	Registrani korisnik ima mogućnost ostavljanja recenzija na odgovarajući proizvod, te uvid u već ostavljene recenzije naših kupaca
Vezani zahtjevi	
Preduvjeti	Opcija "Guest" ima samo uvid u prethodno ostavljene recenzije. Za njihovo pisanje kupac mora biti registrovan!
Posljedice-uspješan završetak	Uspješno ostavljena recenzija koja može biti od pomoći i nekog novog saznanja za naše buduće korisnike.
Posljedice-neuspješan završetak	Upozorenje o nedozvoljenoj radnji!
Primarni akteri	Korisnik
Ostali akteri	
Glavni tok	Korisnik (ukoliko želi) ostavlja recenziju na odgovarajući proizvod. Navedena mogućnost se nalazi ispod svakog proizvoda.
Proširenja/Alternative	Ukoliko recenzija sadrži neprikladan sadržaj u najkraćem roku bit će obrisana.

Tok događaja sa ispunjenim preduvjetom:

Korisnik	Sistem ITShop
1. Ulazak u određeni proizvod	2. Izbacivanje osnovih informacija izabranog proizvoda
3. Pronalazak odjeljka namijenjenog za recenzije	
4. Pisanje recenzije	5. Provjera da li rezencija sadrži neki neprikladan sadržaj
	6. Uklanjanje recenzije ukoliko je odgovor na prethodno pitanje potvrđan

Tok događaja ukoliko preduvjet nije ispunjen:

---

Korisnik	Sistem ITShop
1. Ulazak u određeni proizvod	2. Izbacivanje osnovih informacija izabranog proizvoda
3. Pronalazak odjeljka namijenjenog za recenzije	
4. Pisanje recenzije	5. Upozorenje o nedozvoljenoj radnji!
	6. Pružanje mogućnosti kupcu da se prijavi ukoliko posjeduje korisnički račun

Treći scenarij koji smo obradili se odnosi na provjeru da li je kupac ustvari prijavljen studentskim emailom:

Naziv	Provjera korisnika da li je student
Opis	Korisnik upisuje u TextBox svoj mail
Vezani zahtjevi	
Preduvjeti	Napravljen korisnički račun
Posljedice - uspješan završetak	Promjena statusa računa
Posljedice - neuspješan završetak	Informacija o neuspjeloj radnji
Primarni akteri	Korisnik
Ostali akteri	Student
Glavni tok	Korisnik ulazi u glavni prozor svog računa Tu mu se nude razne opcije, među kojima i verifikacija mail-a
Proširenja/Alternative	

Korisnik u ovom scenariju već ima kreiran račun. Ovaj scenario obrađuje slučaj kada korisnik svoj račun je kreirao u ranijem periodu, a u međuvremenu je dobio status studenta prilikom upisa na fakultet. Ova opcija omogućava korisnicima da, bez pravljenja novog računa, promijene svoj status sa korisnika u studenta. Tok događaja:

Korisnik	Sistem ITshop
1. Korisnik se upisuje na svoj račun	2. Scenario obrađen u drugom dokumentu, pogledati
3. Otvaranje interfejsa za korisnički račun	4. Prikazuje se glavni prozor za pregled stanja korisnika
5. Unos validnog mail-a	
6. Potvrda unesenog podatka	7. Salje se zahtjev u bazu podataka gdje se provjerava da li postoji mail
	8. Signalizacija korisniku da li je radnja bila uspješna ili ne
9. Povratak u glavni prozor	10. Ovisno o ishodu provjere stanje korisnika se mijenja u student ili ostaje nepromjenjeno

Guest osoba nema trenutačno račun unutar sistema, ali ima status student-a.

<b>Naziv</b>	Promjena statusa
<b>Opis</b>	Guest prijavljuje se preko svog student mail-a
<b>Vezani zahtjevi</b>	
<b>Preduvjeti</b>	Posjedovanje student mail-a
<b>Posljedice - uspješan završetak</b>	Napravljen student račun
<b>Posljedice - neuspješan završetak</b>	Informacija da mail nije validan
<b>Primarni akteri</b>	Guest
<b>Ostali akteri</b>	Student
<b>Glavni tok</b>	Guest prilikom pravljenja računa u aplikaciji, bira opciju student, gdje pravi račun koristeći svoj student mail
<b>Proširenja/Alternative</b>	

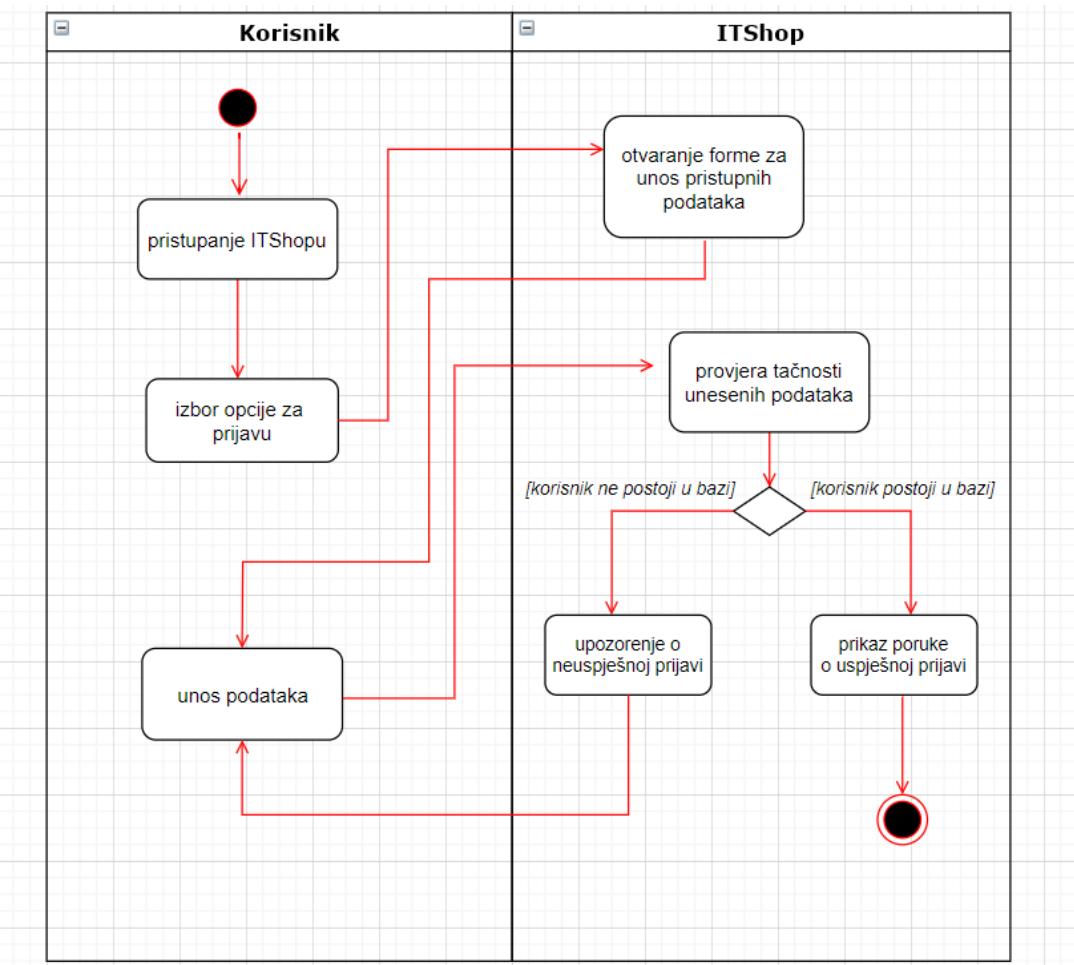
Tok dogadaja:

Guest	Sistem ITshop
1. Guest započinje postupak pravljenja računa	2. Unose se podaci
3. Guest uzima opciju student	4. Traži se unos student mail-a
5. Guest unosi mail	6. Vrši se provjera validnosti mail-a
	7. U zavisnosti od rezultata, TextBox za mail će biti popunjeno ili tražiti novi unos
8. Nastavlja se daljnje popunjavanje podataka o osobi	9. Kreira se novi račun sa statusom student

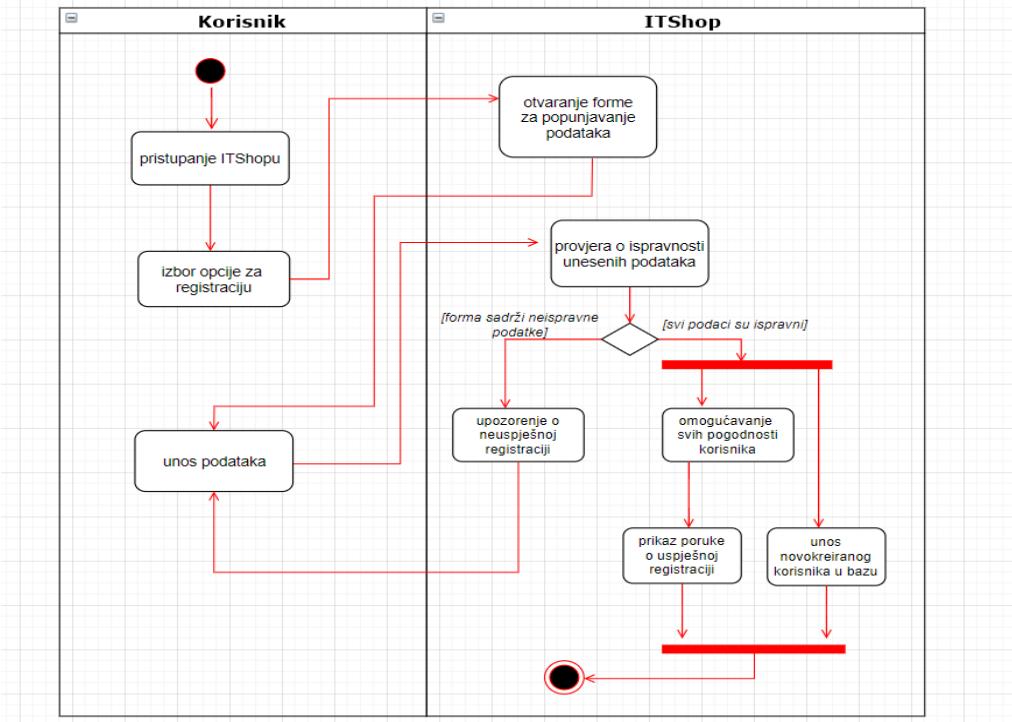
## Dijagrami aktivnosti (Activity diagrams)

Ispod je prikazano ukupno 16 različitih dijagrama aktivnosti.

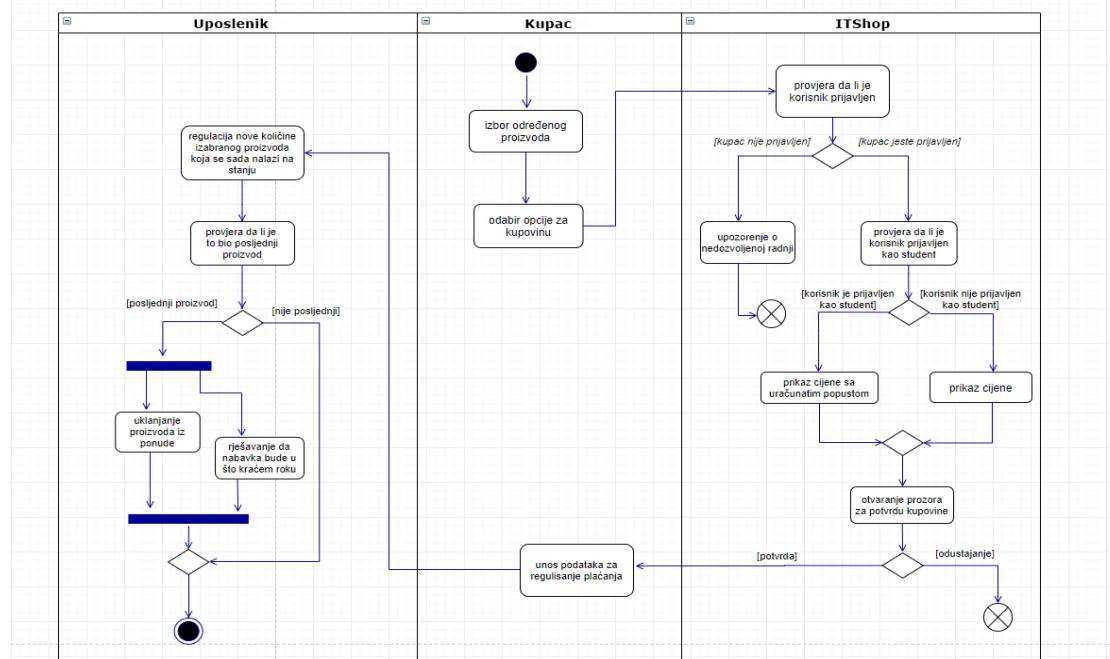
### 1) Prijava



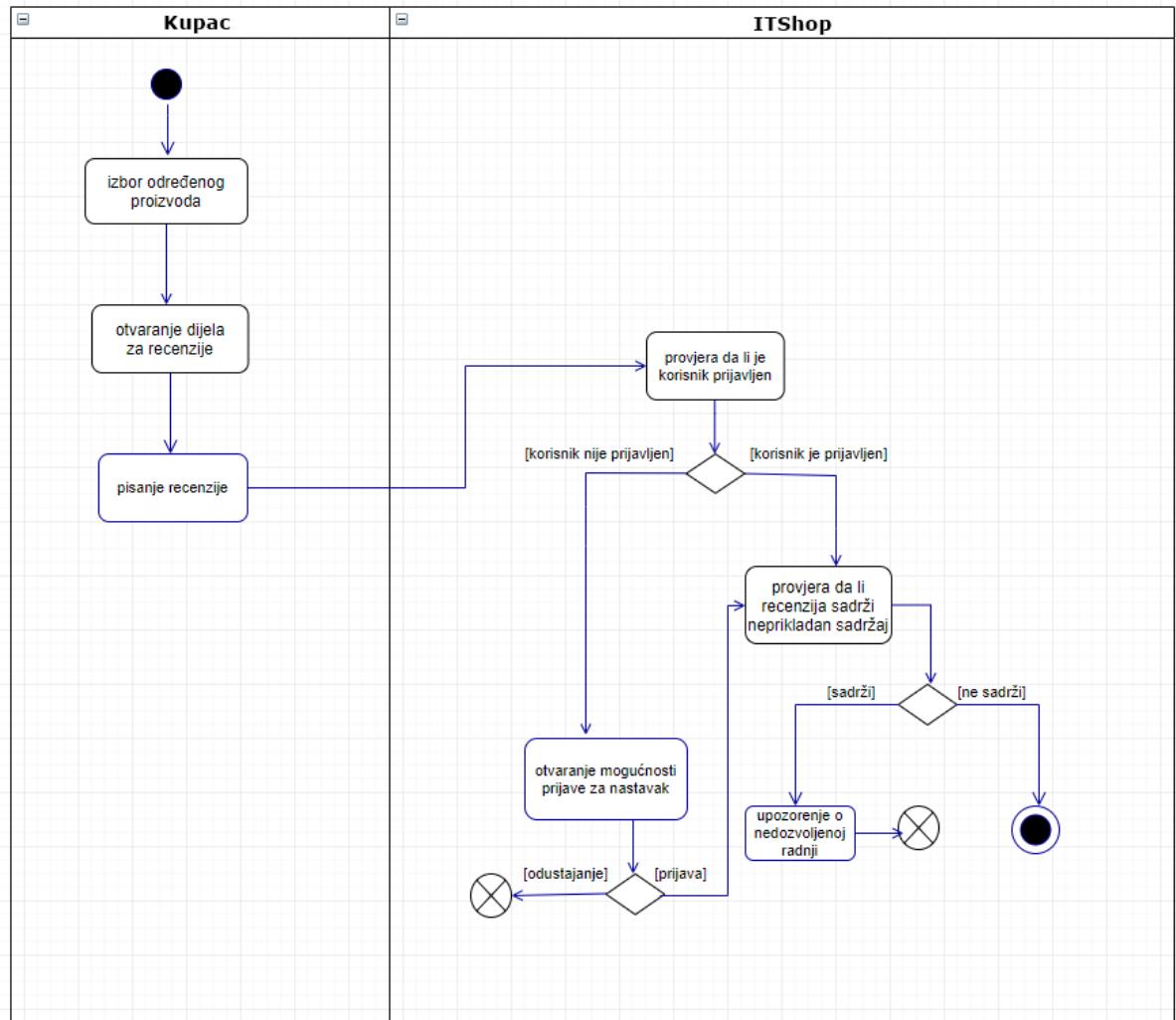
## 2) Registracija



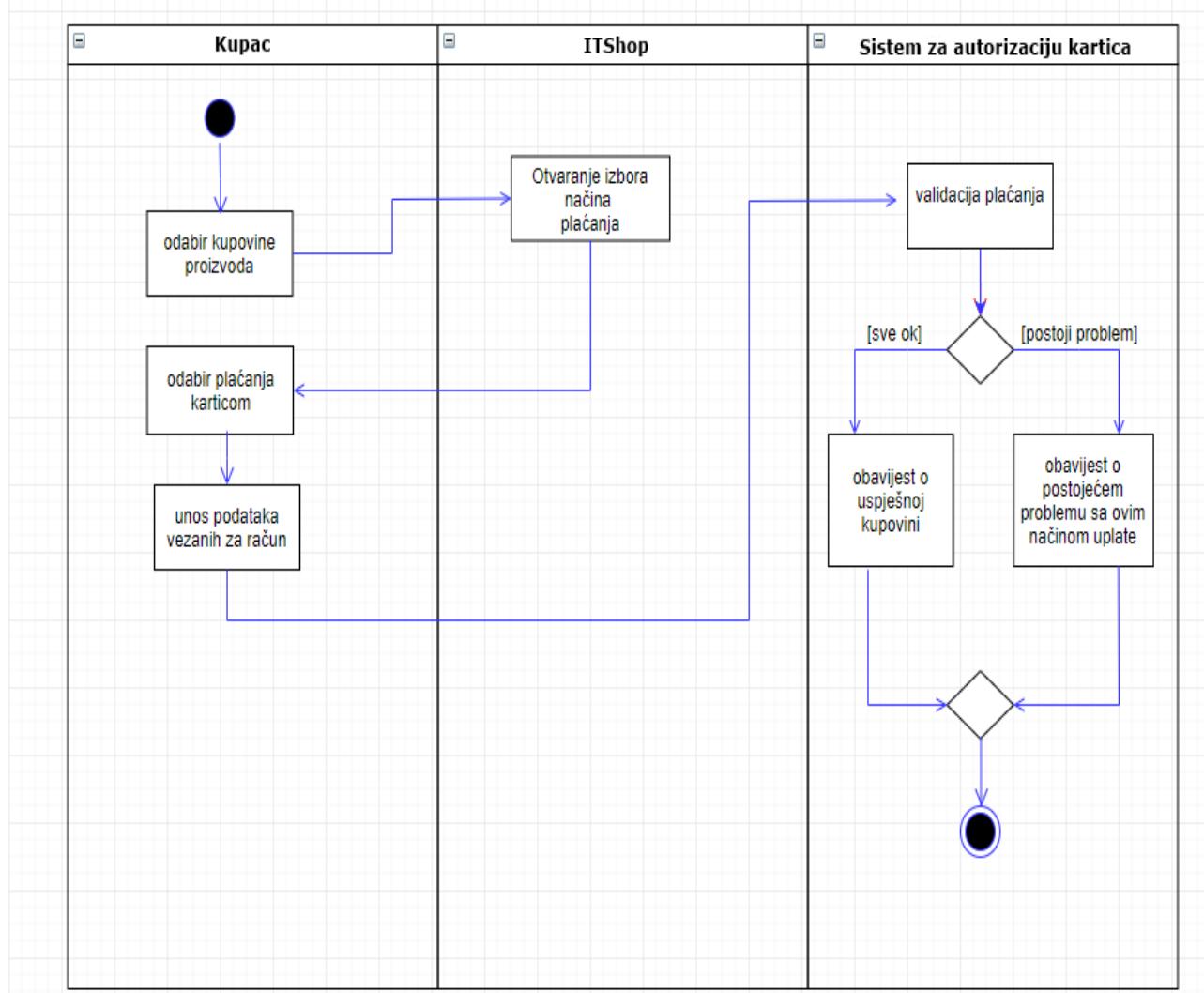
## 3) Kupovina



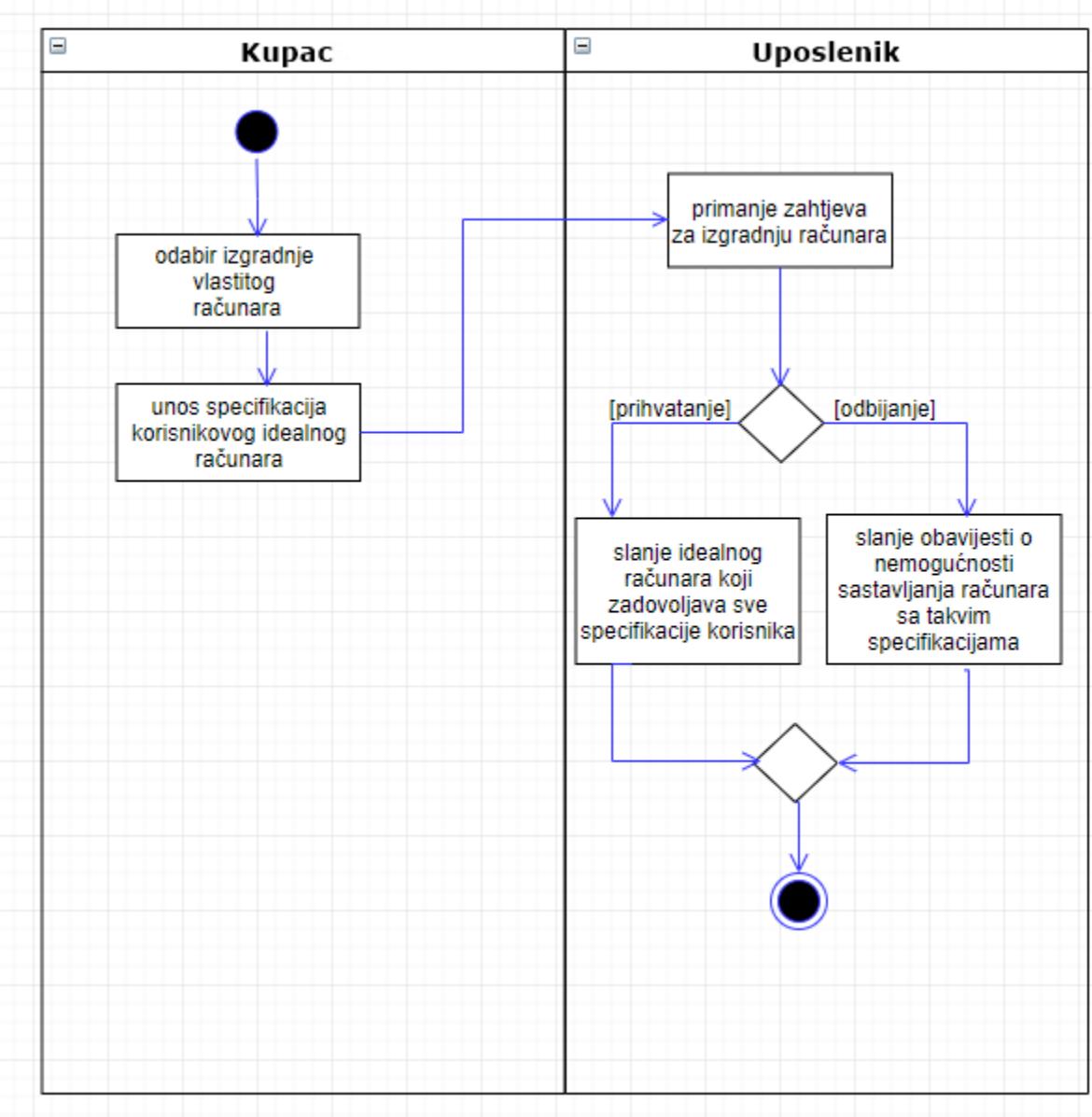
#### 4) Pisanje recenzija



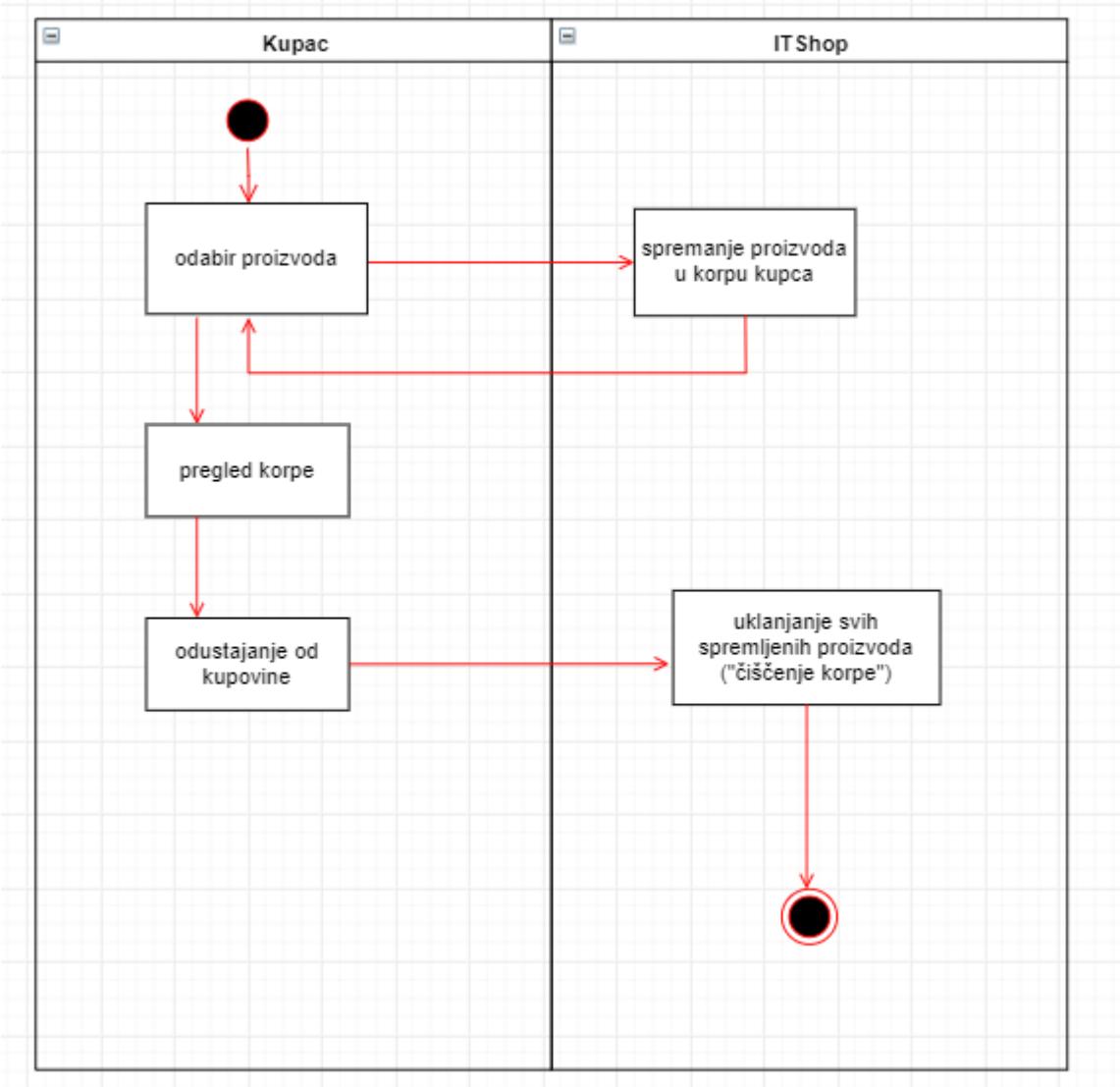
## 5) Plaćanje



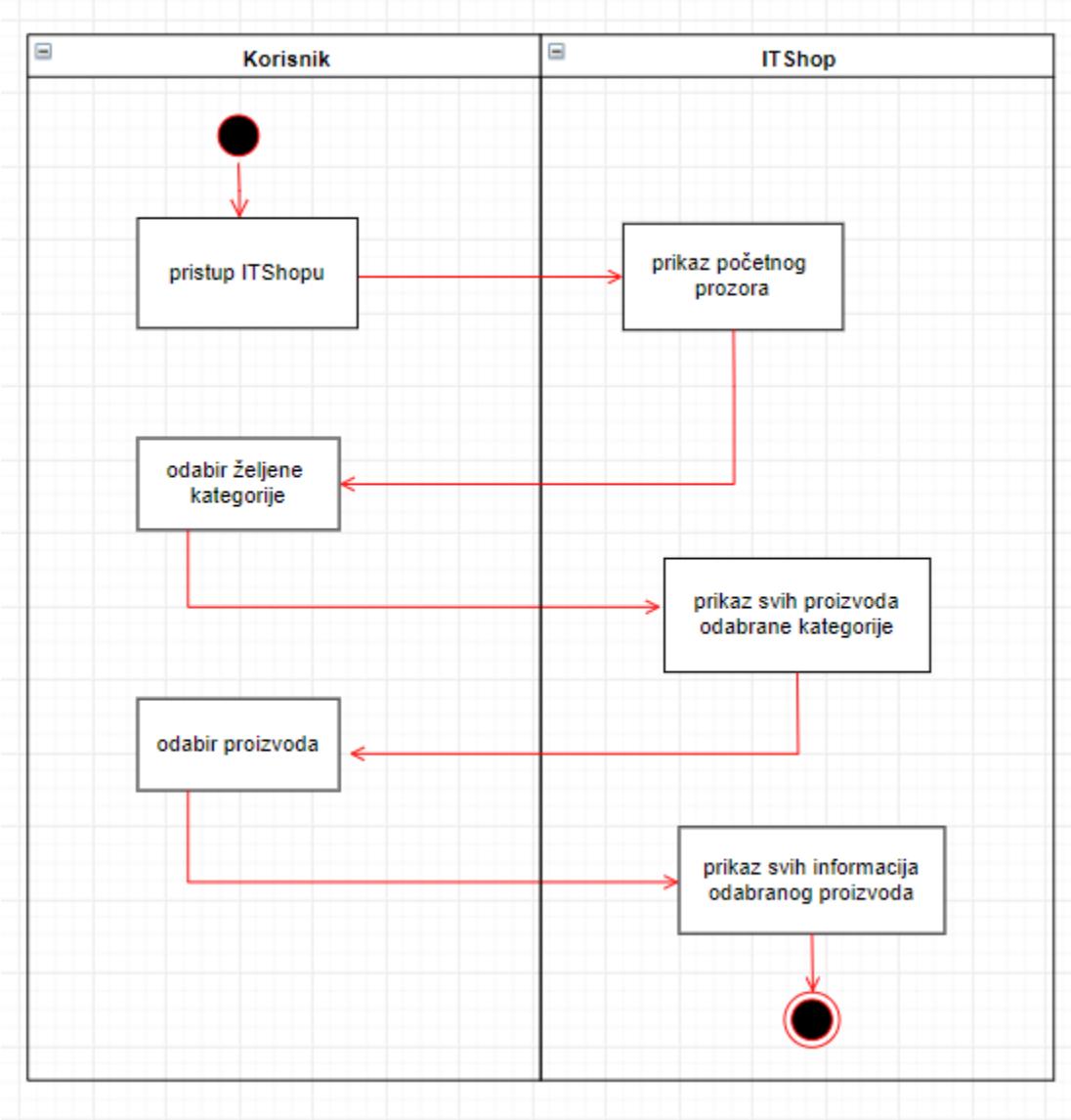
## 6) Prijedlog računara



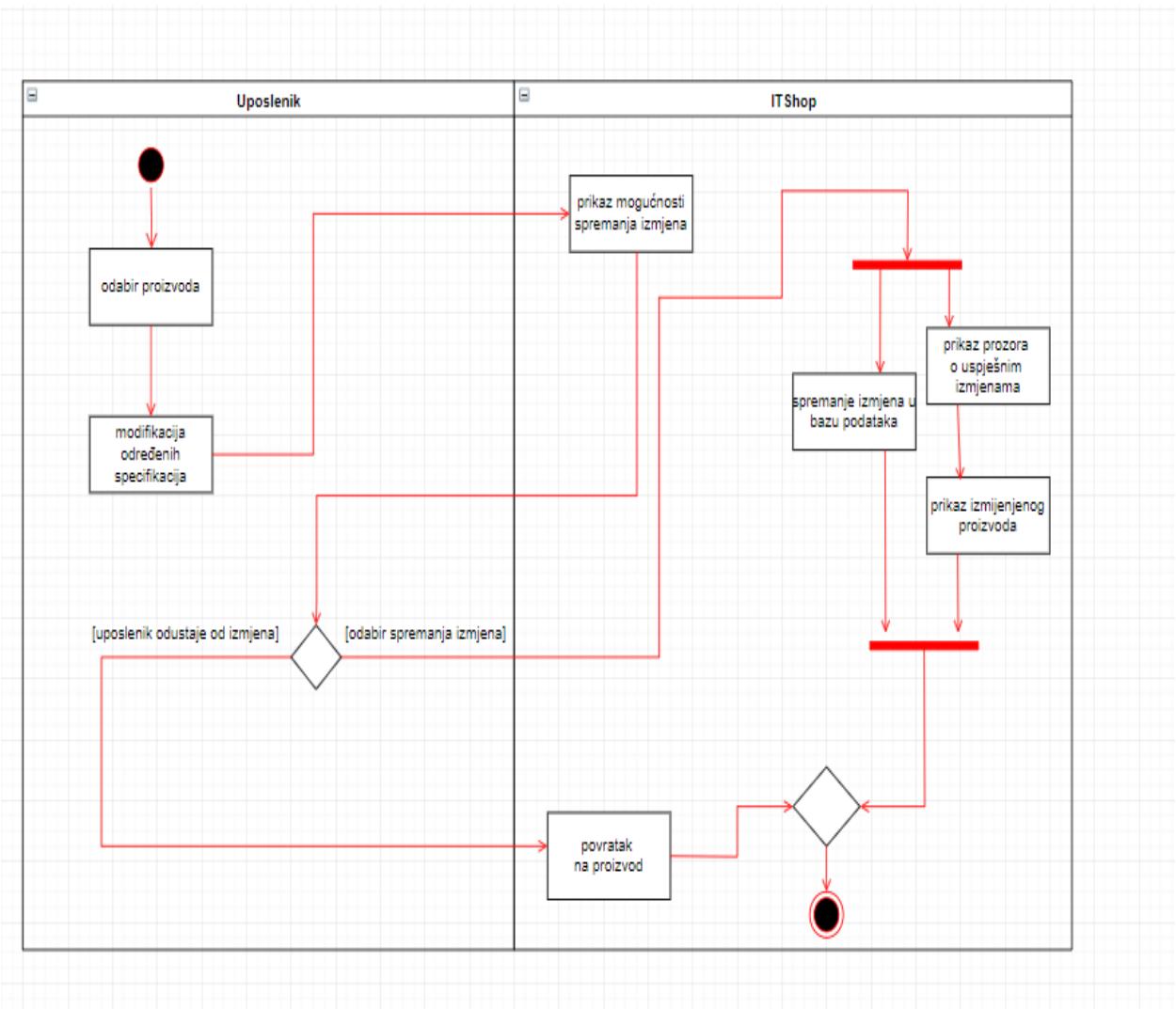
## 7) Odustajanje od kupovine



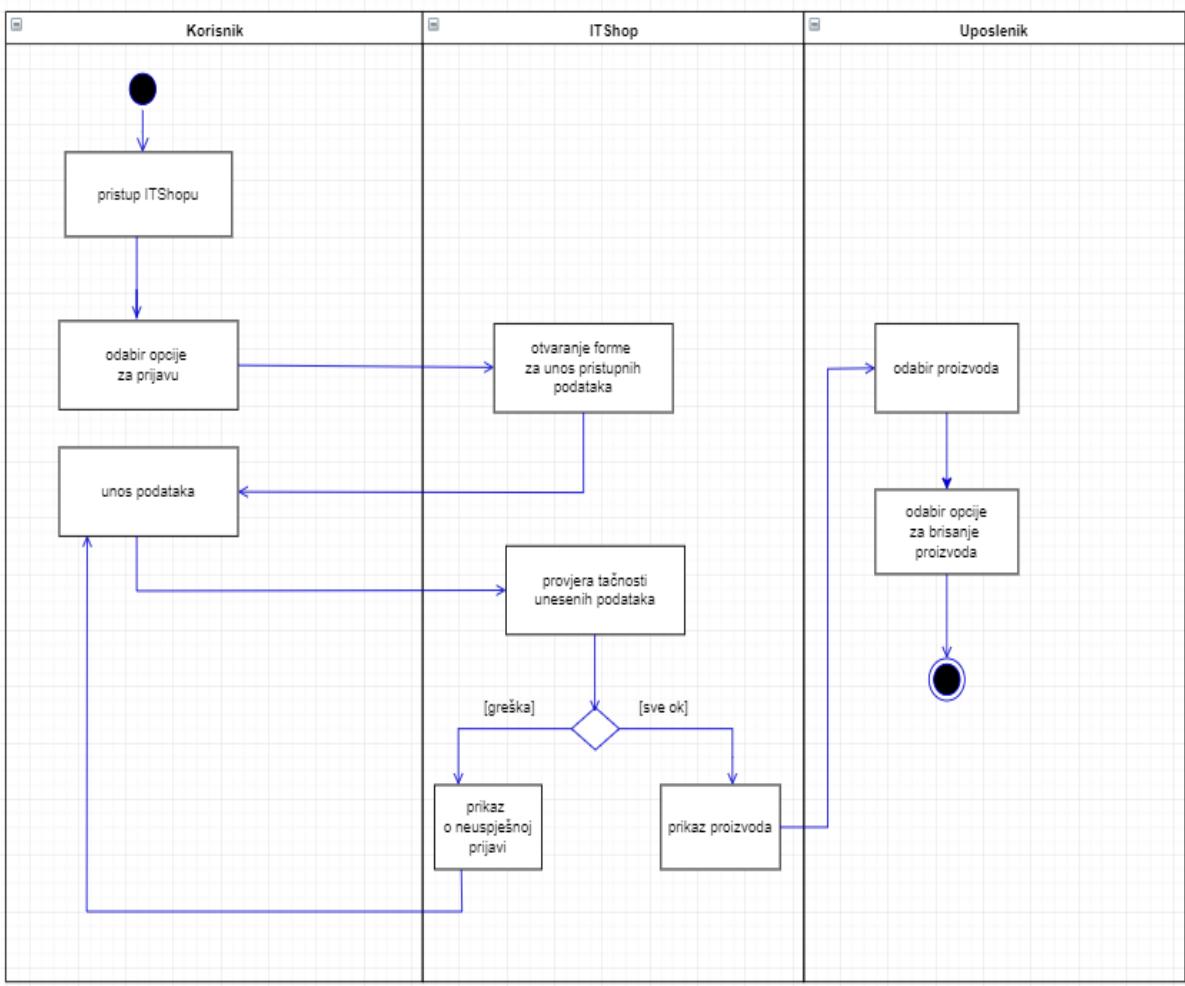
## 8) Prikaz kategorije proizvoda



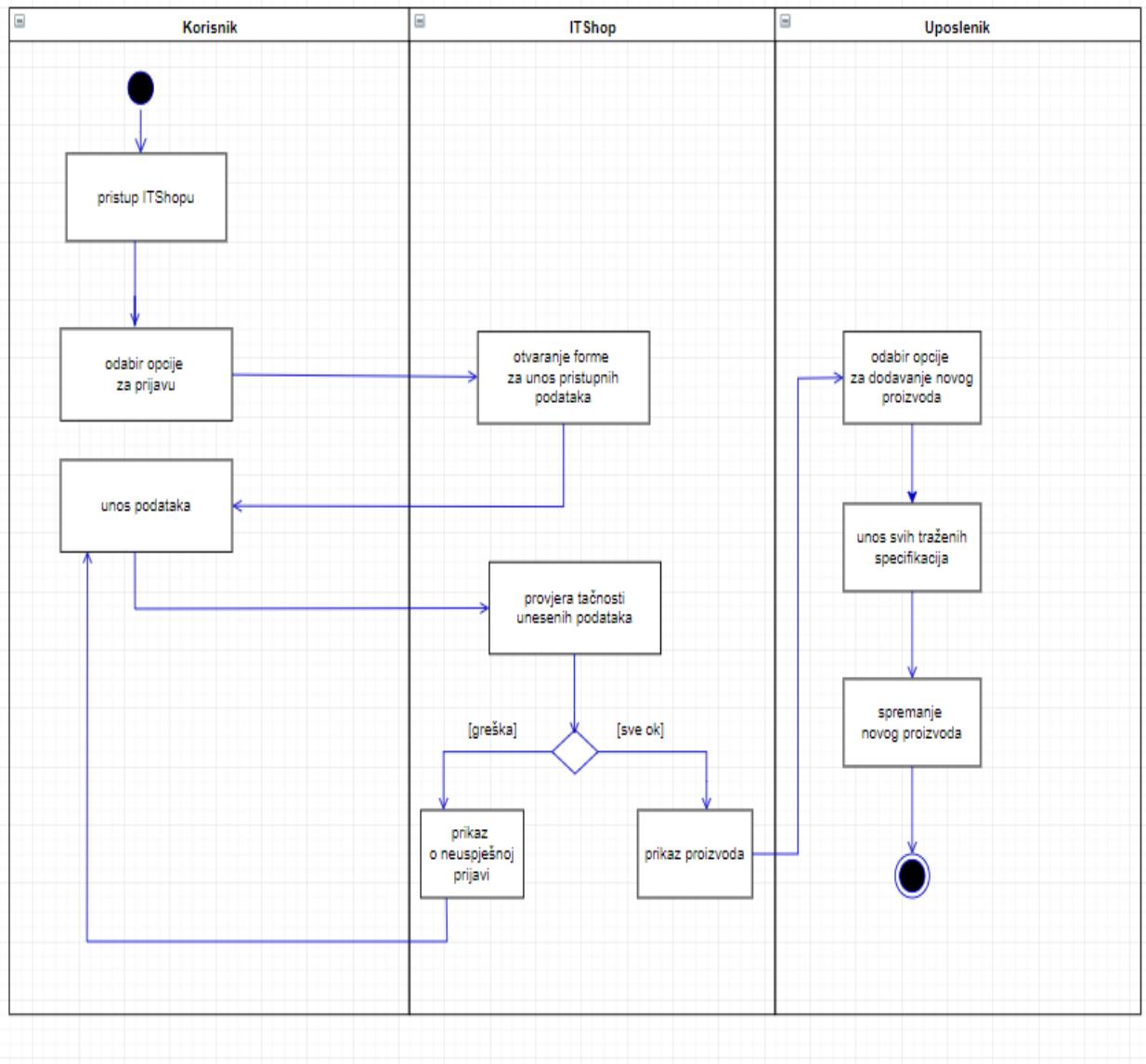
## 9) Izmjena proizvoda



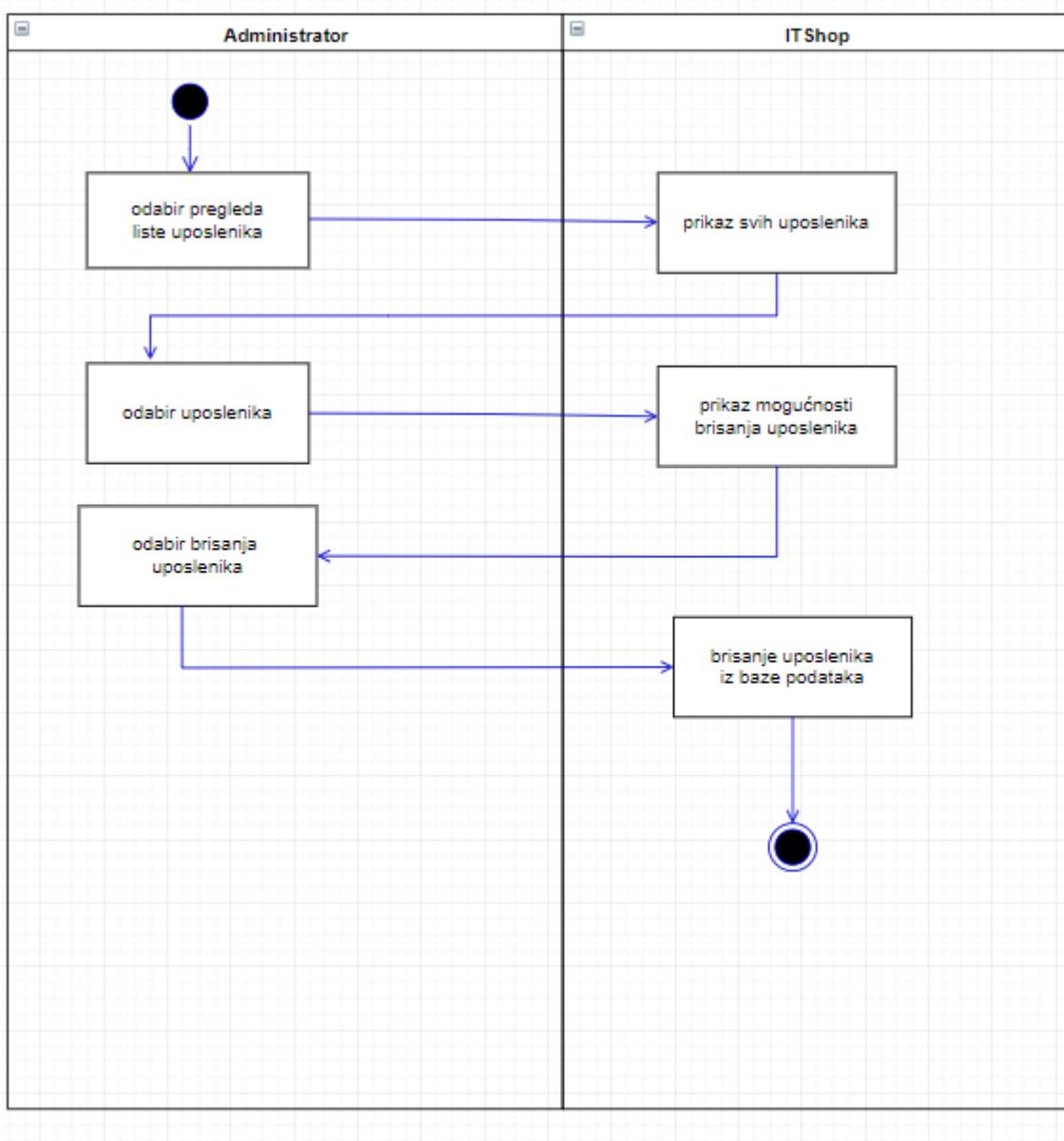
## 10) Brisanje proizvoda



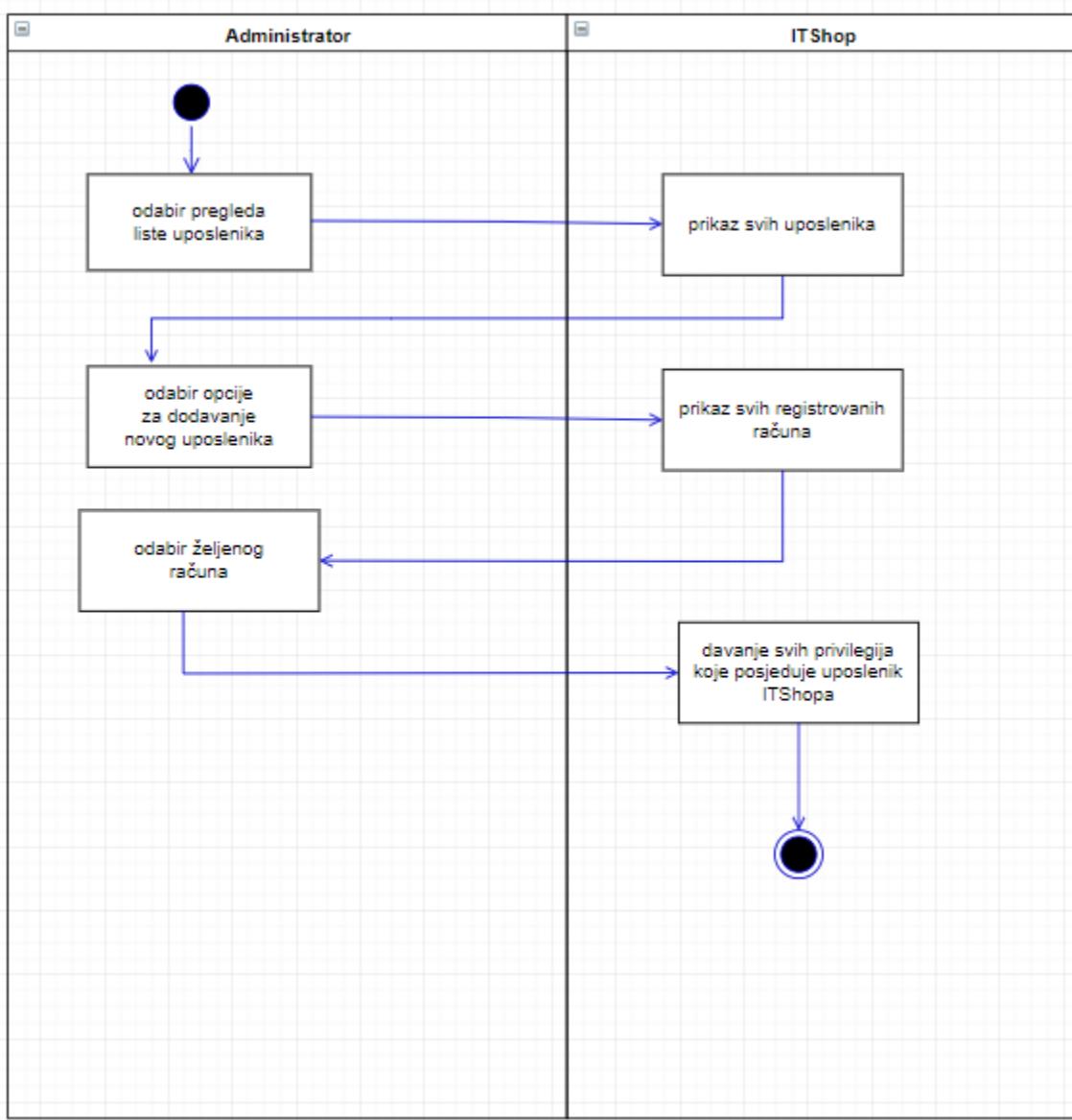
## 11) Dodavanje proizvoda



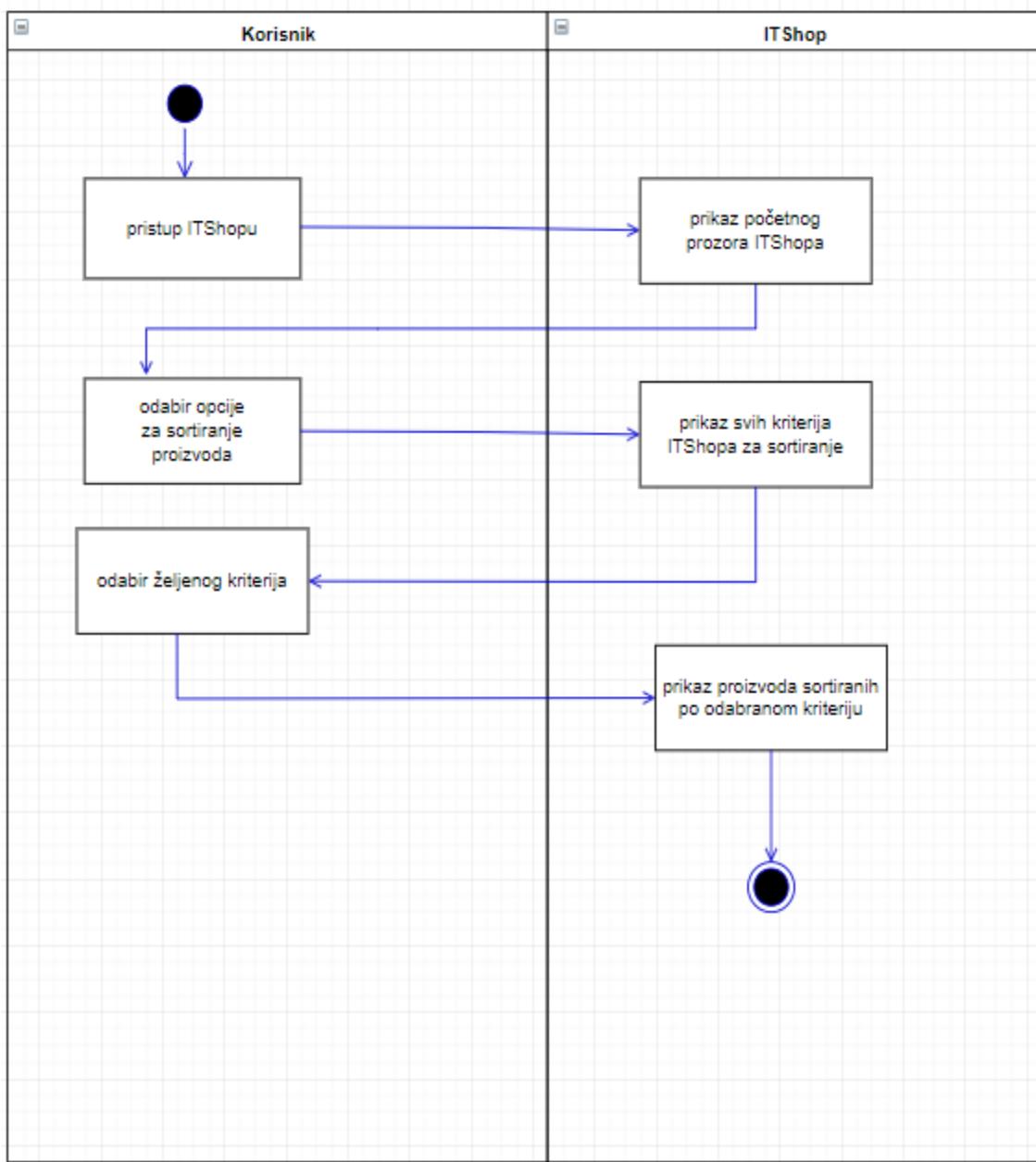
## 12) Brisanje uposlenika



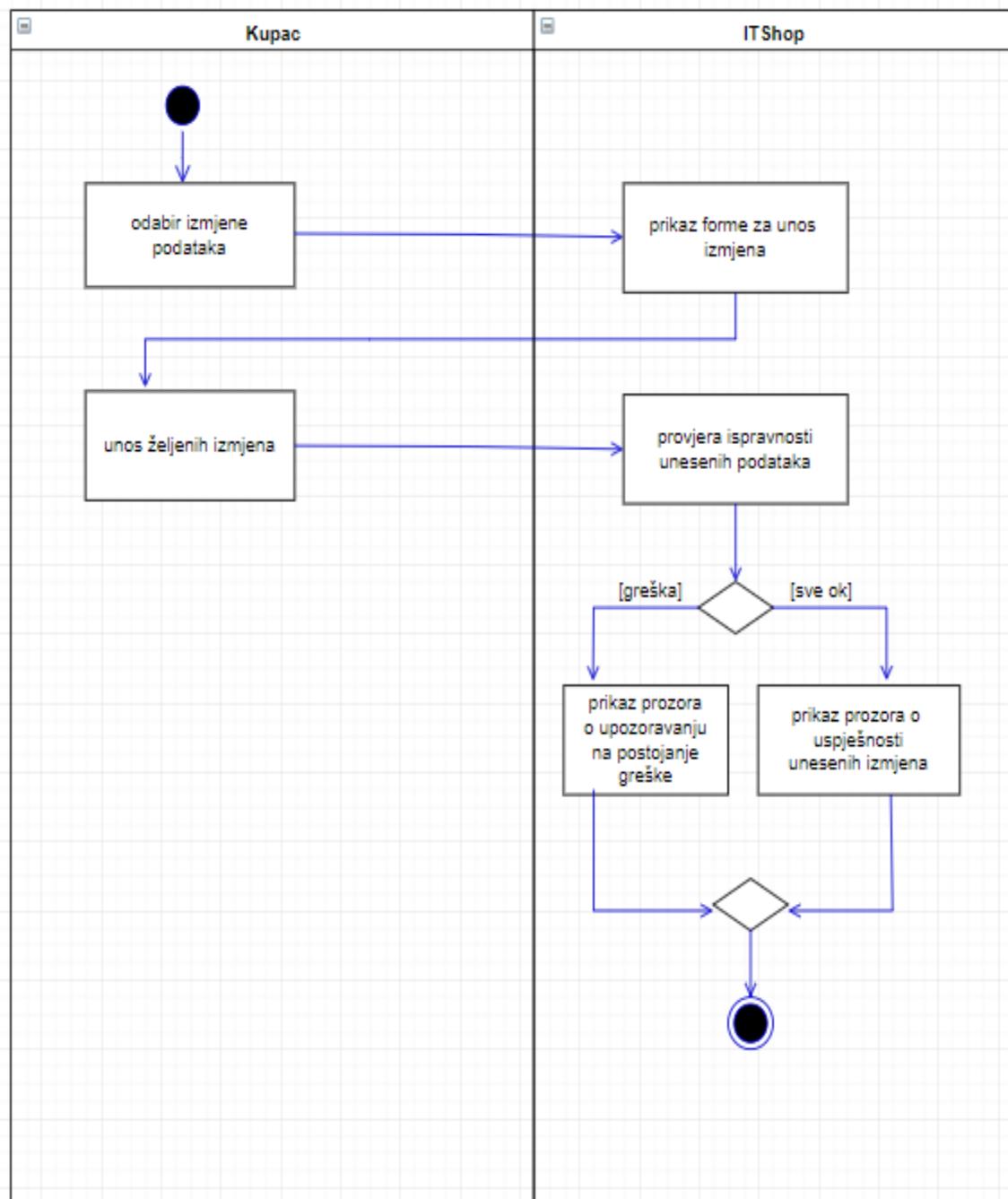
### 13) Dodavanje uposlenika



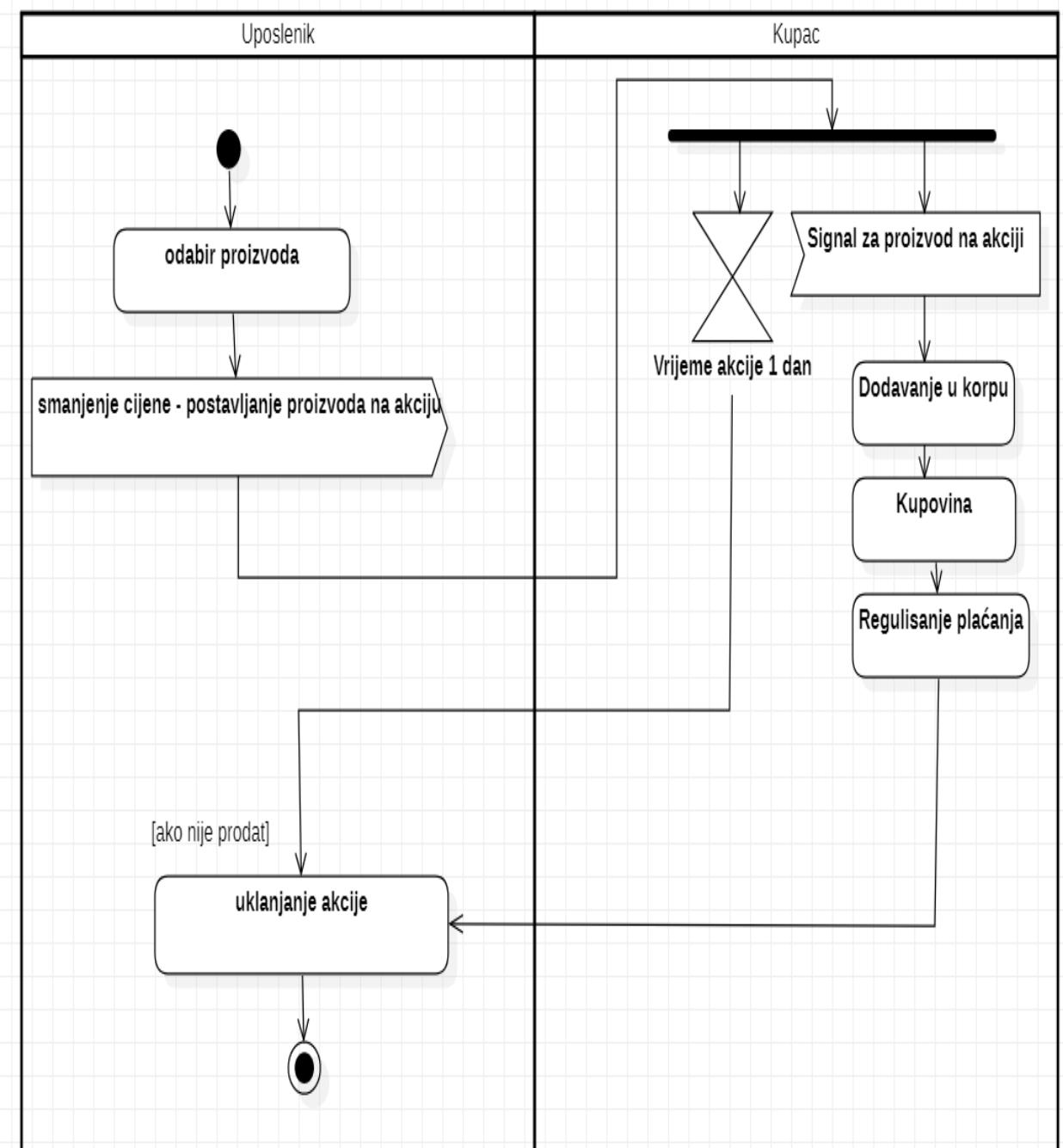
#### 14) Željeni prikaz proizvoda



## 15) Izmjena profila

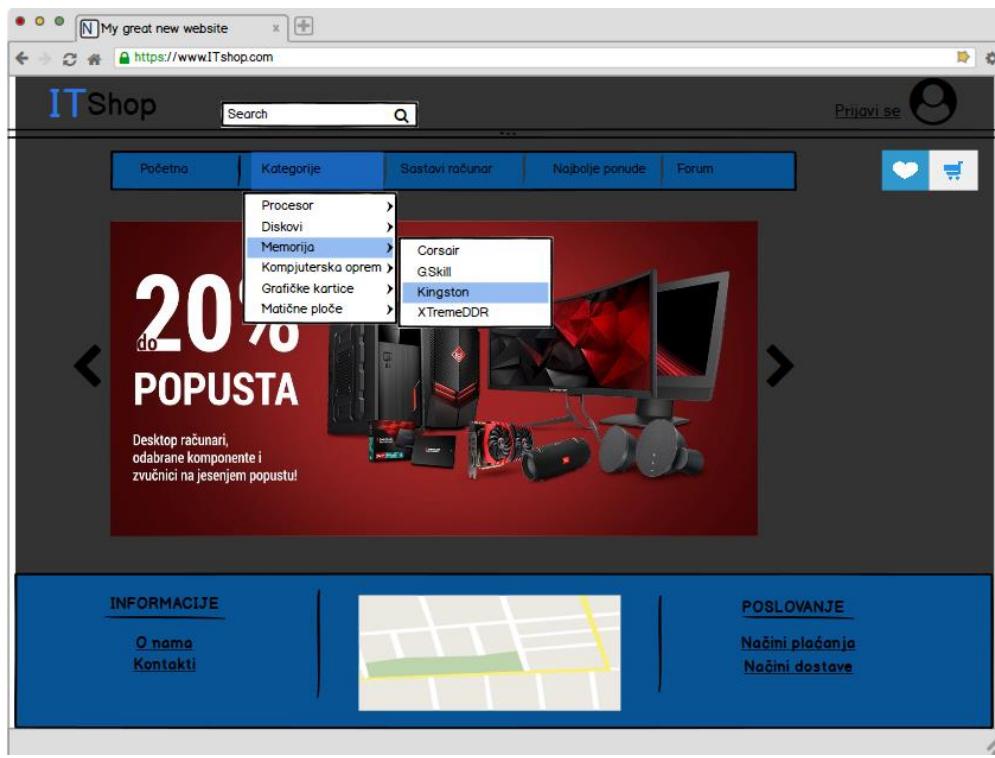


## 16) Akcija

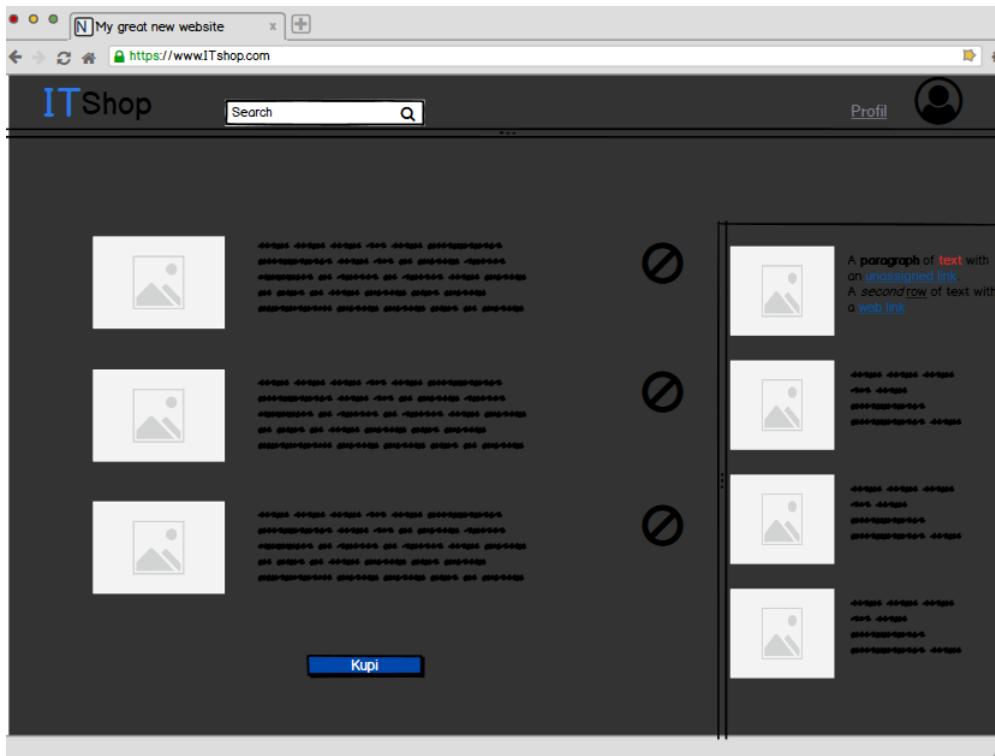


## Prototipovi grafičkih korisničkih interfejsa

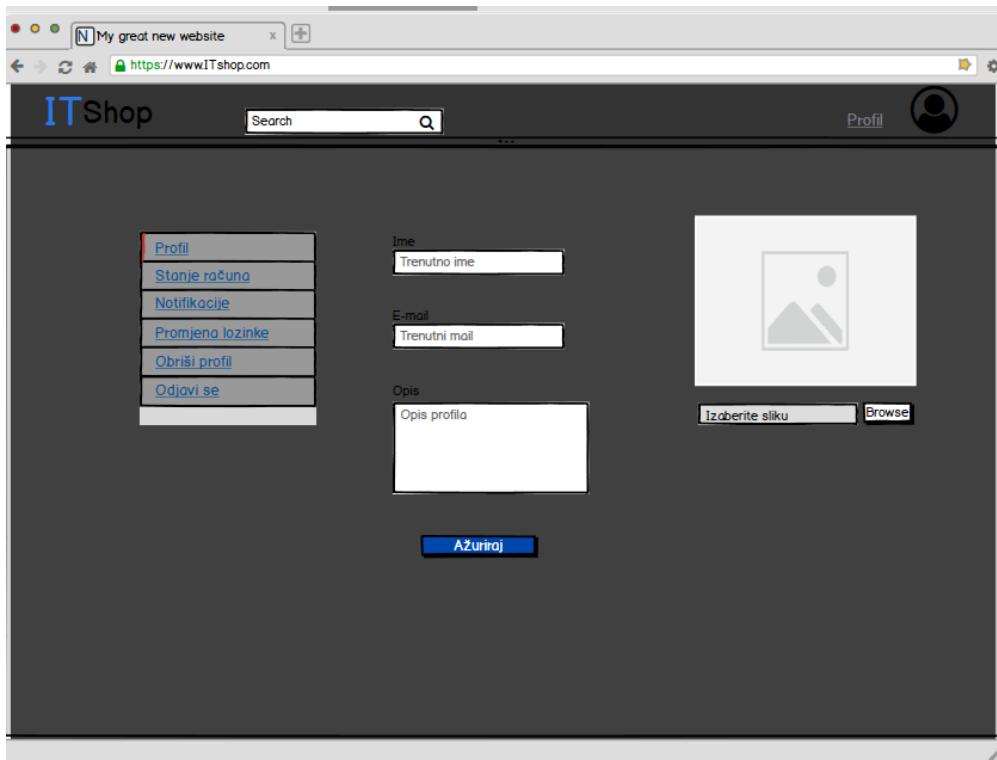
Prikaz početne stranice za neprijavljenog korisnika:



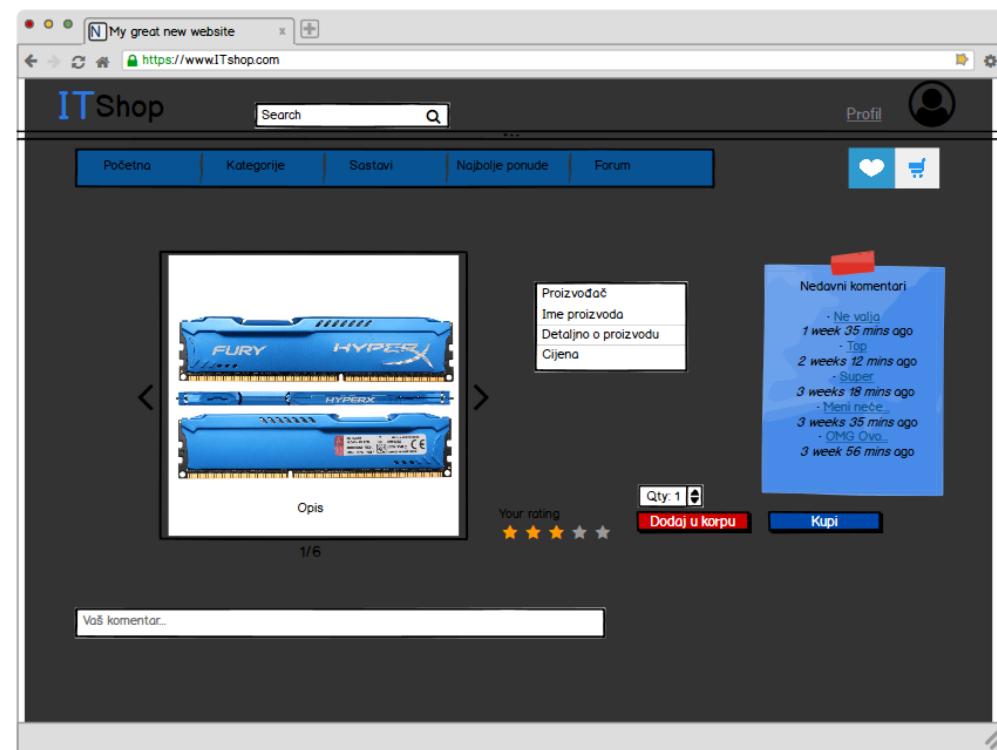
Prikaz korisnikove korpe:



Prikaz profila prijavljenog korisnika:



Prikaz proizvoda:



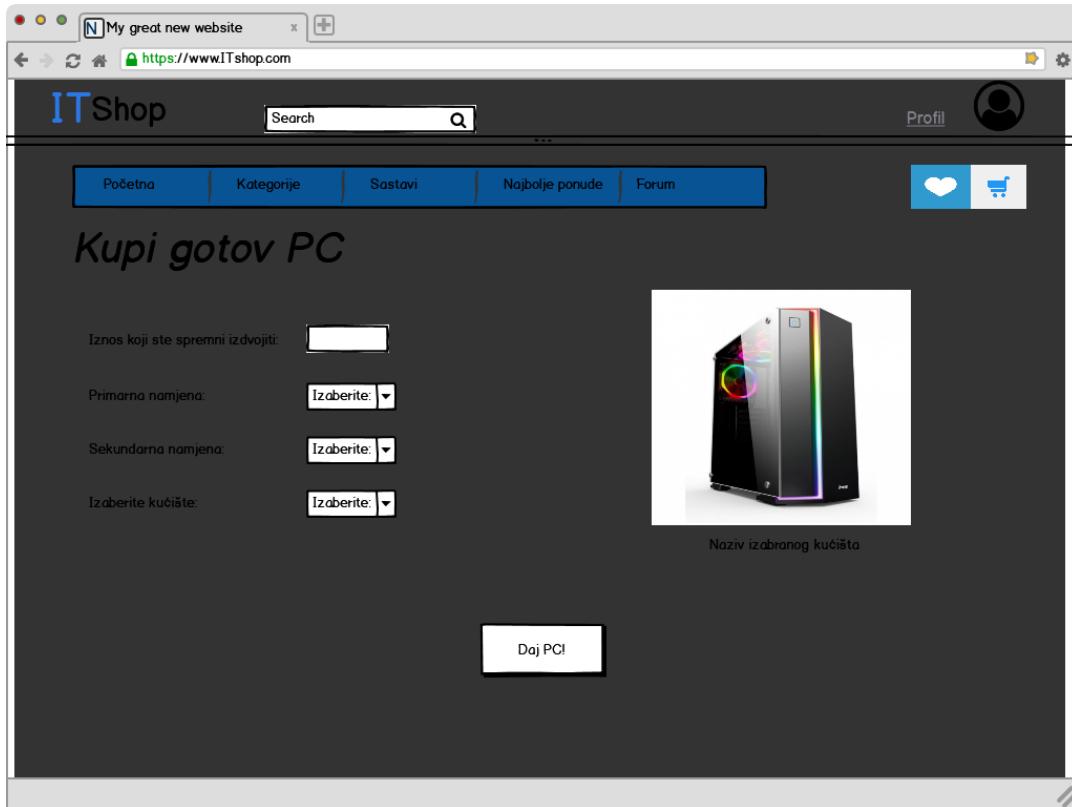
Prikaz završene narudžbe:



Prikaz upravljanja uposlenicima (opcija koju jedino ima administrator):

A screenshot of a web browser window titled 'My great new website' showing the URL 'https://www.ITshop.com'. The page has a dark theme with a blue header bar containing navigation links: 'Početna', 'Kategorije', 'Sastavi', 'Najbolje ponude', and 'Forum'. On the right side of the header is a 'Profil' icon. The main content area is titled 'Uposlenici' and features a table with three columns: 'Ime', 'Godište', and 'Datum zaposlenja'. Below the table are three buttons: 'Izmijeni', 'Dodaj', and 'Izbriši'.

Prozor za generisanje računara:



Prozor za prijavu:

The screenshot shows a login form titled 'Prijava se'. It has two input fields: 'Korisničko ime' (username) and 'Lozinka' (password), each with a corresponding input box. To the right of the password field is a small icon of a person wearing a mask. Below the input fields are two buttons: a blue 'Prijavi se' (Login) button and a red 'Odustani' (Cancel) button. To the right of the buttons is the 'it shop' logo. At the bottom of the form is a link 'Nemate račun? Registrujte se!' (Don't have an account? Register!).

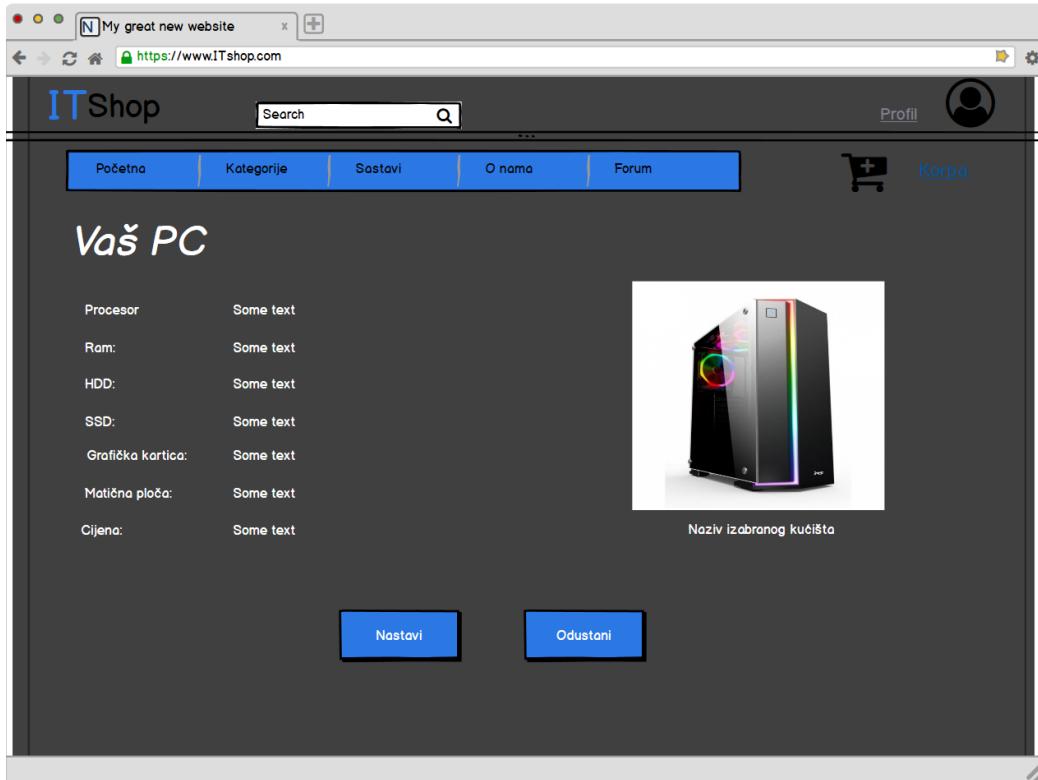
Prozor za registraciju:

The screenshot shows a web browser window titled "My great new website" with the URL "https://www.ITshop.com". The main content area is titled "ITShop" and contains a registration form. The form fields are: Ime (Name), Prezime (Last Name), Korisnicko ime (Username), Lozinka (Password), Snaga lozinke (Password strength indicator showing 1 bar), Potvrdi lozinku (Confirm Password), E-mail (Email), Datum rođenja (Birth Date) with a calendar icon, and Adresa (Address). Below the form are two buttons: "Registruj" (Register) in blue and "Nazad na početnu" (Back to homepage) in red.

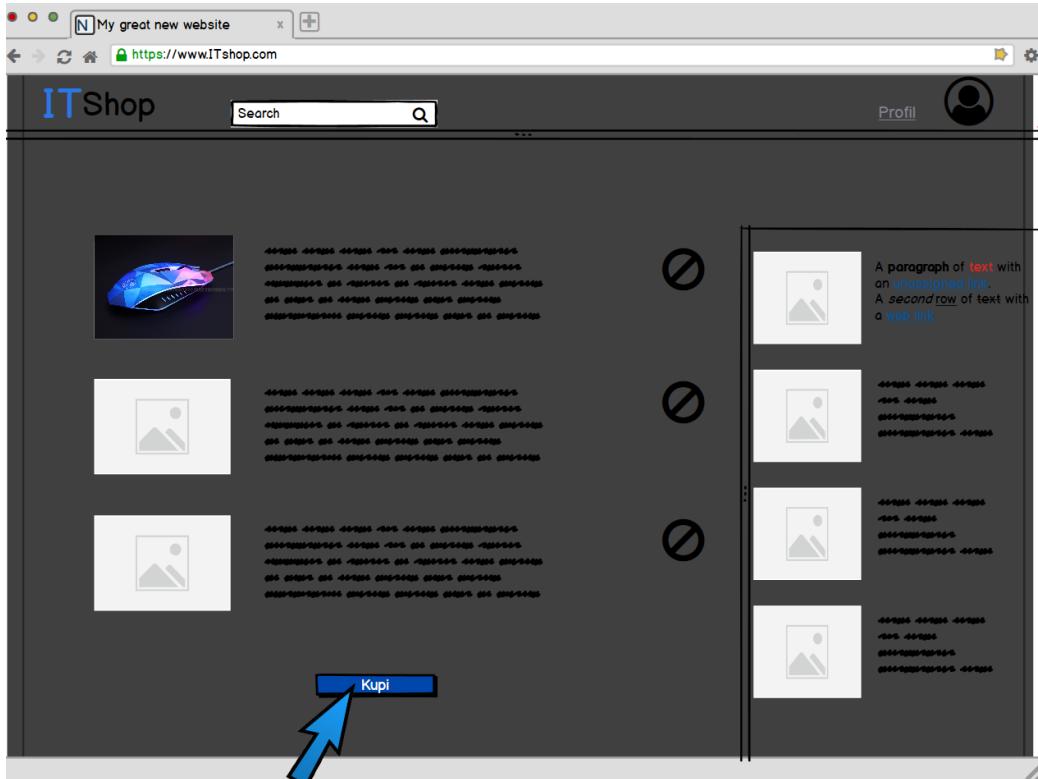
Prozor za plaćanje:

The screenshot shows a web browser window titled "My great new website" with the URL "https://www.ITshop.com". The top navigation bar includes "Početna" (Home), "Kategorije" (Categories), "Sastavi" (Components), "O nama" (About us), "Forum", and a "Profil" (Profile) icon. The main content area displays the message "Izaberite način plaćanja:" (Select payment method:) above two buttons: "Gotovinom pri dostavi" (In cash at delivery) and "Kreditnom karticom" (With credit card).

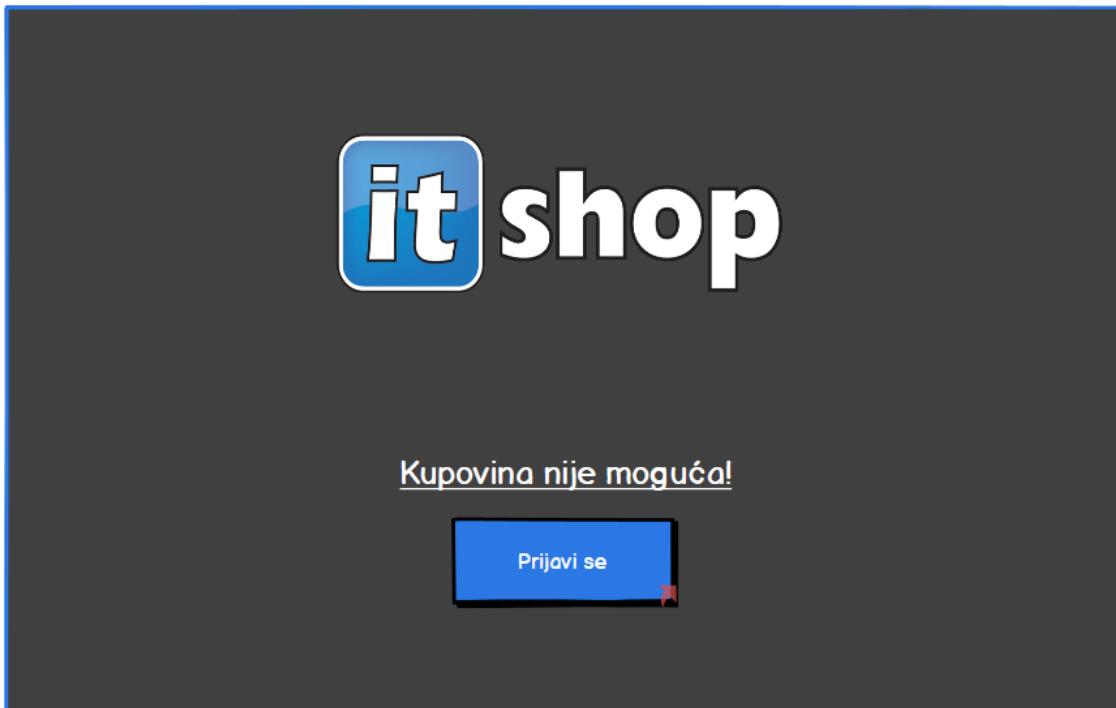
Prozor prikaza generisanog računara:



Prikaz proizvoda u korpi:



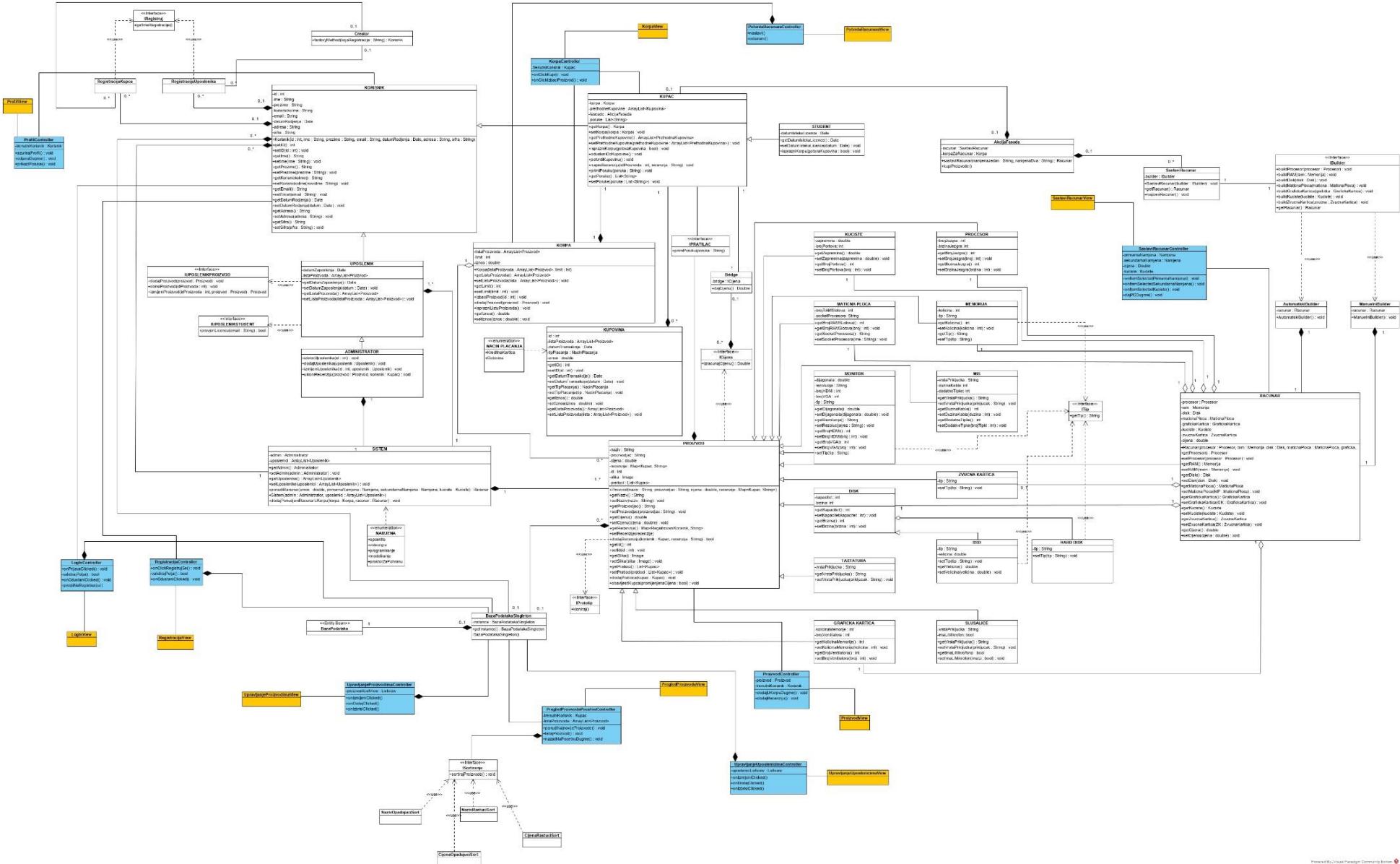
Prikaz poruke ukoliko iz nekog razloga kupovina nije moguća:



Prozor za upravljanje proizvodima (tu opciju imaju samo uposlenici i administratori):

A screenshot of a web browser window displaying the "ITShop" administration interface. The browser's address bar shows the URL "https://www.ITshop.com". The main header has the "ITShop" logo and a "Profil" link. Below the header is a navigation menu with links: Početna, Kategorije, Sastavi, O nama, and Forum. The main content area is titled "Proizvodi". It features a table with three columns: "Kategorija", "Naziv proizvoda", and "Količina". The table has 10 rows, each with a light grey background. At the bottom of the page are three buttons: "Izmjeni", "Dodatak", and "Izbriši". A blue arrow points to the "Izbriši" button.

## Klasni dijagram (Class diagram)



Na prikazanom klasnom dijagramu uključen je i MVC arhitekturalni patern. Zbog veličine dijagrama i nepreglednosti u ovom dokumentu postavljen je i [link](#) koji vodi do slike dijagrama na našem Github repozitoriju.

### SOLID principi

- **S**ingle Responsibility Principe: Svaka klasa treba imati samo jednu ulogu;

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

Posmatrajući naš dijagram klasa možemo vidjeti da je ovaj princip ispoštovan. Prije svega, veliki broj klasa implementira samo gettere i setere. Zatim, imamo naprimjer da klasa Administrator vrši tip akcija vezan za upravljanje uposlenicima, klasa Sistem upravlja tipom akcija koje pokreće korisnik, dok klasa Kupac upravlja kupovinom. Što se tiče klase Uposlenik kod nje nam se javilo pitanje "da li zna previše" budući da je prvo bitno bila zadužena i za upravljanje licencama studenata, kao i za upravljanje proizvodima, te shodno tome odlučili smo je podijeliti na IUposlenikProizvod i IUposlenikStudent da bismo osigurali da je ovaj princip i dalje ispoštovan. Također, možemo uočiti da se zbog zadovoljenja ovog principa na našem dijagramu klasa povećao broj klasa.

- **O**pen/Closed Principle: Klasa treba biti otvorena za nadogradnje, ali zatvorena za modifikacije;

*Princip O zahtijeva da klasa koja koristi neku drugu klasu ne treba biti modificirana pri uvođenju novih funkcionalnosti, ili pri potrebi za mijenjanjem druge klase.*

Ponovo, posmatrajući naš dijagram klasa možemo primjetiti da dodavanje novih metoda neće zahtjevati uređivanje već postojeće klase i njenih atributa. Naravno, klase koriste druge klase, ali to su uglavnom određene

provjere koje ne vrše nikakvu modifikaciju drugih klasa. Također, dosta klasa koristi druge klase u metodama za dodavanje objekata, međutim i to ne predstavlja problem, jer dodavanje u listu, kao generička operacija, radi neovisno o tome kakav objekat se dodaje u listu. Zbog toga, prilikom dodavanja novih funkcionalnosti dovoljno je dodati nove metode, ne razmišljajući o narušavanju prethodnih.

- **Liskov Substitution Principle:** Svaka osnovna klasa treba biti zamjenjiva svim svojim podtipovima bez da to utječe na ispravnost rada programa;

*Princip L 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.*

U našem sistemu postoji više nasljeđivanja: nasljeđivanje klasa Uposlenik i Kupac iz klase Korisnik, nasljeđivanje klase Administrator iz klase Uposlenik, nasljeđivanje klase Student iz klase Kupac, te nasljeđivanje klasa Monitor, Kuciste, HardDisk, Procesor, MaticnaPloca, SSD, ZvucnaKartica, Slusalice, Mis, GrafickaKartica, Disk, Tastatura i Memorija iz klase Proizvod. Međutim, baš zbog postojanja klasa Korisnik i Proizvod i ovaj princip je ispoštovan.

- **Interface Segregation Principle:** Bolje je imati više specifičnih interfejsa, nego jedan generalizovani;

*Princip I zahtijeva da i svi interfejsi zadovoljavaju princip S, odnosno da svaki interfejs obavlja samo jednu vrstu akcija.*

Za sada u našem sistemu postoje dva interfejsa IUposlenikProizvod i IUposlenikStudent, međutim kao što možemo vidjeti, a I po nazivu zaključiti interfejs IUposlenikProizvod sadrži metode koje obavljaju samo akcije nad proizvodom, dok interfejs IUposlenikStudent sadrži metode koje obavljaju samo akcije nad licencama i samim privilegijama studenata.

Također, bitno je napomenuti da ne odbacujemo mogućnost pojavljivanja još interfejsa prilikom olakšavanja određenih poslova i akcija u narednim koracima razvijanja ovog projekta. Kako god, i do sada ovaj princip je ispoštovan.

- **Dependency Inversion Principle:** Sistem klasa i njegovo funkcionisanje treba ovisiti o apstrakcijama, a ne o konkretnim implementacijama.

*Princip D zahtijeva da pri nasljeđivanju od strane više klasa 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.*

Budući da su klasa Korisnik i klasa Proizvod jedine klase iz kojih je nasljeđivan veći broj klasa, te obzirom na to da su one apstraktne imamo da je i ovaj princip ispoštovan.

## Paterni

### Strukturalni paterni

#### 1. ADAPTER PATERN

*Adapter patern 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.*

- Primjer upotrebe Adapter paterna:

Što se tiče Adapter paterna, još uvijek nismo uočili primjer za njegovu upotrebu u našem programu.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:

/

## 2. 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.

- Primjer upotrebe Facade paterna:

Što se tiče Facade paterna, primjer za njegovu upotrebu smo uočili kod sastavljanja idealnog računara. Naime, na kupcu je samo da unese specifikacije željenog računara, a način na koji se taj računar sastavlja je potpuno skriven od kupca koji vidi samo krajnji rezultat. Također i kod same opcije kupovine, kupac treba da vidi samo upozorenje o uspješnoj kupovini, a uklanjanje proizvoda iz korpe, regulisanje nove količine proizvoda, te eventualno uklanjanje proizvoda iako se dešavaju u pozadini, njihov način izvršavanja kupca se ne tiču, te trebaju biti skriveni od njega.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:

1. Definisati novu klasu AkcijeFasada koja će implementirati metode sastaviRačunar() i kupiProizvode();

2. Promijeniti implementaciju kupca na način da se samo pozivaju gotove metode novodefinisane klase;

### 3. DECORATOR PATERN

Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu object (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.

- Primjer upotrebe Decorator paterna:

Što se tiče Decorator paterna, primjer za njegovu upotrebu još uvijek nismo uočili u našem programu.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:

/

### 4. 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 klasa u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama.

- Primjer upotrebe Bridge paterna:

Što se tiče Bridge paterna, mogući primjer za njegovu upotrebu smo uočili kod same cijene proizvoda koji se nalaze u korpi ili koje je kupac odlučio

da kupi. Naime, kupac u našem ITShopu može biti Student pri čemu se ukupna cijena računa uzimajući u obzir popust, ali uvijek sa istom osnovicom (navedenom cijenom proizvoda). To nam ne bi stvaralo problem ukoliko bi željeli uvesti popuste i sniženja za još neke skupine kupaca (npr. kupcima koji su već određeni period registrovani na naš ITShop), međutim ukoliko bismo se odlučili u budućnosti dodati mogućnost raznih saradnji, što bi dovodilo do različitih dogovora oko određenih cijena (što je danas jako popularno kod online shoppinga) uvođenje ovog paterna bi nam znatno pomoglo pri prikazu cijene na ispravan način, jer bi to zahtjevalo samo novu metodu (novu apstrakciju), na koju će se naslanjati konkretna implementacija izračuna cijene za takve kupce.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:
  1. Dodati novi interfejs ICijena koji će sadržavati definiciju metode za izračun cijene određenog proizvoda;
  2. Dodati novu klasu Bridge, koja će sadržavati apstrakciju i kojoj će jedino kupac imati pristup;

## 5. 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.

- Primjer upotrebe Composite paterna:

Što se tiče Composite paterna, primjer za njegovu upotrebu smo uočili kod samih proizvoda. Posmatrajući naš dijagram klase, uvidjeli smo kod klase proizvoda da ih jako dosta koristi istu metodu getTip(). Bez obzira na različite ishode napisane metode, svakoj je potrebno pristupati na isti

način, te uvođenjem Composite paterna smatramo da bi se znatno pojednostavio i smanjio program.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:
  1. Definisati interfejs ITip koji će sadržavati definiciju metode za dobivanje tipa proizvoda;
  2. Naslijediti ga od strane svih klasa proizvoda koje su sadržavale prethodno napisanu metodu getTip();

## 6. PROXY PATERN

Proxy patern 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.

- Primjer upotrebe Proxy paterna:

Što se tiče Proxy paterna, primjer za njegovu upotrebu smo uočili kod korisnika.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:

## 7. 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

instance objekta (tzv. bezlično stanje). Korištenje ovog paterna veoma je korisno u slučajevima kada je potrebno vršiti uštedu memorije.

- Primjer upotrebe Flyweight paterna:

Što se tiče Flyweight paterna, primjer za njegovu upotrebu smo uočili kod upravljanja proizvodima.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:

## Kreacijski paterni

### SINGLETON PATERN

Singleton patern služi kako bi se neka klasa mogla instancirati samo jednom. Na ovaj način može se omogućiti i tzv. lazy initialization, odnosno instantacija klase tek onda kada se to prvi put traži. Osim toga, osigurava se i globalni pristup jedinstvenoj instanci - svaki put kada joj se pokuša pristupiti, dobiti će se ista instanca klase. Ovo olakšava i kontrolu pristupa u slučaju kada je neophodno da postoji samo jedan objekat određenog tipa

- Primjer upotrebe Singleton paterna:

Što se tiče Singleton paterna, primjer za njegovu upotrebu u našem program smo uočili kod klase Administrator. Naime, korisnik posjeduje mogućnost pristupa našem ITShopu, kojem pored korisnika pristupaju uposlenici i jedan administrator,

koji ima punu moć nad ITShopom. Naprimjer, administrator ITShopa je vlasnik ITShopa, koji se ne može promijeniti sve dok on to sam ne odluči.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:
  1. Dodati statički atribut u klasu AdministratorSingleton tipa AdministratorSingleton, koji se označava kao statički;
  2. Dodati statičku metodu dajAdministratora () u klasi AdministratorSingleton koja će vršiti vraćanje jedinstvenog statičkog atributa iz ove klase;
  3. Dodati metodu postaviAdministratora() u klasu AdministratorSingleton koja će kontrolisati promjenu administratora ukoliko za to bude potrebno.

## PROTOTYPE PATERN

Prototype patern omogućava smanjenje kompleksnosti kreiranja novog objekta tako što se uvodi operacija kloniranja. Na taj način prave se prototipi objekata koje je moguće replicirati više puta a zatim naknadno promijeniti jednu ili više karakteristika, bez potrebe za kreiranjem novog objekta nanovo od početka. Ovime se osigurava pojednostavljenje procesa kreiranja novih instanci, posebno kada objekti sadrže veliki broj atributa koji su za većinu instanci isti.

- Primjer upotrebe Prototype paterna:

Što se tiče Prototype paterna, primjer za njegovu upotrebu smo uočili između klasa Kupovina i Uposlenik. Naime, nakon kupovine, uposlenik mora da reguliše količine kupljenih

proizvoda koje su trenutno na stanju, te smatramo da bi upotreba ovog paterna smanjila pristup bazi podataka.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:
  1. Definisati interfejs IPrototip koji se sastoji od metode kloniraj();
  2. Naslijediti interfejs IPrototip od klase Kupovina te implementirati metodu kloniraj() koja će kreirati duboku kopiju objekta;
  3. Promijeniti implementaciju uposlenika tako da se vrši kloniranje proizvoda pri njegovom brisanju.

## FACTORY METHOD PATERN

Factory method patern služi za omogućavanje instanciranje različitih vrsta podklasa koristeći factory metodu koja odlučuje koja će se podklasa instancirati i koja programska logika izvršiti. Na ovaj način osigurava se ispunjavanje O SOLID principa, jer se kod za kreiranje objekata različitih naslijedenih klasa ne smješta samo u jednu zajedničku metodu, već svaka podklasa ima svoju logiku za instanciranje željenih klasa, a samo instanciranje kontroliše factory metoda koju različite klase implementiraju na različit način.

- Primjer upotrebe Factory Method paterna:

Što se tiče Factory Method paterna, primjer za njegovu upotrebu smo uočili prilikom same registracije, jer na osnovu informacija od strane korisnika moguće je instancirati objekte različitih tipova (Kupac, Uposlenik).

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:
  1. Definisati interfejs za registariju – Iregistroacija;
  2. Definisati klase RegistracijaKupac, RegistracijaUposlenik koje implementiraju interfejs;
  3. Definisati klasu Creator koja posjeduje FactoryMethod() metodu koja odlučuje koju klasu instancirati

## ABSTRACT FACTORY PATERN

Abstract factory patern služi kako bi se izbjeglo korištenje velikog broja if-else uslova pri kreiranju različitih hijerarhija objekata. Ukoliko postoji više tipova istih objekata te različite klase koriste različite podtipove, te klase postaju fabrike za kreiranje objekata zadanog podtipa bez potrebe za specificiranjem pojedinačnih objekata. Na ovaj način se, korištenjem nasljeđivanja, ukida potreba za postojanjem if-else uslova jer određeni tip fabrike sadrži određene tipove objekata i zna se tačno koju podklasu će instancirati.

- Primjer upotrebe Abstract Factory paterna:

Što se tiče Abstract Factory paterna još uvijek nismo sigurni za primjer za njegovu upotrebu.
- Kako bi se to postiglo, potrebno je izvršiti sljedeće:

## BUILDER PATERN

Builder patern služi za apstrakciju procesa konstrukcije objekta, kako

bi se kao rezultat moglo dobiti različite specifikacije objekta koristeći isti proces konstrukcije. Ovaj patern koristi se kako bi se izbjeglo kreiranje kompleksne hijerarhije klasa te kako bi se izbjegao kompleksni programski kod konstruktora jedne klase koja može imati različite konfiguracije atributa. Različiti dijelovi konstrukcije objekta izdvajaju se u posebne metode koje se zatim pozivaju različitim redoslijedom ili se poziv nekih dijelova izostavlja, kako bi se dobili željeni različiti podtipovi objekta bez potrebe za kreiranjem velikog broja podklasa.

- Primjer upotrebe Builder paterna:

Što se tiče Builder paterna, primjer za njegovu upotrebu smo uočili kod sastavljanja idealnog računara. Odlučili smo da iskoristiti kod klase Računar jer bi njegovo korištenje znatno smanjilo komplikovanost ove klase, te omogućili bismo jednostavno korištenje različitih dijelova procesa konstrukcije računara za dobivanje različitih rezultnih proizvoda.

- Kako bi se to postiglo, potrebno je izvršiti sljedeće:

1. Dodati interfejs IBuilder koji će omogućiti implementaciju različitih metoda za različite načine građenja objekata računara;
2. Dodati klase ManuelniBuilder i AutomatskiBuilder koje će na različit način implementirati metode građenja računara;
3. Povezati kupca sa interfejsom IBuilder, a ne direktno sa klasom Računar, jer on neće direktno graditi ovaj objekat.

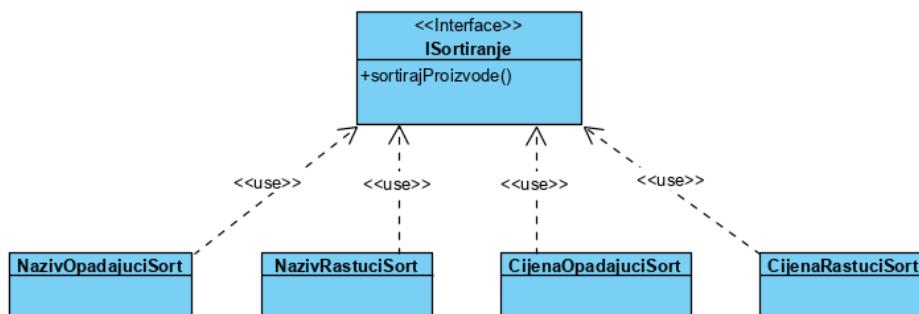
## Paterni ponašanja

### 1. STRATEGY

Ovaj pattern se koristi ukoliko se jedan problem može riješiti sa više različitih algoritama. Najbolji primjer upotrebe ovog paterna možemo vidjeti kod algoritama sortiranja kojih postoji zaista mnogo, kao što su quick sort, merge sort, radix sort, selection sort itd.

Ovaj patern smo iskoristili za sortiranje proizvoda, pri čemu se sortiranje vrši na različite načine. Korisnik bira kategoriju po kojoj će se vršiti sortiranje. Dostupne kategorije su: abecedno, po cijeni itd. Pomoću ovog paterna neće biti teško dodati nove metode za sortiranje ukoliko se za njima ukaže potreba.

Potrebno je dodati interfejs (ISortiranje) koji će imati navedenu metodu sortirajProizvode te implementirati klase koje će izvesti taj interfejs. Promjene su prikazane na slici ispod:



Još je potrebno dodati metodu za sortiranje u klasi koja sadrži listu proizvoda. Osim te metode potrebno je dodati i atribut strategija tipa ISortiranje koji će biti indikator koji će se sort koristiti te će se za njega napraviti set metoda.

### 2. STATE

Sličan je Strategy paternu, tačnije predstavlja njegovu dinamičku verziju. On se koristi pri promjeni stanja objekta neke klase. Postiže se promjenom podklase unutar hijerarhije klasa. Za ovaj patern nismo pronašli primjenu. Međutim, ukoliko bi se odlučili na dodavanje neke funkcionalnosti kod koje bi bilo bitno npr. da li je korpa prazna ili ne, mogli bi popunjenoš korpe posmatrati kao dva različita stanja te bi se ovaj patern mogao primijeniti.

### 3. TEMPLATEMETHOD

Koristi se za izdvajanje koraka algoritma u podklase. Većinom se upotrebljava u sličnim situacijama kao i Strategy patern. Ovaj patern nismo iskoristili ali bi se npr. mogao iskoristiti prilikom plaćanja. Imamo klasu Kupac i klasu Student. Algoritam plaćanja im se razlikuje ali u suštini bi obavljao istu stvar. Mogli bismo taj algoritam izdvojiti u posebnu klasu (mogao bi se koristiti interfejs), koja bi imala metodu za plaćanje. Klasa Kupac i klasa Student bi imale različite implementacije ove metode jer bi se kod klase Student vršio izračun popusta kojeg nemamo kod kupca.

### 4. OBSERVER

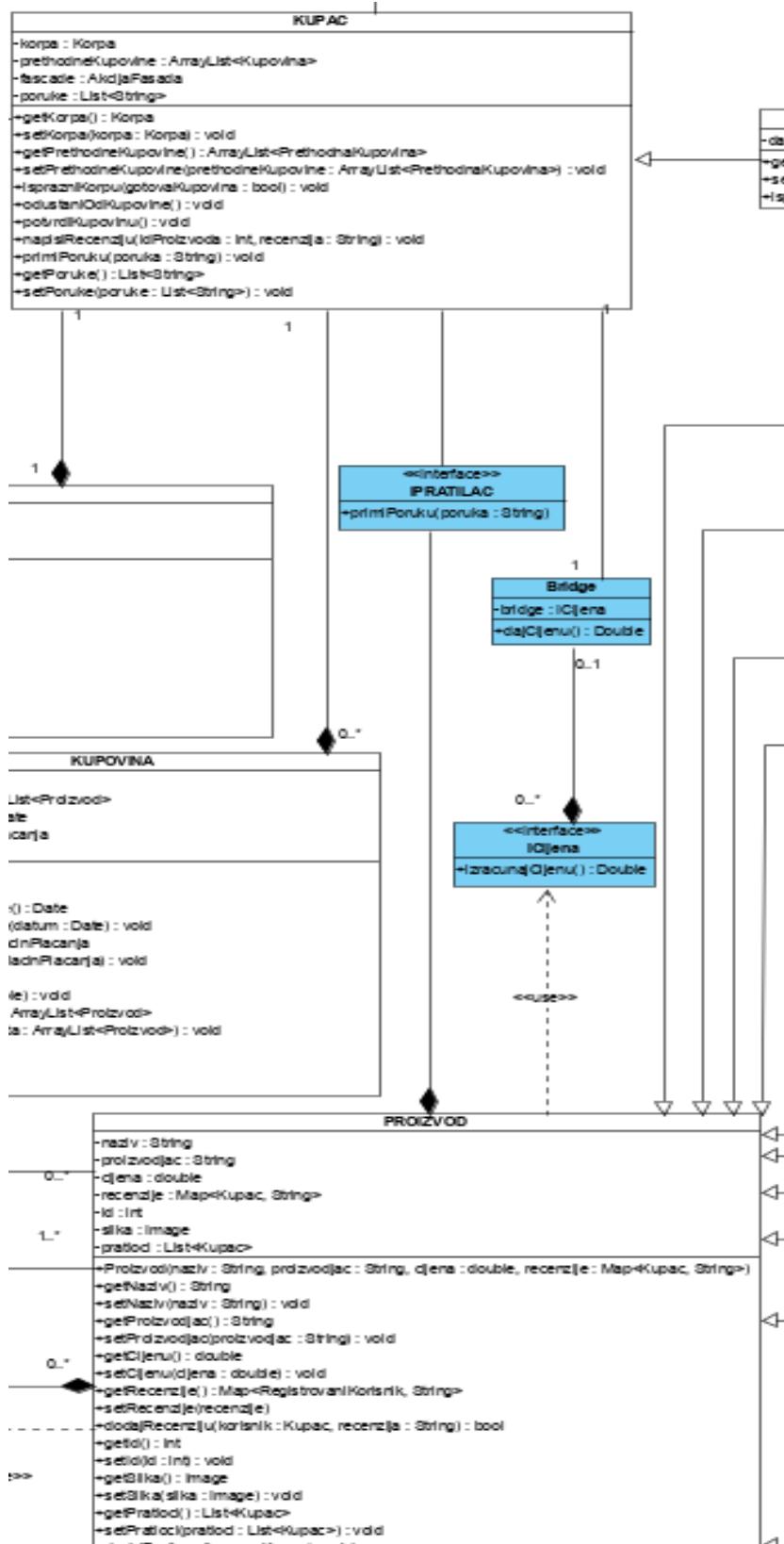
On se koristi da bi se na jednostavan način napravio mehanizam za pretplatu. Pretplatnici dobivaju obavještenja o sadržajima na koje su pretplaćeni, a za slanje obavještenja zadužena je nadležna klasa. Na ovaj način uspostavlja se relacija između klasa kako bi se mogle prilagoditi međusobnim promjenama.

Ovaj pattern smo iskoristili tako što smo dodali potrebnu funkcionalnost. Zamisao je da kupac može uključiti obavijesti za neki proizvod iz naše baze, te kad god se promijeni stanje tog proizvoda (misli se na promjenu dostupne količine ili trenutne cijene) pošalje se obavijest kupcu.

Kao prvo, potrebno je u klasi Proizvod dodati listu kupaca (kao atribut) koji prate posmatrani proizvod. Nazovimo taj atribut pratioci. U klasi Proizvod je potrebno dodati i metodu koja obavještava o promjeni stanja proizvoda, nazovimo je obavijestiKupca. U klasi Kupac je potrebno dodati metodu azuriraj koja će poslati poruku kupcu preko njegovog UI-a. Zbog ove funkcionalnosti smo odlučili dodati i atribut „poruke“ u klasu Kupac koji će čuvati sve pristigle poruke za kupca.

Također je potrebno napraviti interfejs koji sadrži metodu azuriraj koja će se upravo i preklopiti u klasi Kupac, tj. klasa Kupac će implementirati ovaj interfejs.

Novododane dijelove je potrebno dodati i u dijagram klasa:

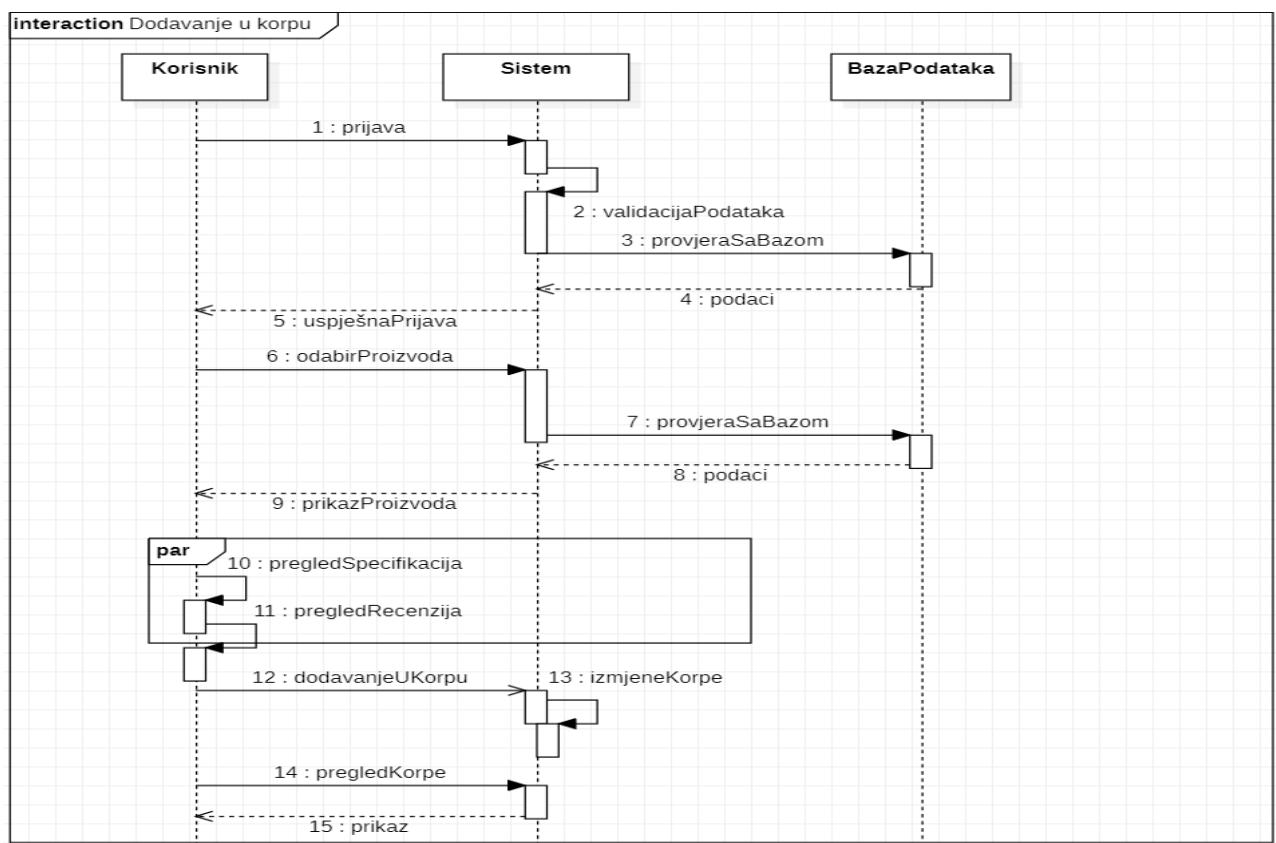


## 5. ITERATOR

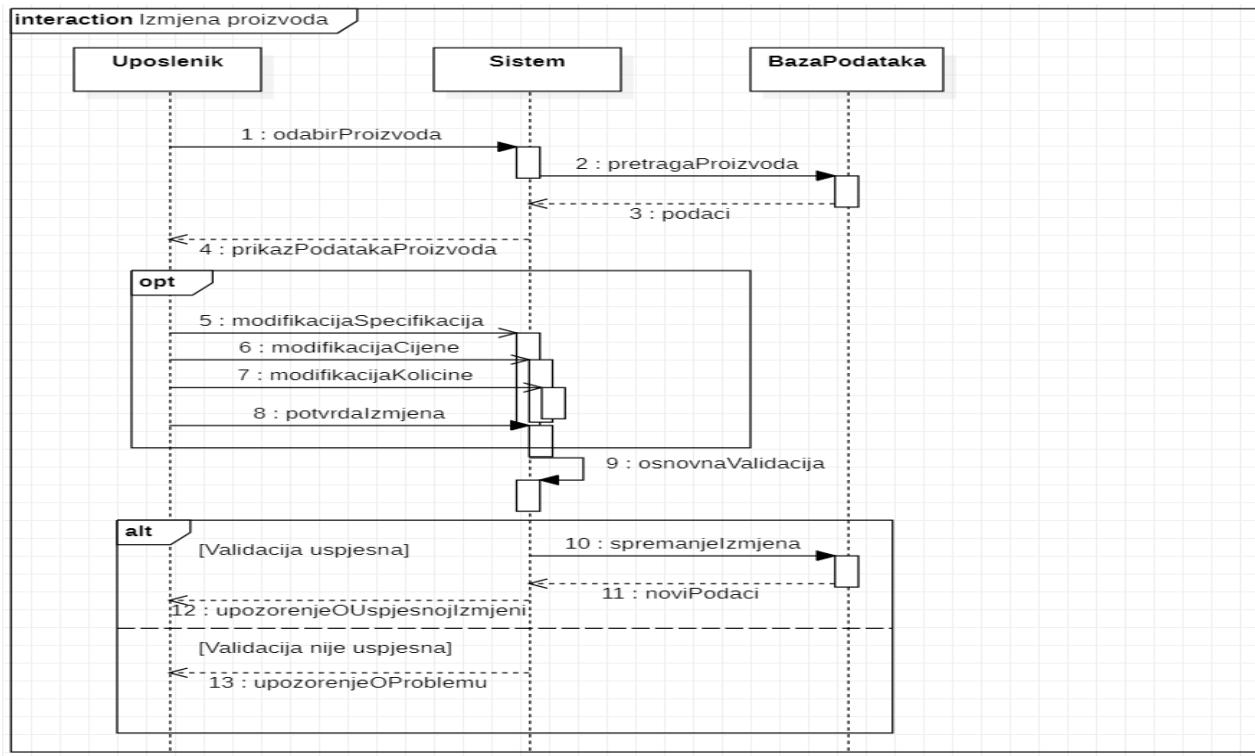
Koristi se za sekvenčniji pristup elementima kolekcije bez poznavanja njene strukture. Ovaj patern preporučljivo je iskoristiti kada se za iteriranje koristi kompleksna logika koja ovisi o više kriterija. Ovaj patern bismo mogli implementirati ukoliko bismo npr. imali funkcionalnost da se kupcu nude proizvodi po nekom specifičnom redoslijedu, npr. neki „shuffle“ poredak ili slično. Pošto tu funkcionalnost nemamo, za ove dosadašnje funkcionalnosti smatramo da nema prevelike potrebe za ovim paternom iz razloga što su nam liste većinom po nekom uobičajenom redoslijedu (abecedno, po cijeni, po vrsti proizvoda i sl.).

### Dijagrami sekvenci (Sequence diagrams)

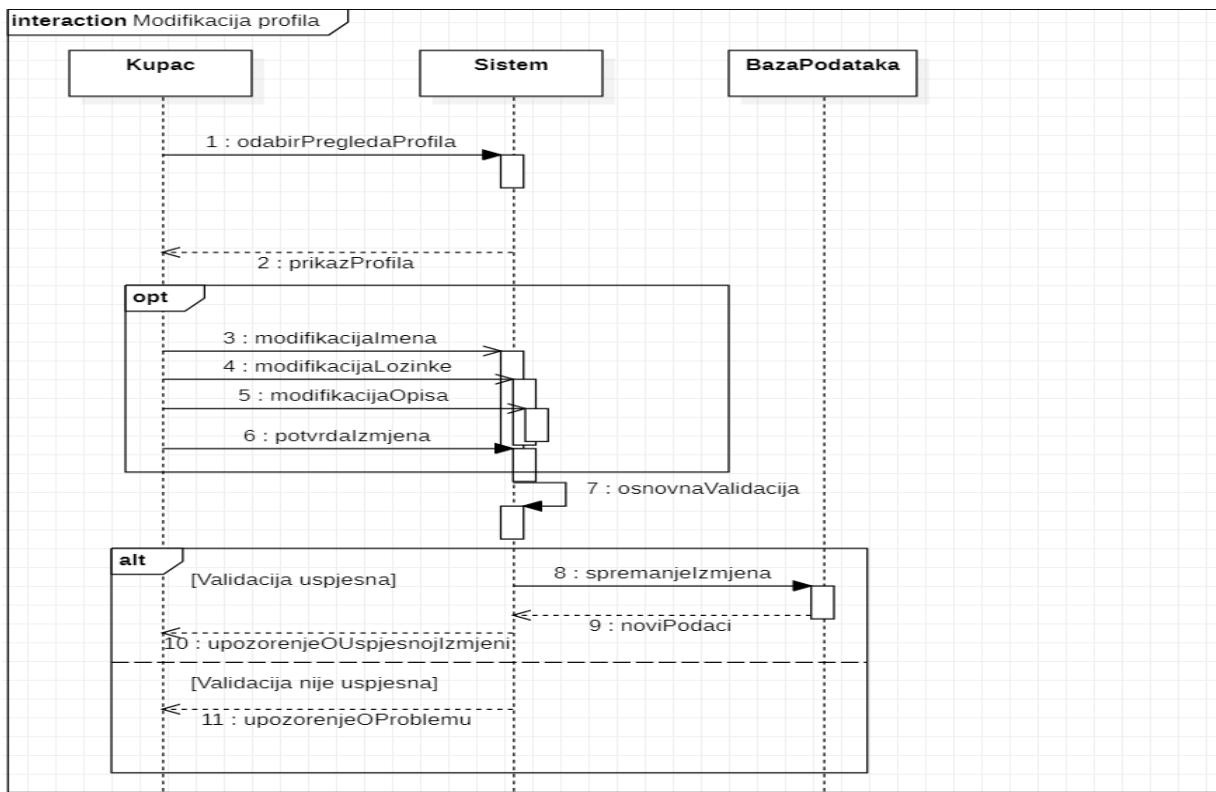
Sekvenca za dodavanje proizvoda u korpu:



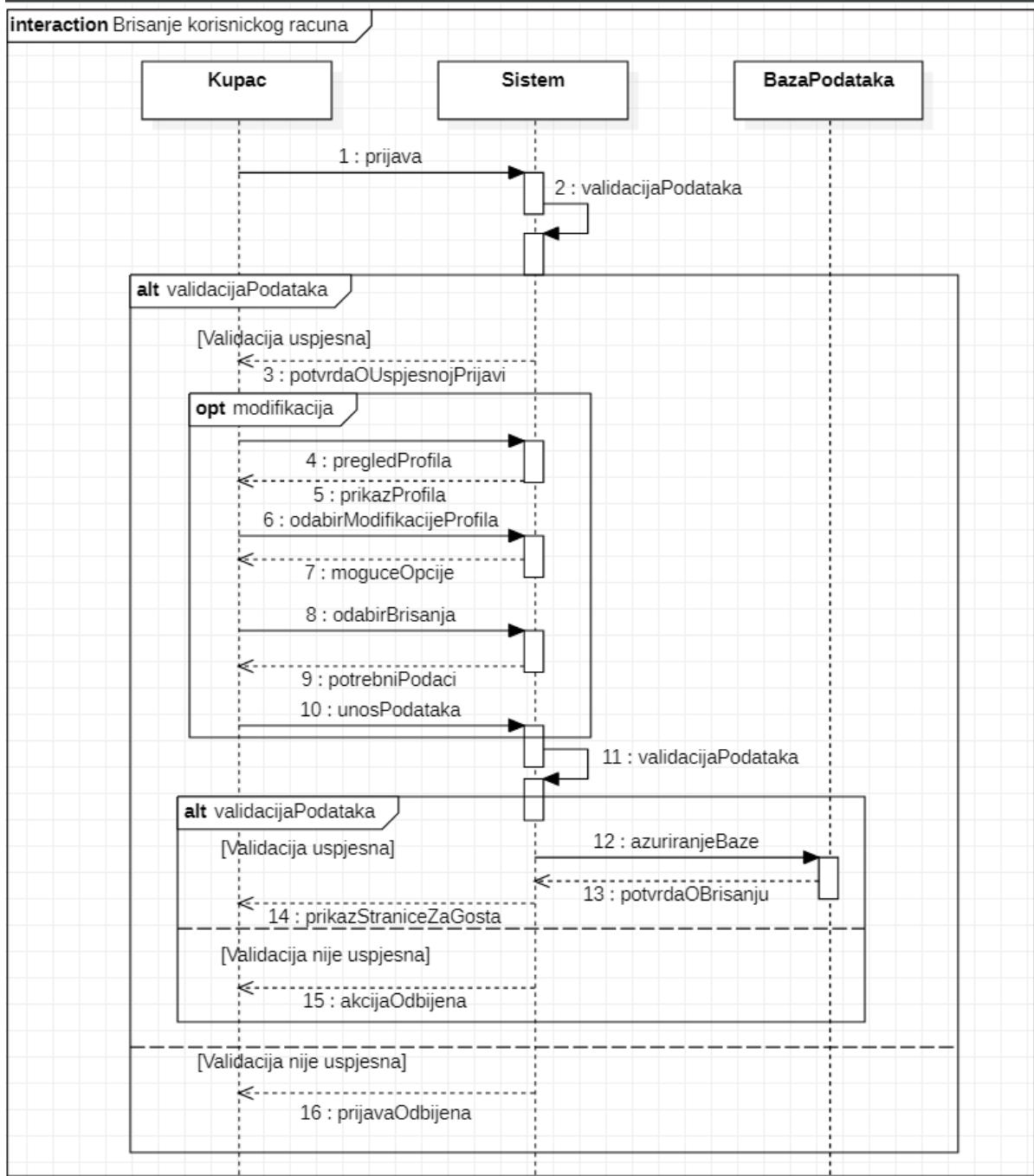
Za izmjenu proizvoda:



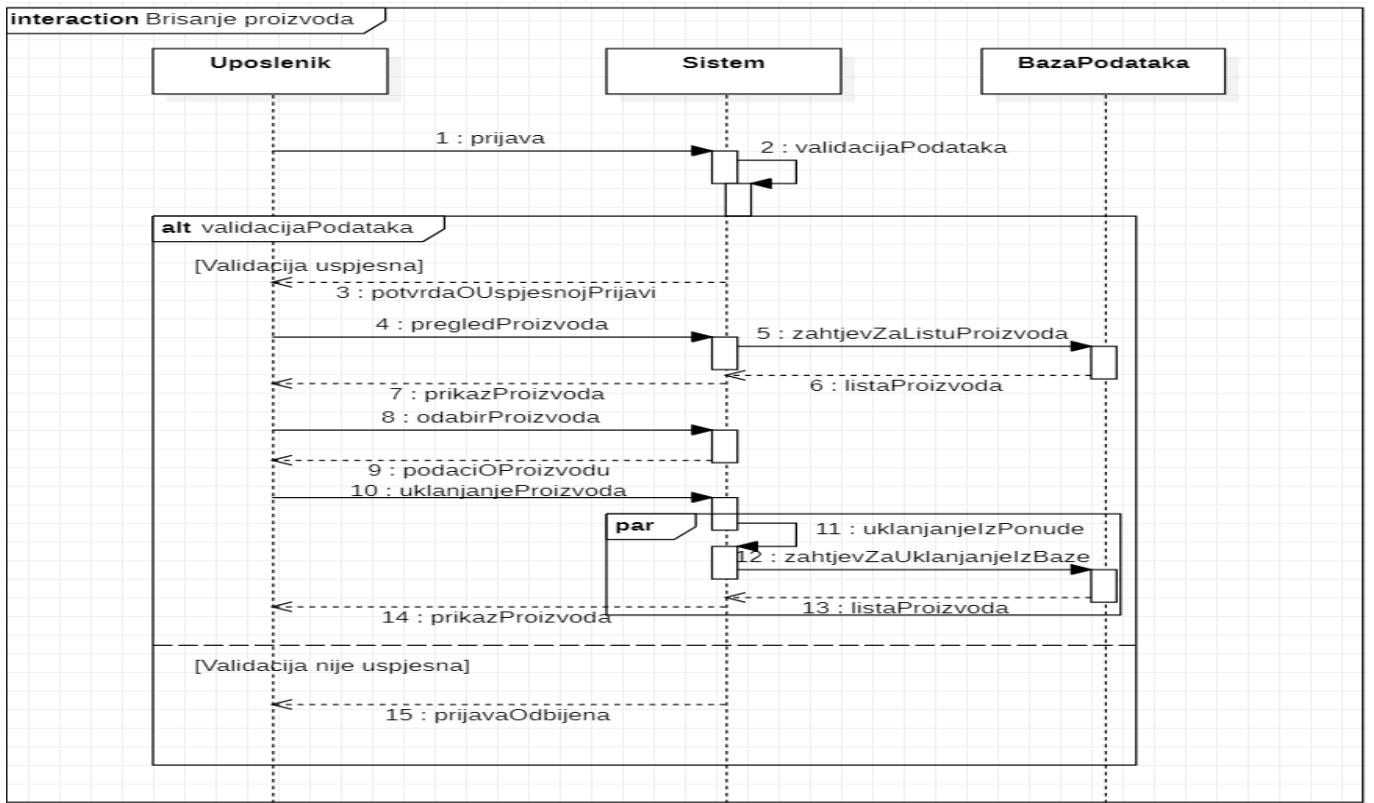
Za izmjenu profila korisnika:



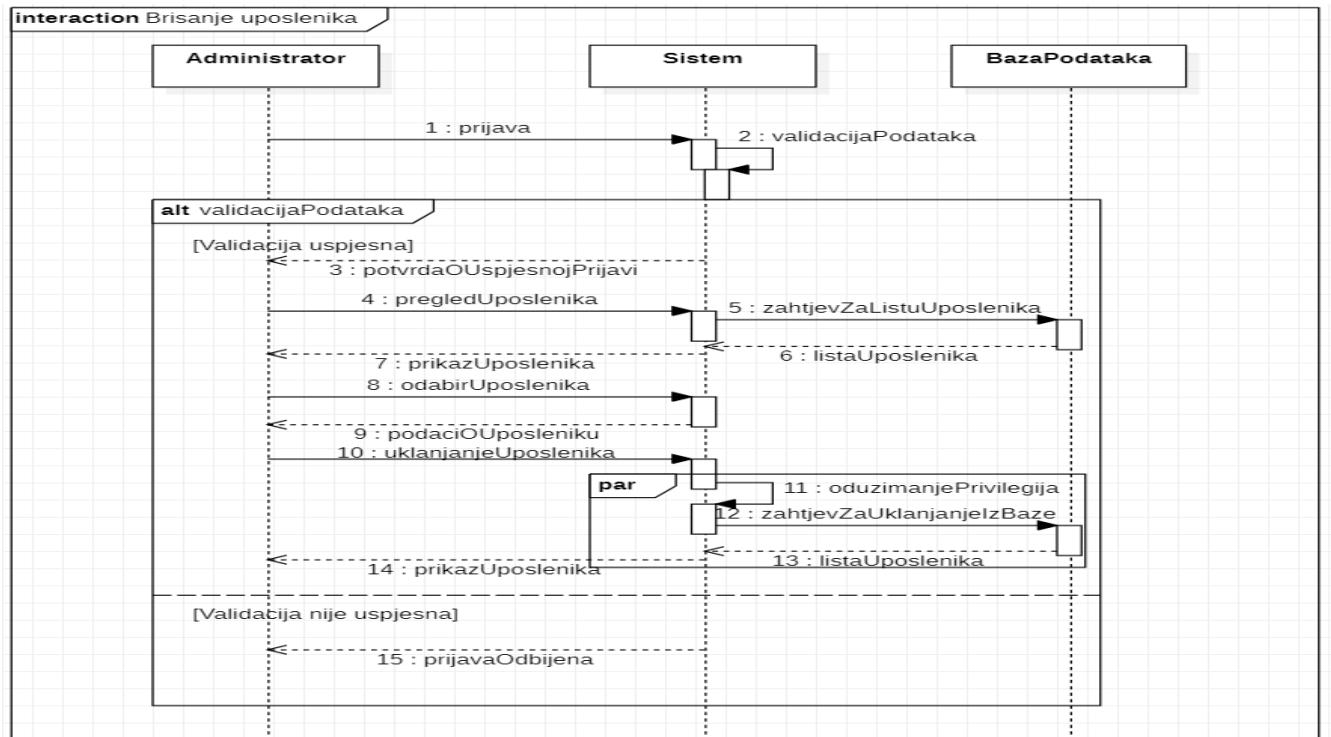
Za brisanje korisničkog profila:



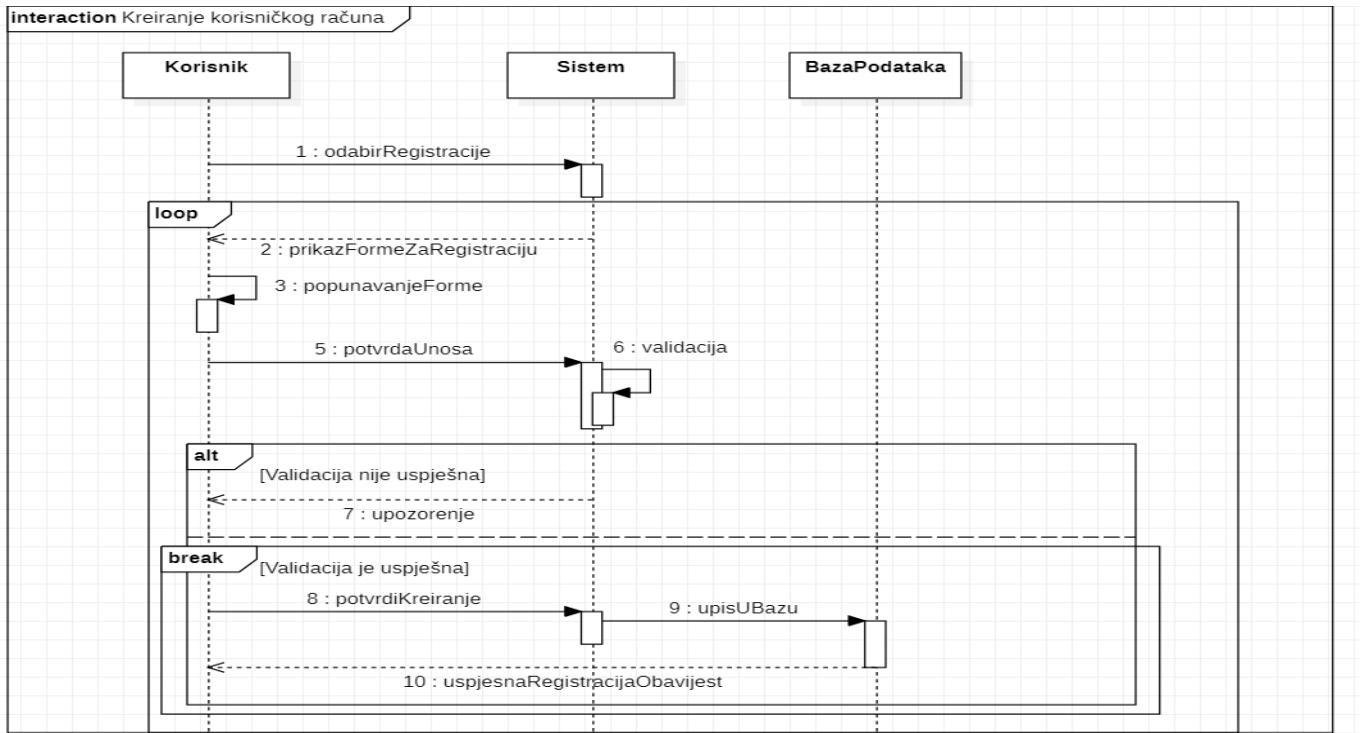
Za bisanje proizvoda:



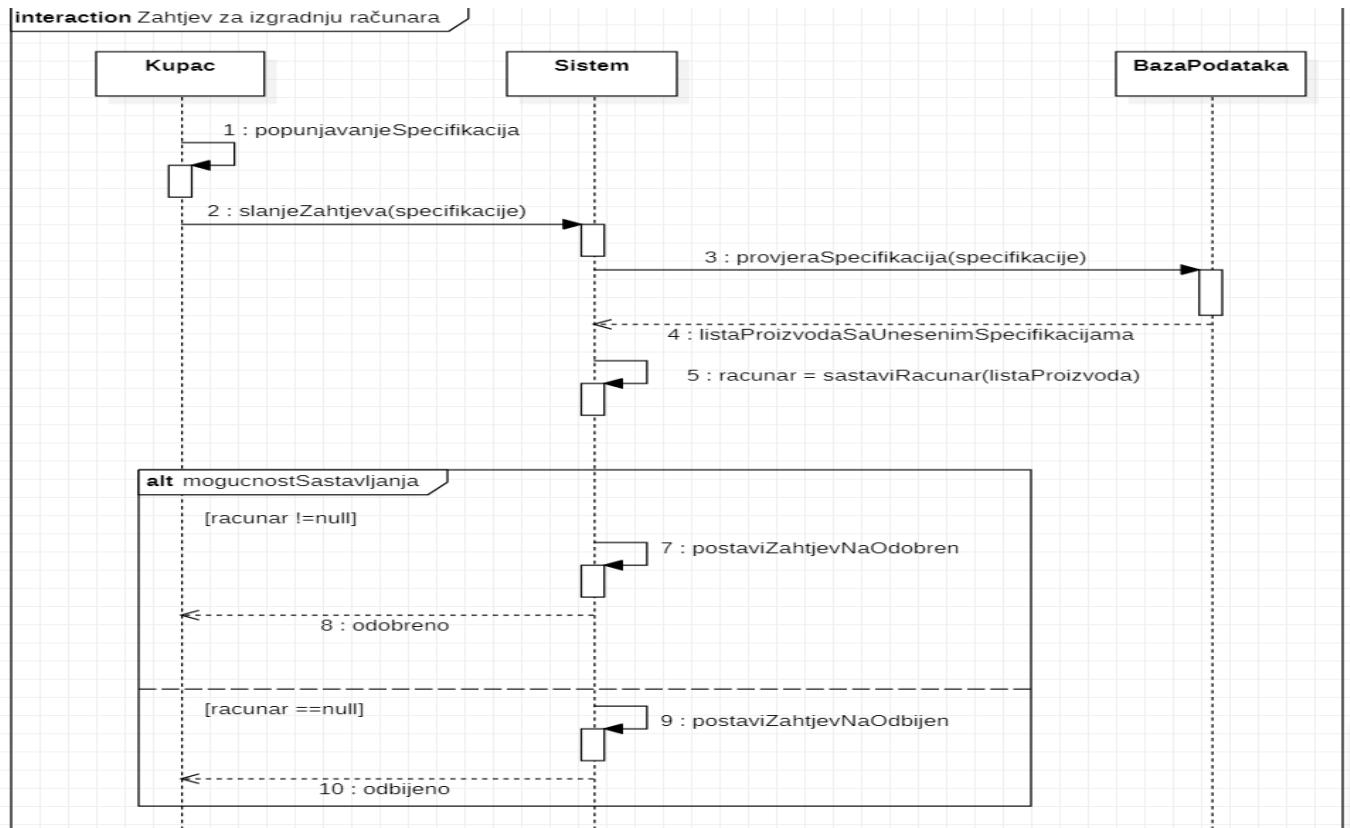
Za brisanje uposlenika:



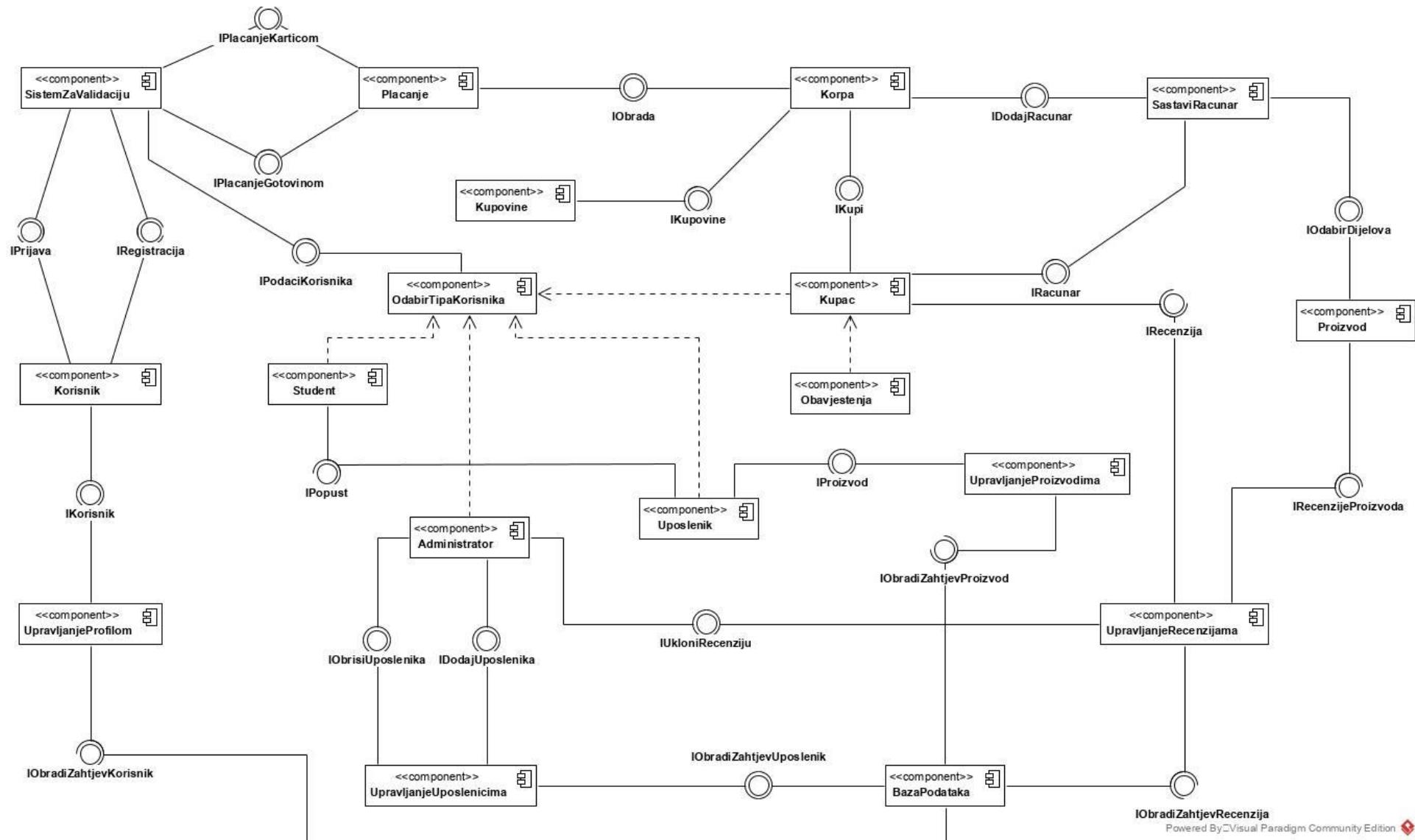
Za kreiranje korisničkog računa:



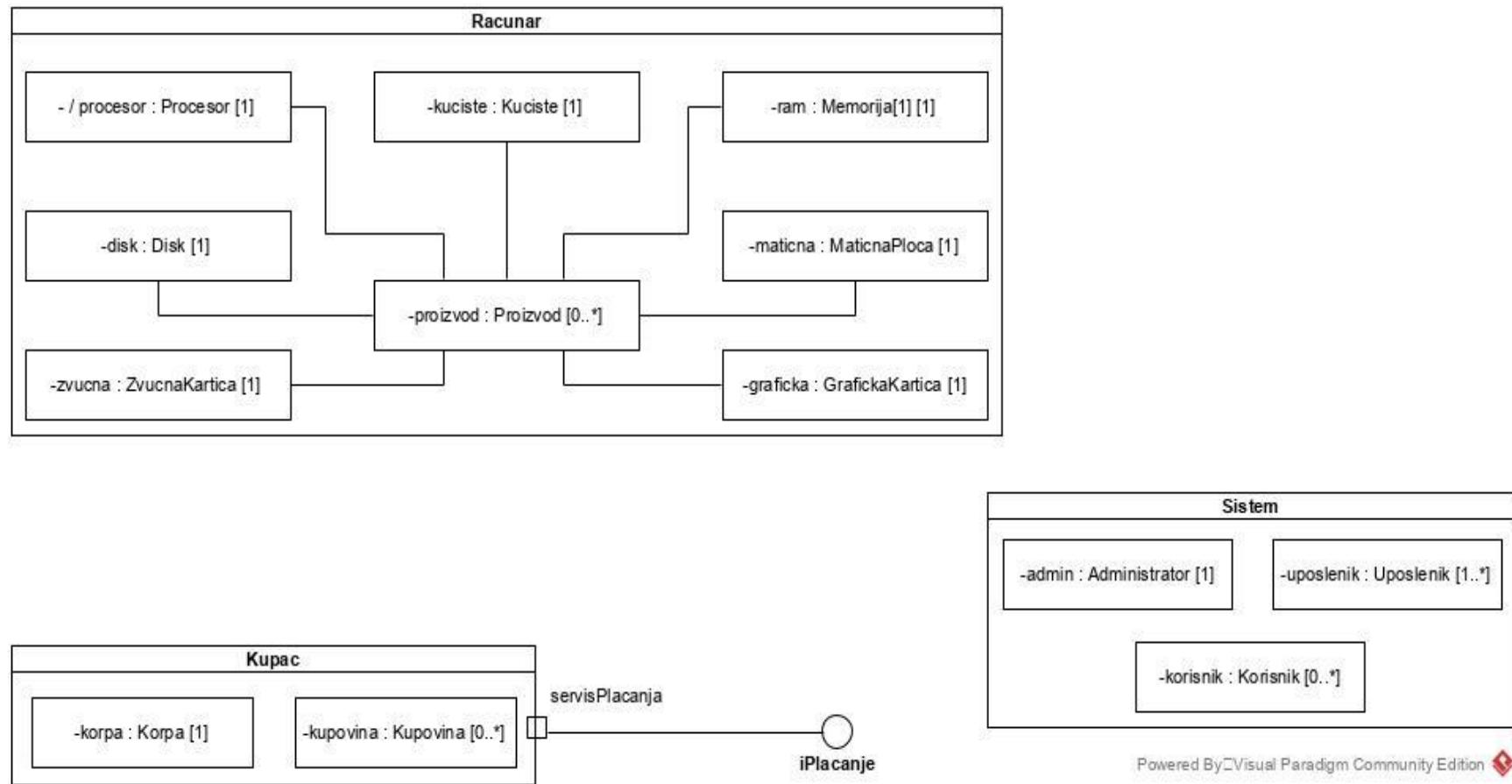
Za generisanje računara:



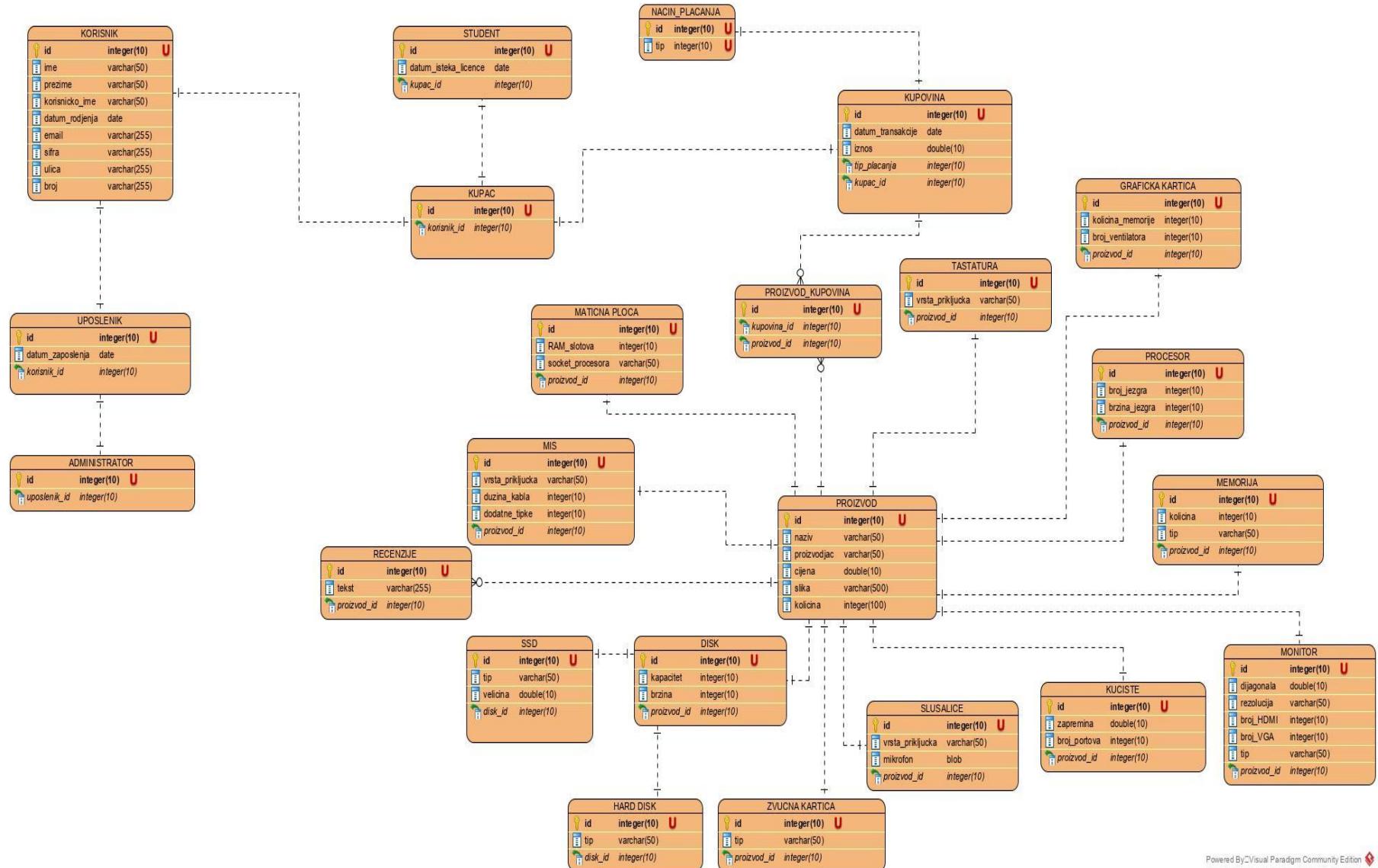
## Dijagram komponenti (Component diagram)



Dijagram složene strukture (Composite structure diagram)



## Entity relationship diagram (ERD)



### Web servis

Što se tiče naše aplikacije, nemamo pretjeranu potrebu za web servisom. Ukoliko bismo imali funkcionalnost da npr. korisnik naše aplikacije pri unosu naziva neke video igre i specifikacije svog računara saznaće da li njegov računar može pokrenuti tu igru. Mogli bismo koristiti API : <https://api.igdb.com/> . Koristili bismo request tipa <https://api-v3.igdb.com/games/> pri čemu bismo morali navesti api ključ koji bismo dobili pri registraciji na stranici. Dalje, postoje različiti parametri koje bismo mogli koristiti pri ovom request-u tipa „name“.