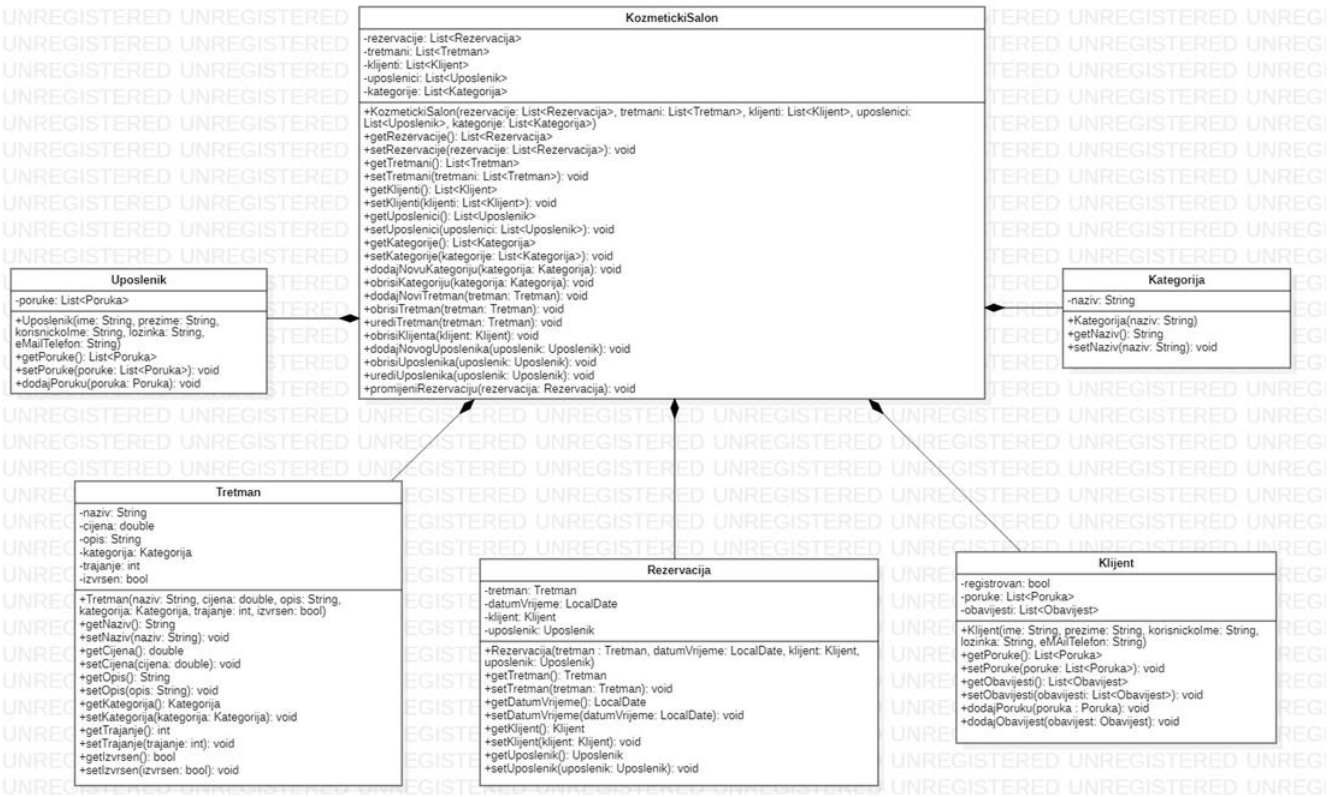
Strukturalni paterni

1. Facade patern

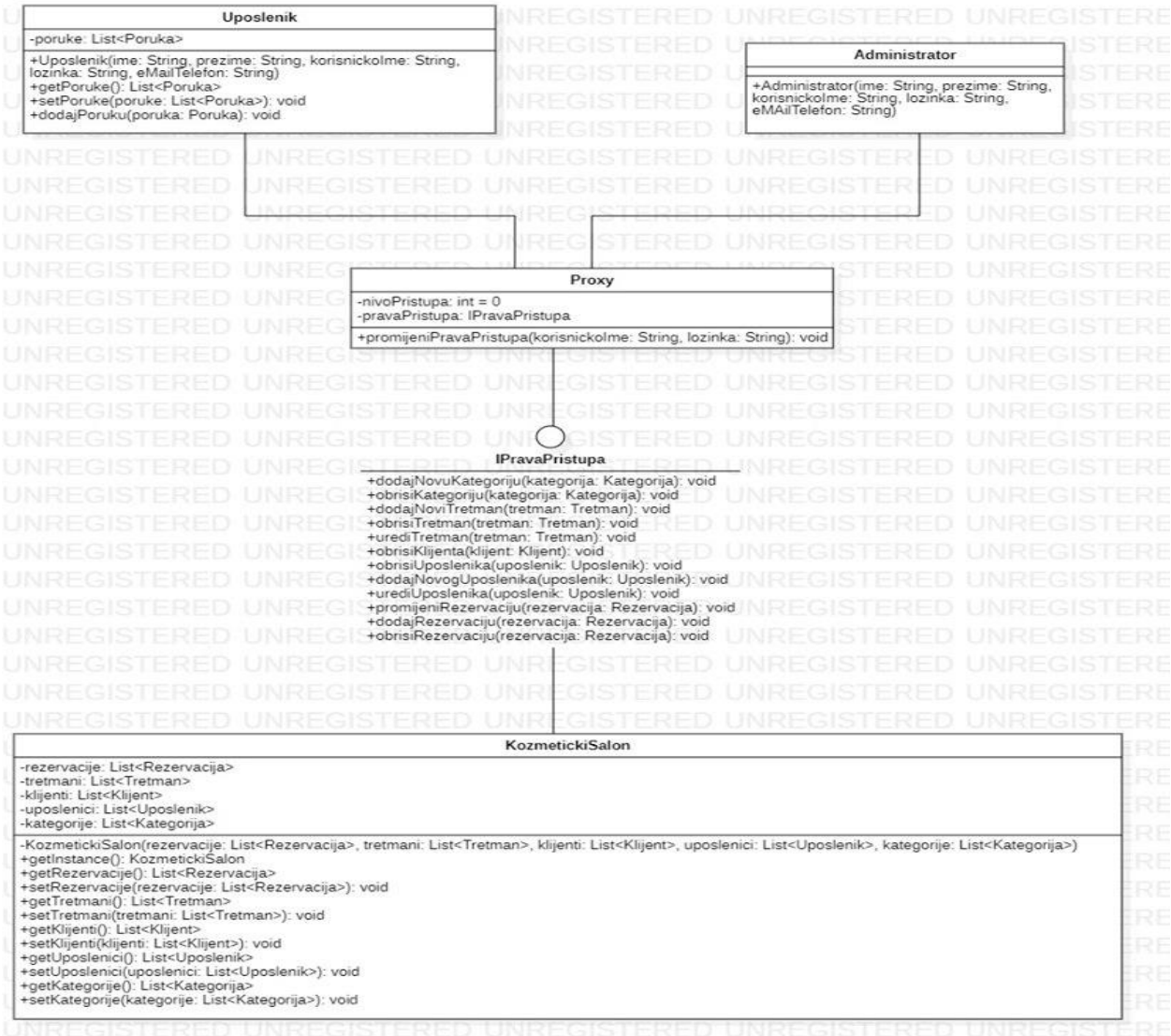
Fasadni patern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema.
Fasadni patern je već primjenjen na našem dijagramu klase. Korisnici aplikacije uopšte ne moraju poznavati sistem kako bi ga mogli korstiti tj. korisnici pozivaju metode koje žele da se izvrše, ali ne znaju koja se kompleksna struktura operacija krije u toku izvršavanja tih metoda (pozivanje drugih metoda i funkcija unutar metode koju je korisnik pozvao - npr. unutar metode void aktivirajHitanSlucaj(Klijent klijent, Uposlenik uposlenik), koja je implementirana u klasi KozmetickiSalon, poziva se metoda posaljiObavijestKlijentu(Obavijest obavijest)).
Klasa KozmetickiSalon ima veliki broj veza sa drugim klasama, a vanjski korisnik to ne vidi, te zbog toga predstavlja fasadu.



2. Proxy patern

Proxy patern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe.
Koristile smo ga da bi omogućile kontrolu pristupa bazi podataka. Potpuni pristup bazi ima administrator, ali određena prava ima i uposlenik. Prava koja ostvaruje uposlenik su: dodajRezervaciju(Rezervacija rezervacija), obrisiRezervaciju(Rezervacija rezervacija), promijeniRezervaciju(Rezervacija rezervacija).

Korisnička imena administratora i uposlenika razlikuju se po tome što su u korisničkom imenu administratora prva dva slova AD, a kod uposlenika UP. U metodi promijeniPravaPristupa se provjerava kojim slovima korisničko ime počinje. Ukoliko je riječ o administratoru atribut nivoPristupa postaje 1, a ukoliko je riječ o uposleniku nivoPristupa postaje 2.
Prilikom implementacije metoda interfejsa IPravaPristupa u klasi Proxy, vrši se provjera nivoaPristupa. Metode promijeniRezervaciju, dodajRezervaciju, obrisiRezervaciju se izvršavaju ukoliko je nivoPristupa 1 ili 2, u suprotnom se ne izvršava ništa. Sve ostale metode ovog interfejsa izrvšavat će se samo ukoliko je nivoPristupa 1.



3. Bridge patern

Bridge patern omogućuje nadogradnju modela klasa u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama.

U našem sistemu, Bridge patern mogle bi iskoristiti za slučaj da u sistemu čuvamo podatke o plati uposlenika te za slučaj da računamo plate za različite tipove uposlenika (svaka vrsta uposlenika predstavlja jednu klasu), pri čemu uposlenici imaju istu osnovicu plate, a različite dodatke u ovisnosti od vrste (isti princip računanja plate, iste osnovice, a različiti dodaci na platu).
Kako bi omogućili da sve napisano bude izvedivo potrebno je dodati novi interfejs IPlata, koji će imati metodu za računanje plate uposlenika, te novu klasu Bridge kojoj će administrator imati pristup. Sve klase koje predstavljaju vrste uposlenika implementirale bi interfejs IPlata. Klasa Bridge imala bi atribut IPlata i metodu dajPlatu(), koja bi vraćala plate uposlenika pozivajući metodu izracunajPlatu() nad atributom tipa IPlata. Metoda izracunajPlatu() računala bi dodatak na platu za svaku vrstu uposlenika.

4. Composite patern

Composite patern koristi se kada svi objekti imaju različite implementacije nekih metoda, ali im je svima potrebno pristupiti na isti način, te se na taj način pojednostavljuje njihova implementacija.

Patern nije iskorišten u našem sistemu. Kada bismo imali više vrsta uposlenika i čuvali podatke o njihovoj plati (koja je različita za svaku vrstu uposlenika) mogli bismo primijeniti ovaj patern. Ako bi salon, koji koristi našu aplikaciju, želio da na kraju svakog mjeseca zna iznos koji treba odvojiti za isplatu plata uposlenicima, onda bi uz pomoć ovog paterna mogli obračunati platu za svaku vrstu uposlenika pojedinačno te vratiti ukupan iznos (različit princip računanja plate, različite osnovice, različiti dodaci na platu).
Potrebno je dodati interfejs IUposlenik i unutar njega metodu dajPlatu() koja bi računala platu za svaku vrstu uposlenika (sve klase koje predstavljaju vrste uposlenika implementirale bi interfejs IUposlenik), te klasu SalonComposite koja bi implementirala interfejs IUposlenik i kao atribut imala listu uposlenika (tipa IUposlenik). Metoda dajPlatu() u klasi SalonComposite računala bi platu za sve vrste uposlenika iz liste i na ovaj način bi bilo omogućeno da se metode za računanje plate uvijek pozivaju na isti način.

5. 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.

Ovaj patern nismo iskristile u našem sistemu. Recimo da u sistemu imamo klasu Prikaz koja ima metodu void prikaziTretman(Tretman tretman). Ukoliko bismo dodale klasu VIPTretman, potrebno je omogućiti da se i ovi tretmani prikažu. To ćemo izvesti dodavanjem interfejsa IPrikaz koji će imati metodu prikaziVIPTretman(VIPTretman tretman), te klase VIPTretmanAdapter koja bi implementirala interfejs IPrikaz i klasu Prikaz. U metodi prikaziVIPTretman(VIPTretman tretman) vrši se konverzija VIPTretmana u Tretman i omogućava se njegov prikaz.

6. Flyweight patern

Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji svi u suštini predstavljaju jedan objekat.

Ovaj pater ne bi mogli iskoristiti u našem sistemu za slučaj da posmatramo aktere sistema, jer je svaki akter (administrator, uposlenik, klijent) jedinstven i ima jedinstvene osobine.

Recimo da administrator želi pristupiti pojedinačnim kategorijama. Pretpostavimo da ima više vrsta kategorija. Za unesenu kategoriju, ukoliko se ista nalazi u bazi treba vratiti postojeću, a za slučaj da ne postoji treba je dodati u bazu (spriječeno je dodavanje više istih kategorija).
Za slučaj da u sistemu imamo klase koje predstavljaju različite vrste kategorija tretmana (imaju atribut: nazivKategorije), implementirale bi interfejs IKategorija koji bi imao metodu dajKategoriju(). Postojala bi i klasa KatalogKategorija koja bi za atribut imala List<IKategorija> i bila bi povezana sa administratorom.

7. Decorator patern

Decorator patern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu).

Patern nismo iskoristile u našoj aplikaciji. Recimo da je tretmane u našem sistemu moguće nadograditi na način da oni postanu PosebniTretman, a dodatnim nadogradnjama učinimo da oni budu VIPTretmani.
Potrebno je dodati nove klase PosebniTretman i VIPTretman, te interfejs ITretman koji će biti implementiran u novododanim klasama. Klasa PosebniTretman ima atribut tipa Tretman, a VIPTretman ima atribut tipa ITretman. Interfejs ITretman imao bi metode dajTretman() i nadogradiTretman(int vrstaNadogradnje), koje bi u klasama PosebniTreman i VIPTretman omogućavale sve nadogradnje u ovisnosti od vrsteNadogradnje. Kako je u našem sistemu klasa Rezervacija povezana sa klasom Tretman, ona bi pored toga dodatno implementirala interfejs ITretman (što bi omogućilo da se pri rezervaciji, pored običnog tretmana, mogu rezervisati i poseban i vip tretman).