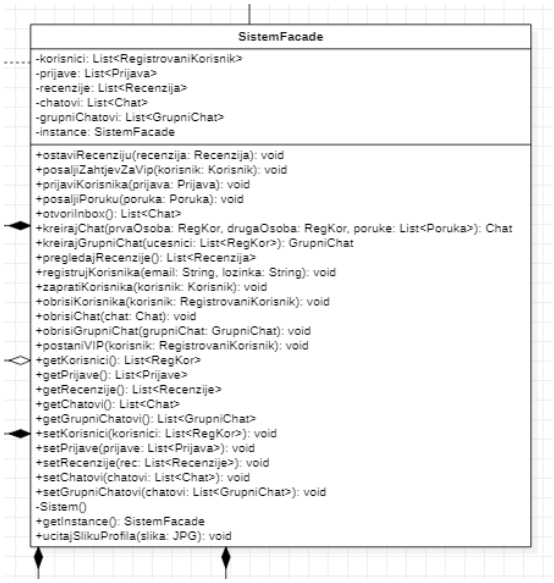
STRUKTURALNI PATERNI

1. Adapter

U našem programu imamo već kreiranu klasu *RegistrovaniKorisnik*, te ćemo kreirati novi interface *IDodatnoUcitavanjeSlika*, gdje bi se u budućnosti moglo dodati još različitih formata za učitavanje slika i koji je naslijedio adapter klasu. Poznato je da je PNG format u široj upotrebi na web-u jer se smatra da je „lossless compression file format“, tako da ćemo kreirati *UcitajSlikuAdapter* klasu koja će izvršiti adaptiranje formata učitane slike iz JPG u PNG format, a osnovna metoda za učitavanje slike profila nalazi se u kontejnerskoj klasi *Sistem*, sa metodom ucitajSlikuProfila(slika:JPG):void, koja se poziva u adapter klasi nakon koverzije.

2. *Facade*

U našem programu želimo omogućiti registrovanom korisniku što lakše kreiranje običnog razgovora ili razgovora sa više osoba, ukoliko je VIP član. Prethodno smo kreirali klasu *RegistrovaniKorisnik,* te klase *Sistem, Chat* i *GrupniChat.* Da bi smo „sakrili“ detalje o kreiranju običnog ili grupnog razgovora, proglasit ćemo našu klasu *Sistem,* fasada klasom. Zašto? Jer je to već klasa koja uveliko pojednostavljuje korištenje aplikacije registrovanom korisniku. Komunicira sa šest različitih klasa, sakriva kompleksnosti kreiranja chatova i grupnih chatova korisniku, gdje isti ne mora razmišljati šta se nalazi ispod površine određene mogućnosti, dovoljno je samo da pozove metode iz klase *SistemFacade*, sve ostale akcije klasa čini za njega. Odnosno, korisnik ne mora da zna sastav i internu implementaciju neke pogodnosti.



3. *Decorator*

Još jedan način kako bismo mogli unaprijediti iskustvo korištenja aplikacije korisniku, te olakšati mu modifikacije (rotacija, rezanje) slike profila je idući.

Možemo kreirati zasebnu klasu *Slika* koja će sadržavati atribute ime:String i slika:Bitmap. Potrebno je dodati interfejs *IslikaProfila*, te tri nove klase *SlikaPromjena, SlikaRezanje, SlikaRotacija.* U interfejs je potrebno implementirati metode edituj i dajSliku, a ostale tri klase će naslijediti interfejs.

Bitno je napomenuti da bi osnovna vrsta slike imala atribut tipa Slika, a ostale bi imale tipa ISlika, čime bi se osigurao tok akcija.

4. ***Bridge***

   Ovaj patern bismo mogli iskoristiti u našoj aplikaciji kada bi postojala neka opcija kupovine dodatnih pogodnosti u aplikaciji za registrovane korisnike, gdje bi se naknada za kupovinu različito računala za običnog korisnika i VIP korisnika.
   Tada bi bilo potrebno dodati novi interfejs *IdodatnePogodnosti,* koji će sadržavati definiciju metode za izračun naknade kupovine korisnicima. Također, bilo bi potrebno dodati klasu *Bridge*, koja će sadržavati apstrakciju i kojoj će korisnici jedino imati pristup, ukoliko žele pregledati cijenu pogodnosti na osnovu njihovog korisničkog računa. Te bilo bi potrebno dodati atribut koeficijent: double u, već postojeće klase, *RegistrovaniKorisnik* i *VIPKorisnik*, na osnovu kojeg bi se vršio izračun cijene.

5. ***Composite***

   Ovaj patern u našoj aplikaciji možemo implementirati na sljedeći način. Ukoliko želimo dopustiti adminu pregled svih korisnika u aplikaciji podijeljenih na obične i VIP, s obzirom da se tako dijele, trebali bismo u klasu *Sistem* dodati medotu dajSveKorisnike:List<RegistrovaniKorisnik> *(primjer vraćenog rezultata: „Registrovani korisnici: Ajla Ajlić, Zejd Zejdović,... VIP Korisnici: Adna Adnić,...“)*, koju će admin moći pozvati i koja će vratiti sve korisnike registrovane u aplikacijiali sortirane po nivou računa. Zatim, kreirati interfejs *IKorisnici* u koji ćemo implementirati definiciju metode razvrstajKorisnike za sortiranje korisnika po vrsti računa. Sve tri klase, odnosno *Sistem, RegistrovaniKorisnik* i *VIPKorisnik,* će naslijediti interfejs kako bi se kreirala hijerarhija objekata. Klase *RegistrovaniKorisnik i VIPKorisnik* će dodatno zadržavati nove atribute nivoRačuna:int (koji može biti 1 za običnog korisnika i 2 za VIP korisnika) i koji će koristiti pri sortiranju.

6. ***Proxy***

   Ovaj patern u našoj aplikaciji možemo implementirati na sljedeći način. Ukoliko korisnik ili admin aplikacije žele pretražiti korisnike, na osnovu bilo kojeg filtera, pri čemu se korisniku pri pretrazi prikazuju samo osnovne informacije drugih korisnika koje su javne na njihovom profilu, dok se adminu prikazuju i javne informacije dostupne na korisničkom računu i dodatna opcija; obriši korisnika. Potrebno je na već definirani interfejs *IPretragaKorisnika* u prethodnom dijagramu, unaprijediti sa dvije različite metode; pretragaKorisnika i pretragaKorisnikaAdmin, koje su definisane kako je gore već navedeno. Zatim, definisati novu klasu *Proxy* koja će sadržavati atribute nivoPristupa:int (za određivanje da li pretragu zahtijeva admin ili reg. korisnik), te listu korisnika koja

treba biti zaštićena korisnici:List<RegistrovaniKorisnik>. Ova klasa će naslijediti interfejs i njegove metode.

7. ***Flyweight***

Ovaj patern u našoj aplikaciji možemo implementirati na sljedeći način. S obzirom da želimo omogućiti korisniku pri slanju poruka u chat, unos emotikona, slova ili fotografija, potrebno je da definišemo nove tri klase *Emotikon* (izraz:PNG), *Slovo* (velicina:int, font:String, broj:int), *Fotografija (slika: PNG)* sa njihovim atributima. Zatim definirati interfejs *IDajZnak* koji će sadržavati definiciju metode dajVrstuZnaka za razlikovanje traženih znakova. Potrebno je definisati i novu klasu *Unos* sa metodom dajZnak koju će korisnik pozvati kada želi unijeti slovo ili emotikon. Te sve četiri klase, *Emotikon, Slovo, Fotografija* i *Unos*, povezati sa definisanim interfejsom.

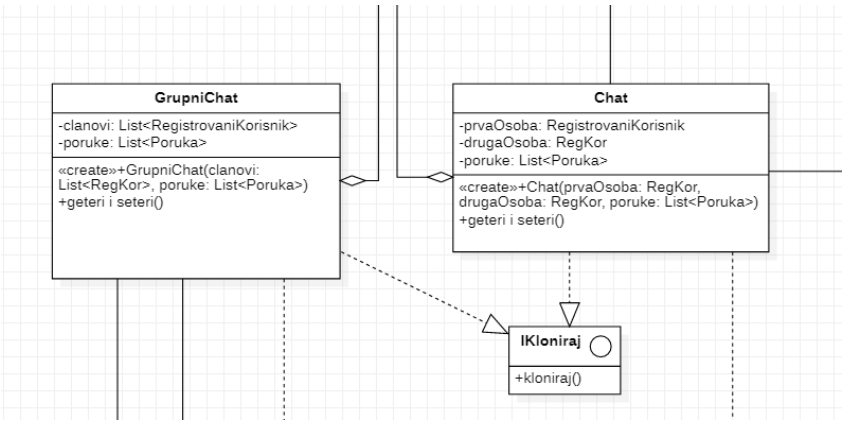
KREACIJSKI PATERNI

1. ***Singleton***

S obzirom da će postojati samo jedan sistem u kojem će biti sadržane sve kolekcije drugih objekata i koji će upravljati njima, potrebno je osigurati jedinstveno instanciranje klase Sistem pomoću Singleton paterna. Singleton patern će biti implementiran u klasu Sistem, a sadržavat će privatnu static varijablu koja čuva jednu/jedinstvenu instancu klase, javnu static metodu (`getInstance`) preko koje se pristupa Singleton klasi i privatni static objekat koji se interno instancira korištenjem privatnog konstruktora.

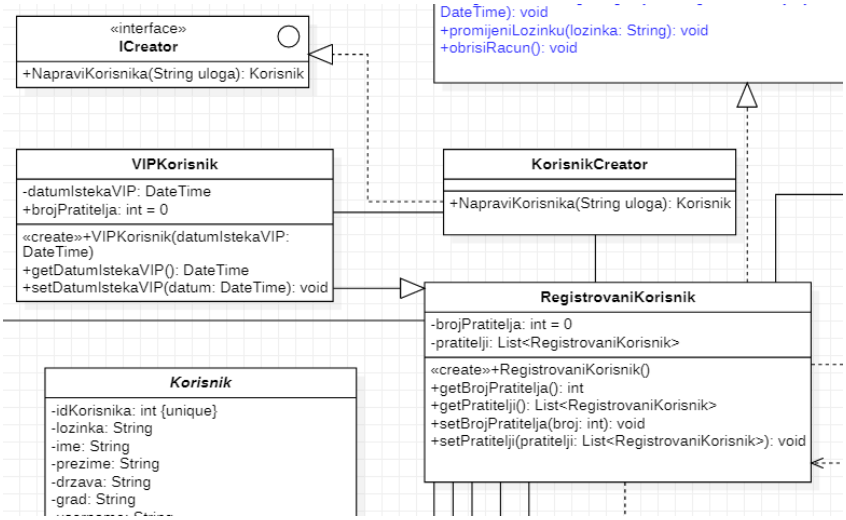
2. ***Prototype***

Da bismo izbjegli konstantno uzimanje istih informacija iz baze podataka napraviti ćemo interfejs *IKloniraj* sa metodom *kloniraj()*. Ovaj interfejs će implementirati *Chat* i *GrupniChat* jer su to klase koje bi inače najčešće pristupale bazi podataka. Ove dvije klase će implementirati metodu kloniraj zbog potencijalno velikog broja poruka koje mogu biti između dva korisnika. Ova funkcionalnost će biti iskorištena unutar Sistem klase pri radu sa Chat i GrupniChat klasama



3. *Factory Method*

Za realizaciju ovog paterna ćemo imati interfejs ICreator sa metodom NapraviKorisnika(String uloga) koja vraća Korisnika. Ovaj interfejs će implementirati klasa KorisnikCreator pomoću koje ćemo praviti nove objekte Korisnik tipa u našem programu. Glavni razlog za implementiranje ovog paterna u našu aplikaciju je olakšano dodavanje novih vrsta korisnika(npr botovi). Također je ovaj patern koristan radi skrivanja logike pravljenja objekata tipa Korisnik od osobe koja će koristiti aplikaciju.



4. *Abstract Factory*

Ovaj patern nećemo koristiti u našoj aplikaciji, barem ne u prvoj verziji. U slučaju da se nekad odlučimo na nadogradnju chatova i grupnih chatova, mogli bismo iskoristiti ovaj patern odnosno zahvaljujući njemu, korisnik bi mogao birati vrste chatova i grupnih chatova koji će se nalaziti u njegovom inboxu (boja poruka, font poruka, izgled emojisa i slično). Za izvedbu ovog paterna, dodali bismo određeni broj factory klasa koje bi predstavljale koja vrsta chatova će biti korištena u

inboxu jednog korisnika. Npr jedna fabrika bi mogla biti MoneyHeistFactory po uzoru na trenutno aktuelnu seriju i u tom slučaju bi chatovi bili u tonovima crvene boje sa stikerima likova iz te serije i slično. Naravno, svaka fabrika bi sadržavala metode koje postavljaju boje, stikere, fontove na odgovarajući način u skladu sa željenom temom. Po volji bi se mogao dodavati veliki broj ovakvih fabrika, a sve te fabrike bi nasljeđivale apstraktnu klasu Factory.

5. ***Builder***

   Ovaj patern bismo mogli iskoristiti kod kreiranja objekata tipa Korisnik jer klasa Korisnik ima veći broj atributa koji bi se mogli odvojeno definisati ali i dalje nije prevelik broj atributa tako da nije kompleksno ni postojeće rješenje što bi se moglo promijeniti ukoliko se klasa Korisnik nadogradi sa mnogo više informacija o korisniku i u tom slučaju bi ovaj patern bio itekako od koristi. U slučaju implementacije Builder paterna, postojao bi interfejs IBuilder sa različitim dijelovima kreiranja korisnika, npr. metode dodajImeIPrezime(String ime, String prezime), dodajDrzavuIGrad(String drzava, String grad), dodajSliku(String url), dodajBiografiju(String biografija), dodajDatumRodjenja(DateTime datum), dodajObrazovanje(String obrazovanje), dodajPosao(String posao), dodajVisinu(Double visina) itd. Također postojali bi i različiti builderi - BuilderSingle bi pozvao samo osnovne metode, dok bi BuilderRazvedeni pozvao i dodatne metode npr. dodajGodinuRazvoda(DateTime datum) i dodajRazlogRazvoda(String razlog), također bi se mogao dodati i BuilderUdovac koji bi npr. pozvao metodu dodajBrojGodinaOdSmrtiPartnera(Integer brojGodina).