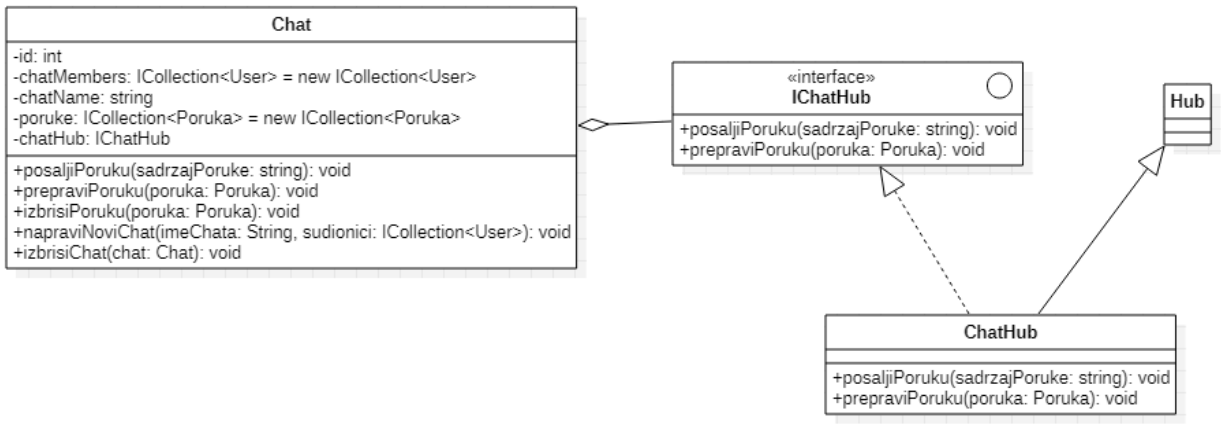
STRUKTURALNI PATERNI

1. ***ADAPTER PATERN:***

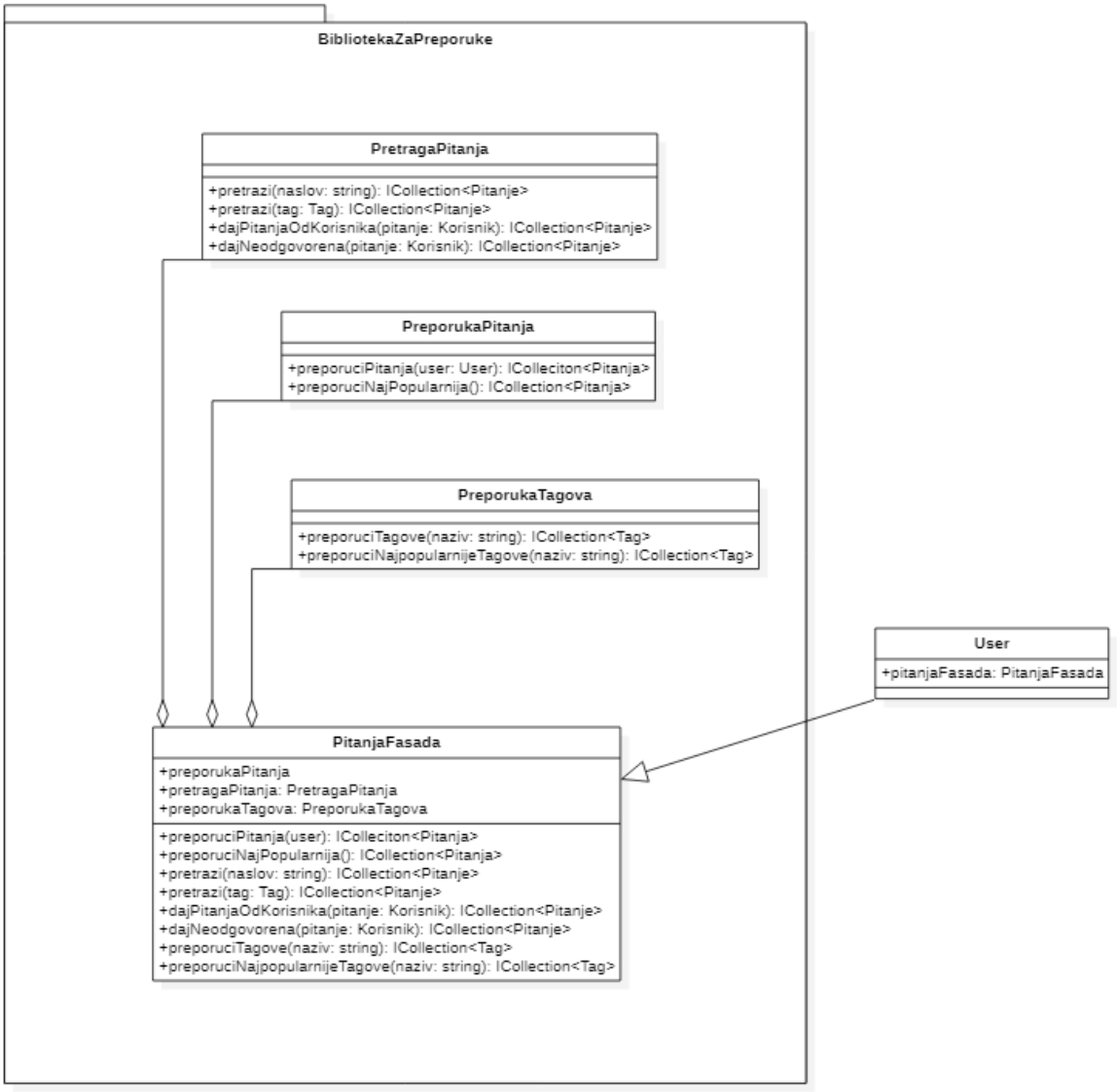
Adapter patern se koristi kada je potrebno funkcionalnost jedne, već postojeće, klase adaptirati za korištenje nekog interfejsa, bez izmjene same već postojeće klase.
Ovaj patern će biti iskorišten u implementaciji chata. Kako za implementaciju chata, kako smo je mi zamislili, je potrebno da u pravom vremenu se izmjenjuje sadržaj ekrana u zavisnosti od poslanih i primljenih poruka. U tu svrhu je potrebno koristiti SignalR biblioteku koja sadrži klasu Hub koja izvršava tražene funkcionalnosti. Ali kako je javni interfejs te klase komplikovan za korištenje najbolje bi bilo napraviti neku adapter klasu koja na osnovu predefinisanopg interfejsa omogućava željene funkcionalnosti.



Kao što vidimo imamo klasu ChatHub koja adaptira metode iz Hub klase na način da zadovoljavaju interfejs IChatHub kojeg dalje korisnik koristi iz Chat klase

2. ***FACADE PATERN:***

Facade patern služi za pojednostavljenje korištenja neke biblioteke ili skupine kompleksnih klasa. Tačnije facade patern daje interfejs preko koga je omogućeno korištenje više različitih klasa koje imaju sličnu upotrebu. Korisnik dobija interfejs koji je mnogo lakši i intuitivniji za korištenje od skupine pojedinačnih klasa.
U našem projektu smo odlučili iskoristiti ovaj patern u svrhu pojednostavljenja rada sa sistemima za preporuke, bilo pitanja ili tagova.



Kako je korisniku često potrebno istovremeno generisanje više vrsta preporuka, bilo to tokom pretrage ili tokom kreiranja pitanja, bilo bi jako nezgrapno uskladiti rad više neuvezanih klasa koje zajedno daju željeni rezultat, zato je povoljno kreirati jednu fasadnu klasu koja će se brinuti za sve funkcionalnosti vezane za preporuke nama potrebnih objekata.

3. ***DECORATOR PATERN:***

Dekorator patern je patern koji omogućava dodavanje novih funkcionalnosti već postojećim klasama na način da ih postavljamo u poseban omotač koji još uvijek implementira orginalne metode omotanih klasa i još mu daje neke nove funkcionalnosti. Ovaj patern bi mogao biti implementiran na dva mjesta u našoj aplikaciji. Prvi slučaj je klasa Notifikacija. U zavisnosti na koji uređaj ili uređaje je potrebno poslati notifikaciju

moguće je dodati omotač na baznu klasu Notifikacija i dodati dekoratorske klase koje bi odgovarale tipu notifikacije, npr. Dekorator za mail notifikacije, za mobilne notifikacije, za notifikacije na samoj aplikaciji itd. Također ovaj patern bi se mogao iskoristiti kod klasa vezanih za korisnika. To bi se moglo izvršiti na sljedeći način. Prvo bi napravili jedan osnovni interfejs koji definiše sta svaki neregistrovani korisnik može raditi u aplikaciji, kao pretraga pitanja i kreiranje i logovanje u račun. Potom na implementaciju tog interfejsa možemo dodati dekorator za korisnike koji su registrovani u aplikaciji te oni bi imali dodatnu funkcionalnost naspram neregistrovanih korisnika, kao mogućnost postavljanja pitanja i odgovora, ostavljanja rejtinga na pitanja i odgovore, funkcionalnost chata itd. Te još bi mogli dodati jedan dekorator i na tu klasu i proširiti je funkcionalnostima korisnika koji je također i administrator na aplikaciji, ali koji također još uvijek održava sve funkcionalnosti koje imaju dvije prethodne klase.

4. BRIDGE PATERN:

Bridge patern se koristi u slučajevima kada je potrebno razdvojiti apstrakcije od implementacije neke klase. Obično se koristi kada je potrebno kreirati neki veći sistem, tada ga je povoljno dizajnirati na način da pojedini dijelovi sistema ne zavise od konkretne implementacije nego samo od apstrakcija, što i opisuje ovaj patern.
Ovaj patern u našem programu bi mogli implementirati u dijelu zaduženom za rad sa reakcijama, već posjedujemo interfejs IRejting kojeg implementira klasa Rejting. Ukoliko bi napravili dvije zasebne implementacije interfejsa IRejting, jednu za rad sa rejtingom pitanja RejtingPitanja i jednu za rad sa rejtingom odgovora RejtingOdgovora, a potom jednu klasu koja bi koristila interfejs IRejting, npr. klasa Rejting, koju bi dalje koristile sve ostale klase kojima je potreban neki način dobivnja rejtinga. Samim ovim postupkom bi implementirali bridge patern, u ovom slučaju bi nam implementacije bile RejtingPitanja i RejtingOdgovora, bridge bi bio interfejs IRejting, a abstrakcija bi bila klasa Rejting.

5. PROXY PATERN:

Proxy patern se koristi na način da se za pristup jednog objekta koristi drugi objekat koji je u potpunosti zadužen za preradu svih zahtjeva koji bi trebali ići ka pravom objektu. Ovaj patern je povoljno koristiti kada je držanje pravog objekta nije praktično ili kada je pristup pravom objektu ograničen. Proxy bi mogli implementirati ukoliko bi uveli neki oblik statistike za korisnike. Dodavajući klasu Statistika koja bi čuvala statistiku svih korisnika, npr. broj postavljenih pitanja, broj poslanih poruka, broj ostavljenih reakcija, mjesta iz kojih najčešće postavljaju pitanja itd., ali kako su te informacije privatne te ih ne bi trebali vidjeti svi korisnici nego samo određeni, npr. Svaki korisnik može vidjeti svoju statistiku i globalnu statistiku dok admin može vidjeti sve statistike. U tu svhu dodajemo StatistikaProxy koji bi funkcionirako kao proxy za klasu Statistika, a obe klase bi bile realizacije interfejsa IStatistika kojeg bi koristile druge klase.

6. ***COMPOSITE PATERN:***

Composite patern služi u slučaju kada je potrebno da se program ponaša isto bilo da radimo sa više ili samo jednom istancom neke klase. Ovaj patern ima smisla implementirati samo u slučajevima kada se model sistema može prikazati u obliku stabla. Ovaj patern bi se mogao iskoristiti u implementaciji chata. Kao composite ove strukture bi bila klasa Chat dok bi leaf-ovi bile klase ChatUser i klasa Message. Najbolji primjer načina rada ovakvog sistema bi bila funkcionalnost brisanja. Pošto brisanje korisnika chata podrazumjeva njegovo izbacivanje iz chata, a brisanje poruke podrazumjeva samo brisanje poruke, ali ako želimo obrisati čitav chat potrebno je obrisati svaku poruku kao i izbaciti svakog korisnika, samim tim nemamo razlike u funkcionalnosti izmedju leafova i composita ovakvog sistema

7. ***FLYWEIGHT PATERN:***

Flyweight patern se koristi kada je potrebno smanjiti količinu potrebne memorije za rad aplikacije. Ideja paterna je da se dijelovi neke klase koji su isti za sve instance klase izvuku u zasebne objekte koji su zajednički za sve te instance, samim tim smanjujemo potrebu za memorijom kod te klase.
Flyweight patern bi također mogli iskoristiti u dijelu programa za regulisanje chatova. Kako će svaka poruka imati u sebi atribut koji će sadržavati sliku korisnika, bilo bi dobro da ne kreiramo više instanci tog atributa nego da unutar chata postoji samo jedna zajdnička slika koju će sve poruke koristiti. Također ovo bi radilo u grupnom chatu gdje samo umjesto da bude zajednička jedna slika, bilo bi više slika koje bi se dodjeljivale porukama u zavisnosti od korisnika koji ju je poslao. Implementacija bi bila sljedeća, kontekst bi bila klasa Chat koja ima više instanci klase Message. Message ima u sebi atribut Picture koji sadrži sliku korisnika koji je poslao poruku. Potom bi postojao MessageFactory koji i sadržavao u jednoj mapi sve slike korisnika koji se nalaze u trenutnom chatu te na osnovu njihovog id-a bi dodjeljivao sliku odgovarajućoj poruci.