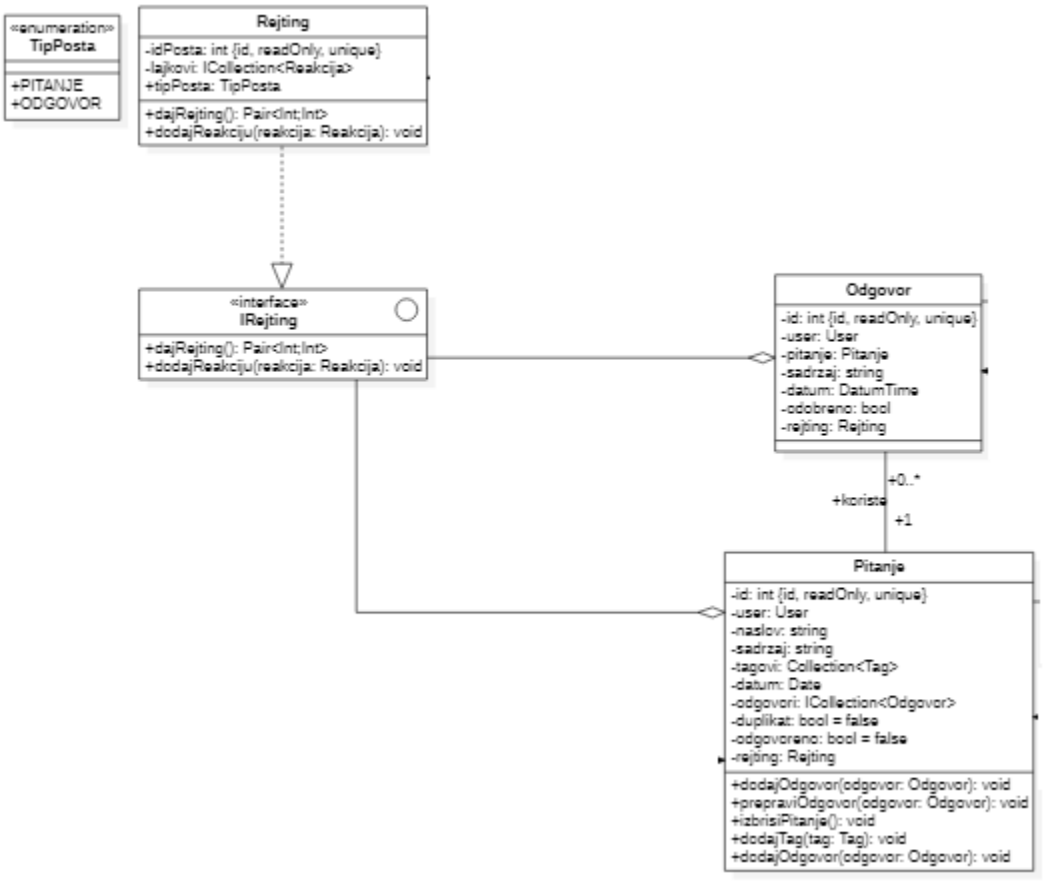
PATERNI PONAŠANJA

1. ***STRATEGY PATERN:***

Strategy pattern se zasniva na ideji da možemo imati familiju algoritama koji su međusobno zamjenjivi, te u zavisnosti od potrebe sistema možemo lagano izmjenuti jedan algoritam sa drugim.
Ovaj pattern smo iskoristili kod dijela programa zaduženog za računanje rejtinga nekog posta. Nama kontekst klasu predstavljaju klase Pitanje i Odgovor, strategiju nam predstavlja interfejs IRejting, a implementacija je Rejting. Ovdje je povoljno iskoristiti ovaj patern iz razloga ako budemo htjeli proširiti aplikaciju na način da imamo na različite načine racunanja rejtinga, npr ako bi htjeli da rejting nije samo lajk i dislajk nego i dodatno od nula do 5 i slično.



2. ***STATE PATERN:***

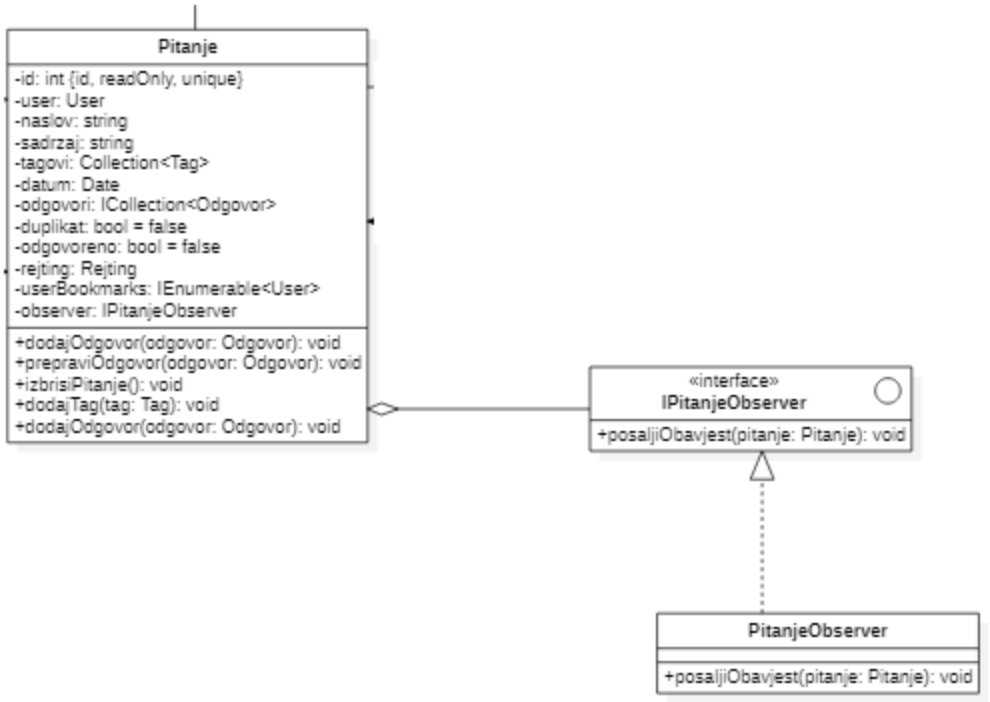
State pattern se koristi kada imamo objekat koji se ponaša različito u zavisnosti od njegovog trenutnog stanja. Ovaj patern je najkorisniji ukoliko objekat ima veliki broj stanja.
Ovaj patern bi se mogao iskoristiti u dijelu programa zasluženog za prikaz odgovora na pitanje. Naime, kada je pitanje u stanju neodgovorenog način na koji se prikazuju odgovorio bi bio da odgovori koji imaju najviše lajkova budu pri vrhu, a kada pitanje pređe u stanje odgovorenog, samim tim ako je pitanje odgovoreno mora postojati barem jedan prihvaćen odgovor, sada bi se prikaz odgovora radio na način da se prihvaćeni odgovori, u zavisnosti od vremena postavljanja, stavljaju pri vrhu dok ostali opet prema broju reakcija.

3. ***TEMPLATE PATERN:***

Template patern se koristi ukoliko imamo više algoritama koji rade slične stvari i koji se međusobno razlikuju bilo po redoslijedu obavljanja koraka ili po broju. Ovaj patern govori da se svi koraci koji se ponavljaju između tih algoritama izdvoje u jednu apstraktnu klasu koja sadrži zajedničke korake u tim algoritmima, te pojedinačni algoritmi mogu proizvoljno implementirati metode te klase prema vlastitim potrebama.
Trenutno nemamo niti jedan algoritam koji dijeli određeni broj koraka sa nekim drugim algoritmom tako da ga je prilično teško implementirati u našoj aplikaciji.

4. ***OBSERVER PATERN:***

Observer patern se koristi kada vrijedi da promjena stanja jednog objekta povlači za sobom da se trebaju promenuti i drugi objekti, a broj tih drugih objekata je u početku nepoznat.
Ovaj patern čemo iskoristiti u okviru sistema za slanje notifikacija korisnicima, tačnije u slučaju postavljanja odgovora na pitanje. Kako će sem korisnika i drugi korisnici imati pravo spasiti neka pitanja kao bookmarks, te prilikom dodavanja novog odgovora na pitanje će se slati notifikacija svim korisnicima koji imaju to pitanje u bookmarks. Subjekt klasa u našem slučaju bi bila kalsa Pitanje, imali bi interfejs IPitanjeObserver koji ima metodu Update koja će se pozvati prilikom svakog dodavanja novog pitanja, dok korektna implementacija ovog interfejsa bila PitanjeObserver.



5. ***ITERATOR PATERN:***

Iterator patern se koristi kada je potrebno napraviti neki način prolaska kroz neku kolekciju objekata s tim da je u potpunosti nepoznata građa takve kolekcije.
Iterator patern bi mogli koristiti u kontekstu pitanja. Ako bi smo napravili poseban konternjer za skup odgovora na nekom pitanju koji radi po principu heap-a čiji se elementi redaju na osnovu broja reakcija i prihvačenosti odgovora. Potom ako bi dodali iterator u tu kolekciju koji bi kretajući samo radio breadth-first search.

6. ***MEMENTO PATERN:***

Memento patern omogučava da se vrši stašavanje i povratak trenutnog stanja nekog objekta bez da pokažemo način na koji se taj objekat implementira.
Ovaj patern bi se mogao iskoristiti kod postavljanje pitanja, tj.ukoliko bi omogučili da prilikom postavljanja pitanja korisnik izađe iz datog obrasca da se promjene koje je već napisao ne izbrišu nego spase pa kada ponovo bude ulazio u taj obrazac da budu unešeni ti podaci.

7. ***VISITOR PATERN:***

Visitor patern se koristi kada želimo odvojiti algoritam od objekata na kojima se on vrši.
Ovaj patern bi iskoristili u algoritmu za preporuku, te tada bi bilo svejedno da li se algoritam koristi nad tagovima ili pitanjima.