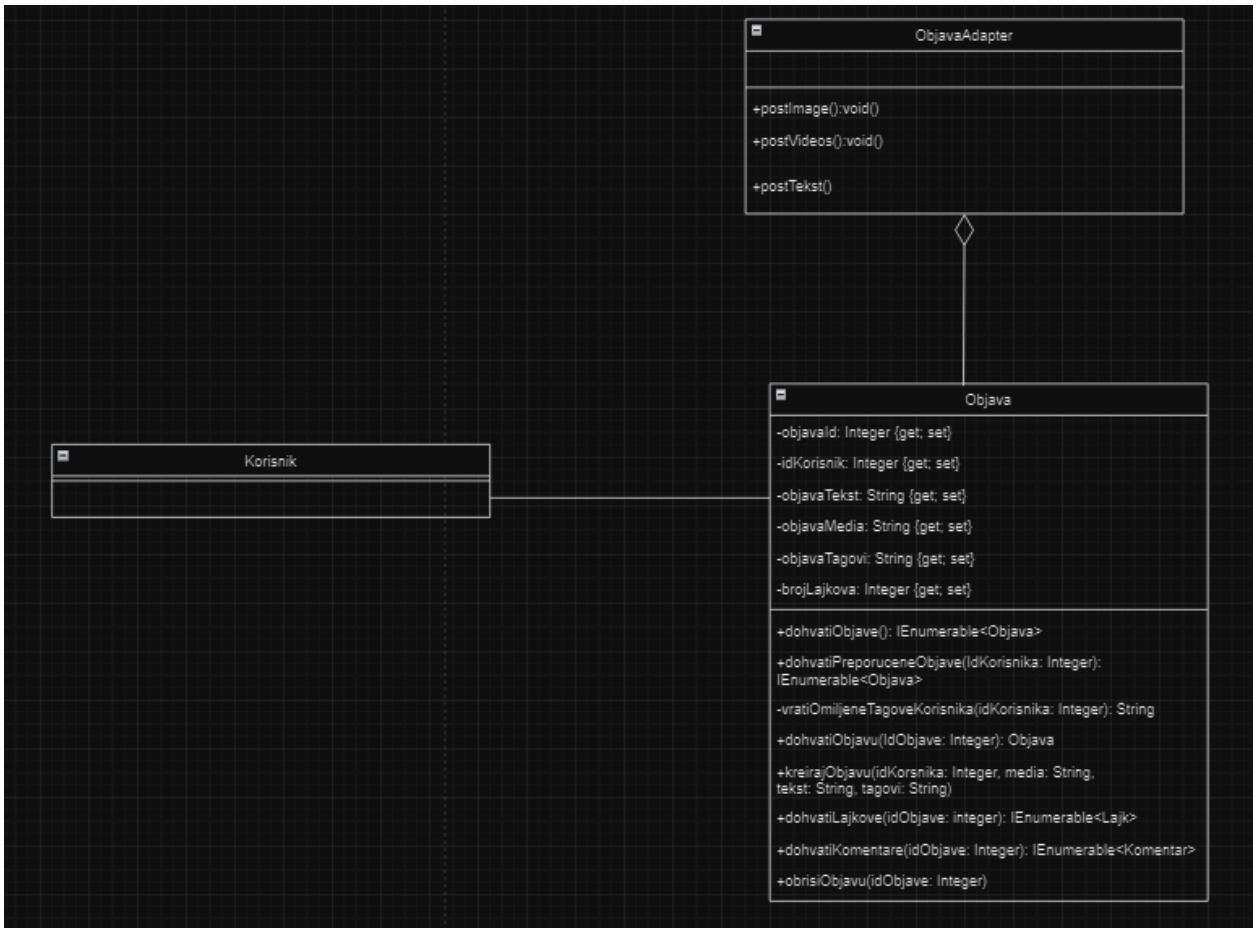
STRUKTURALNI PATERNI

Adapter patern

Osnovna namjena Adapter pattern-a je da omogući širu upotrebu već postojećih klasa. Kada je potreban drugačiji interfejs postojeće klase, a ne želimo da mijenjamo postojeću klasu koristimo Adapter pattern. Tada se kreira nova Adapter klasa koja se koristi kao posrednik između originalne klase i interfejsa. Primjena u sistemu:

U našem sistemu Adapter patern bi mogli iskoristiti za objave. U našem sistemu trenutni tip atributa za tip objavu je string. Taj tip bi mogli zamijeniti uz pomoć metoda konverzija Adapter klase i omogućiti prihvatanje raznih tipova formata objave, bila to slika, tekst ili videozapis. Adapter bi omogućio da se prilikom objavljivanja slike omogući objava iz različitih formata(jpg, pdf) i drugo.



Bridge patern

Osnovna namjena Bridge paterna je da omogući odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije. Moguće je implementirati i sistem za razmjenu poruka primjenom Bridge paterna.

Ovaj patern bismo mogli primjeniti na tipove različitog slanja notifikacija. Naprimjer mogli bismo implementirati da korisnik može primiti notifikaciju putem aplikacije kao i putem emaila vezano o događajima, notifikacijama i drugo.

Composite patern

Composite patern opisuje grupu objekata koji se tretiraju na isti način kao pojedinačna instanca istog tipa objekta. Namjera kompozita je da "komponira" objekte u strukture stabla koje predstavljaju hijerarhiju dio-cjelina.

U našem sistemu Composite patern bismo mogli iskoristit tako što bi mogli uvesti još jednu vrstu korisnika – gost, koja će imati sličnu funkcionalnost kao korisnik, koja bi mogla samo pregledati preporučeni sadržaj.

Decorator patern

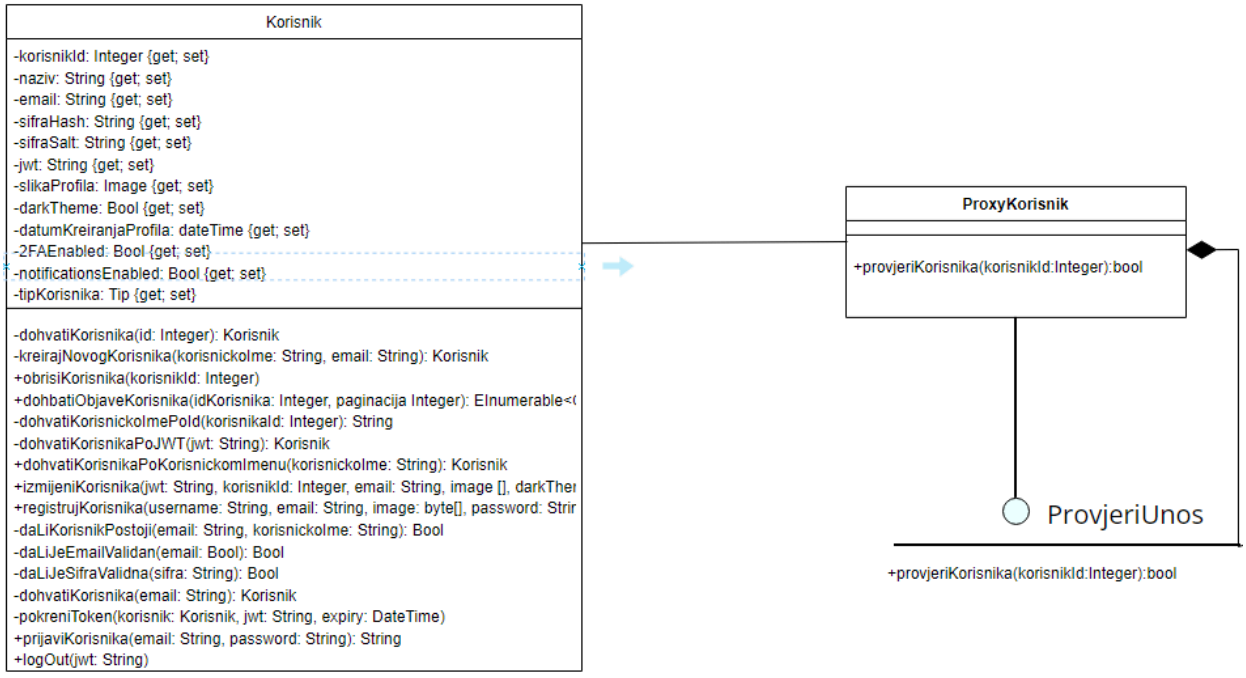
Decorator pattern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta. Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti različito dekoriranje objekata (tj. dodavanje različitih detalja), te se na taj način pojednostavljuje i rukovanje objektima klijentima, i samo implementiranje modela objekata. U suštini, decorator pattern se koristi u slučajevima kada želimo dodijeliti dodatna ponašanja objektu tokom izvođenja programa bez da mijenjamo kod koji je na neki način u interakciji sa datim objektom.

U našem sistemu decorator patern primjenili smo kao funkcionalnost koja uključuje promjenu postavki i prikaz profila korisnika. Decorator pattern može se koristiti za dodavanje ili promjenu funkcionalnosti na objektima profila korisnika, poput dodavanja mogućnosti promjene profila kao korisničkog imena, profilne slike kao i uređivanje postavki na korisničkom računu.

Proxy pattern

Svrha **Proxy patern-a** je da omogući pristup i kontrolu pristupa stvarnim objektima. Proxy je obično mali javni surogat objekat koji predstavlja kompleksni objekat čija aktivizacija se postiže na osnovu postavljenih pravila.

U našoj aplikaciji ovaj patern ćemo iskoristiti tako što ćemo zabraniti zlonamjerne upotrebe. Recimo ukoliko neki korisnik koji nije registrovan kao administrator ili zaposlenik pokuša da koristi privilegije koje ima administrator, ta akcija će biti zabranjena, te će administrator biti obavješten o tome.



Pored toga, ovaj patern se može se koristiti za optimizaciju prikaza profila korisnika. Prilikom učitavanja korisničog profila, može se prethodno učitati informacije o korisniku poput imena i objava na profilu, dok se ne smije moći uređivati jer to nije profil koji pripada trenutačnom korisniku sesije.

Također, u našem sistemu, administrator često pristupa i mijenja podatke o komentarima I objavama koji su prijavljeni, kao I brisanje korisničkog računa. Kako je autentičnost tih podataka jako bitna, trebamo sa sigurnošću znati da sve izmjene vrši administrator. U tome nam može pomoći ovaj patern.

Façade patern

Fasadni pattern služi kako bi se korisnicima pojednostavilo korištenje kompleksnih sistema, odnosno koristimo kada imamo neki sistem koji ne želimo da razumijemo kako funkcioniše “ispod haube”. Korisnici vide samo kranji 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 korištenja ovog paterna u našem sistemu je primjenjen, s obzirom da korisnik vidi samo određeni dio sistema koji je njemu potreban. Primjer toga je se u pozadini kreiraju operacije kao što su stvaranje nove objave, uređivanje korisničkog profila, brisanje objava

Flyweight patern

Ovaj pattern se koristi kada kreiramo objekte samo po potrebi kada imaju različito specifično stanje, a osnovno stanje je isto za sve objekte.

Korištenjem ovog patterna se onemogućava stvaranje velikog broja instanci objekata koji u suštini predstavljaju jedan objekat. Samo ukoliko postoji potreba za kreiranjem specifičnog objekta sa jedinstvenim karakteristikama (specifično stanje), vrši se njegova instantacija, dok se u svim ostalim slučajevima koristi postojeća opća instanca objekta (bezlično stanje).
Korištenje ovog patterna je veoma korisno u slučajevima kada je potrebno vršiti uštedu memorije.

Ovaj patern smo iskoristili u našem sistemu da nakon prikaza prve tri objave, možemo učitati dodatne objave dinamički kako ih korisnik pregledava. Umjesto da odmah učitate sve objave na stranici, učitavanje dodatnih objava može se pokrenuti kada korisnik dosegne određeni dio stranice ili pritisne gumb za učitavanje više objava. Kada se objave dinamički učitavaju, preglednik može bolje upravljati memorijom jer ne mora zadržavati veliku količinu podataka u memoriji odjednom. To može poboljšati performanse aplikacije i spriječiti preopterećenje preglednika.