STRUKTURALNI PATERNI

ADAPTER PATTERN

Adapter strukturalni pattern omogućava da interfejs jedne klase prilagodimo očekivanom interfejsu druge klase, kako bi klase koje inače ne bi mogle sarađivati mogle međusobno komunicirati. Koristi se kada želimo integrisati postojeće klase s različitim interfejsima bez mijenjanja njihovog izvornog koda.

U našem sistemu ovaj patern možemo iskoristiti za slanje notifikacija različitim korisnicima. Pošto sve vrste notifikacija/obavijesti imaju različite načine slanja i interfejse, možemo kreirati adapter patern za različite tipove notifikacija i tako napraviti saradnju i zajednički interfejs između oba načina slanja obavijesti.

FACADE PATTERN

Facade pattern pruža jednostavan interfejs ka složenom podsistemu. Umjesto da korisnik mora direktno komunicirati sa više klasa i njihovim metodama, fasada nudi jednu „ulaznu tačku“ kroz koju korisnik može obaviti kompleksne operacije bez poznavanja svih detalja implementacije.

Konkretan primjer na našem sistemu bi bio da realizujemo “fasadu” za upravljanje aukcijama koja bi obuhvatila pokretanje i vođenje aukcije. U sistemu, kreiranje i vođenje aukcije uključuje interakciju sa više klasa:

- Umjetnina – mora biti učitana i validirana
- Aukcija – kreira se nova instanca
- Korisnik – mora biti verifikovan
- Status – mora biti „Aktivna“
- (Opcionalno) slanje notifikacija

U sklopu fasade bi se objedinjavali procesi validacije umjetnine, kreiranja aukcije i slanja notifikacija u jednu metodu, čime se pojednostavljuje korištenje i održavanje koda. Pri tome se kreira klasa **AukcijaFacade** koja orkestrira sve korake u jednom.

DECORATOR PATTERN

Decorator je strukturalni patern koji omogućava dinamičko dodavanje novih funkcionalnosti objektima, bez mijenjanja njihovog osnovnog koda. Umjesto da nasljeđujemo klasu kako bismo

dodali ponašanje, dekorater "umotava" objekat i proširuje njegovo ponašanje, zadržavajući isti interfejs.

U našem sistemu ovaj patern možemo primjeniti pri dekorisanju notifikacija. S obzirom da naš sistem podržava više tipova notifikacija (email i unutar aplikacije), koristeći decorator patern možemo omogućiti da korisnik prima različite notifikacije u isto vrijeme pri čemu svaku od njih možemo po želji obogatiti dodatnim informacijama poput QR koda, verifikacijskog koda, timestamp-a i slično. Umjesto pravljenja različitih klasa, pravimo dekoratore poput: **BaznaNotifikacija, QRKodDekorator, VerKodDekorator, TimestampDekorator**, itd.

Također ovaj patern možemo iskoristiti pri prikazu aukcije i umjetnina, tako što bismo osim prikaza osnovnih informacija o umjetnini (naziv, autor, cijena) također mogli dodavati dinamički I druge informacije na prikazu kao što su to status aukcije, prosječna cijena aukcije i sl. U tom slučaju ne bismo pravili odvojene klase za sve te poglede već koristimo samo dekoratore koji mogu omogućiti da se doda željena informacija na prikaz umjetnine i aukcije.

BRIDGE PATTERN

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. Koristi se kada postoji više varijacija apstrakcije i implementacije koje bi u suprotnom dovele do velikog broja podklasa. Kombinira dvije (ili više) ortogonalne hijerarhije klasa.

U našem sistemu ovaj patern bi se mogao potencijalno koristiti kod korisničkih naloga i pri autentifikaciji. U našem sistemu imamo klasu Korisnik sa različitim atributima, a ako želimo omogućiti različite vrste prijave i autentifikacije korisničkog računa bez promjene načina prijave možemo to uraditi koristeći ovaj patern. Kreiramo apstraktni dio/klasu **Autentifikacija** koja ukljucuje **AutentifikacijaKorisnika, AutentifikacijaAdministratora** i sl., a zatim u dijelu implementacije obuhvatamo klasu **AutentifikacionaStrategija** koja moze sadrzavati na primjer : **LozinkaAut, FacebookAut, GoogleAut, QRKodAut**, ukoliko bismo omogućili prijavu na sve navedene načine.

PROXY PATTERN

Proxy je patern koji omogućava da kontrolišemo pristup nekom objektu tako što umjesto njega koristimo „zamjenski“ objekat. Ovaj zamjenski objekat se ponaša kao pravi, ali može dodati dodatnu logiku — npr. autorizaciju, logovanje, keširanje, odlaganje učitavanja (lazy loading) i slično.

U našem sistemu ovaj patern možemo primjeniti prilikom učestvovanja u akciji umjetnine. Naime, samo odeđeni korisnici mogu učestvovati pri aukciji. Registrovani korisnici imaju pravo

da licitiraju/bidaju i da prate tok aukcije uživo, dok gosti to ne mogu. Također samo korisnici prijavljeni kao “kritičar” mogu postavljaju početne ponude. Umjesto da vršimo direktnu komunikaciju sa klasom Aukcija kako bismo osigurali ove zaštitne uslove, možemo kreirati klasu ProxyAukcija koja potencijalno može vršiti provjere korisnika, logovati neovlaštene radnje, bilježiti stanje aukcije i slično. Za ovaj patern kreiramo interfejs **IAukcija**, zatim pravu klasu **Aukcija** i zamjensku **ProxyAukcija** koja sadrži referencu na Aukcija i dodaje zaštitnu logiku.

COMPOSITE PATTERN

Composite je strukturalni patern koji omogućava da grupiraš objekte u hijerarhiju stabla, gdje se pojedinačni objekti i njihove grupe tretiraju na isti način. Omogućava klijentima da se prema pojedinačnim i složenim objektima (kompozicijama) ponašaju jednoliko – kao da su isti.

U našem sistemu se ovaj patern može koristiti da se organizuju umjetnine u kategorije i kolekcije. Umjetnine možemo organizovati u kategorije tipa slikarstvo, kiparstvo, digitalna umjetnost itd., koje također mogu sadržavati potkategorije tipa pejzaž, portret itd. Na osnovu toga pravimo zajednički interfejs **KolekcijskaJedinica**, nakon čega kreiramo list **Umjetnina**. Također se kreira **Kolekcija ili Kategorija** koja predstavlja kompozit jer može sadržavati druge kolekcije ili umjetnine.