Patterni Ponašanja

Strategy

Strategy pattern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada se želi enkapsulirati različiti algoritmi koji obavljaju istu funkciju, ali na različite načine bez modifikovanja *client-side* koda ili kada se želi promijeniti ponašanje objekta u zavisnosti od trenutne situaciju ili potrebe. Implementacije algoritma su neovisne od klijenata koji ih koriste.
Ovaj pattern je implementiran u našem sistemu tako što smo omogućili pretraživanje albuma. Korisnik odlučuje da li će pretragu vršiti po žanrovima, po ocjeni ili po imenu, kucajući ime albuma u posebnu traku za unos teksta.

State

State pattern omogućava objektu da mijenja svoje ponašanje u zavisnosti od unutrašnjeg stanja, tako da se ponaša kao da je promijenio svoju klasu. Koristi se kada se želi enkapsulirati logika za svaki mogući slučaj u zasebnim klasama.
Primjer korištenja ovog patterna je uvođenje stanja 'Nije objavljen' i 'Objavljen' trenutno postojećoj klasi Album. U prvom stanju, korisnici ne mogu, na primjer, slušati kratke uzorke pjesama niti pisati recenzije prije nego što se Album zvanično objavi od strane Artista. U drugom stanju, sve navedene funkcionalisti su omogućene korisnicima, nakon sto prođe odgovarajuće vrijeme.

Observer

Observer pattern definiše jedan-na-više ovisnost između objekata, tako da ako jedan objekat promijeni stanje, svi objekti vezani za njega će biti obaviješteni. Često se koristi u situacijama kada je potrebno da više objekata bude obaviješteno o promjenama u jednom objektu i kada se želi smanjiti međuzavinost između objekata.
Implementiranje Observer patterna bi se moglo svesti na uvođenje veze između klase Album i Korisnik, gdje korisnici, nakon što nastupi dan zvanične objave albuma, dobiju obavijest o istom.

Iterator

Observer pattern omogućava pristup elementima kolekcije sekvencijalno bez poznavanja kako je kolekcija strukturirana. On odvaja logiku prolaska kroz kolekciju od same kolekcije.
Korisnik može da pregleda listu svih albuma ili onih u određenom žanru. Također, moglo bi se implementirati da admini mogu imati na svom panelu listu svih korisnika i sortirati na osnovu njihovih podataka.

Command

Command pattern inkapsulira zahtjev kao objekat, što omogućava parametrabilnost klijenata sa različitim zahtjevima i izvršavanje ili bilježenje zahtjeva u određenom redoslijedu. Informacije uključuju: naziv metode, objekat koji posjeduje metodu i vrijednosti za parametre metode.
Artist ima mogućnost da objavi svoj album, nakon što unese potrebne podatke i pjesme vezane za taj album. RevAlb sistem bilježi ovaj zahtjev i postavlja album na već određenom datumu.

Interpreter

Interpreter pattern definiše interpreter koji na ulazu uzima gramatiku nekog jezika u kojem su napisane naredbe.
Primjer izvršavanje istrukcija je promjena teme web stranice (light/dark mode), promjena jezika ili puštanje isječaka muzike pri klikom na dugme.

Memento

Memento pattern služi za spašavanje određenog stanja (van sistema) nekog objekta i njegovo ponovno vraćanje, bez kršenja enkapsulacije.
U slučaju da padne RevAlb sistem, može se implementirati *rollback* funkcionalnost, tako da zadnje spašeno stanje sistema se postavi kao trenutno stanje.

Chain Of Responsibility

Chain Of Responsibility pattern rasterećuje primaoca zahtjeva tako što daje priliku drugim objektima da ga obrade. Objekti se spajaju u lanac, koji omugaćava njihovu komunikaciju i definiše redoslijed u kojem će dalje slati zahtjev u slučaju da jedan ili više objekata nisu u stanju obraditi zahtjev.
Ovaj pattern bi se mogao uvesti tako što se napravi sistem za prijavu kršenja pravila korištenja RevAlb-a od strane korisnika. Prijava se pošalje korisničkoj podršci. U slučaju da ne mogu riješiti zahtjev, prosljeđuje se dalje administratoru, pa možda nekim advokatima u slučaju da je kršenje autorskih prava sadržaj prijave.