Patterni ponašanja

1. Strategy – izdvaja algoritam iz matične klase i uključuje ga u posebne klase.

   U našem projektu, Strategy pattern bismo mogli iskoristiti za sortiranje oglasa po različitim kriterijima(po datumu, po cijeni, po broju prijava, itd.). Time bismo izbjegli if-else grananja u kodu i postigli da možemo: zamijeniti algoritme u toku izvršavanja programa, odvojiti implementaciju od koda koji ga koristi i ne narušavamo Open/Closed princip jer možemo uvesti nove strategije bez promjene postojećih klasa.

2. State - mijenja način ponašanja objekata na osnovu trenutnog stanja

   Umjesto da koristimo switch ili if logiku da provjeravamo da li stanje oglasa(kreiran, aktivan, zatvoren itd.) možemo koristiti state pattern. Također, ovim patternom ne narušavamo Single responsibility princip jer kod koji se odnosi na pojedinačna stanja je u posebnim klasama i Open/Closed princip jer uvodimo nova stanja bez promjene postojećih stanja klasa.

3. TemplateMethod

   Objava oglasa može imati unaprijed definisan „kostur“ koraka: validacija, snimanje u bazu, slanje notifikacije i sl. Koristeći TemplateMethod pattern, definira se apstraktna klasa sa šablonom, dok se konkretni koraci (npr. način validacije ili oblik notifikacije) mogu mijenjati u izvedenim klasama. Ovo omogućava jednostavno proširivanje bez narušavanja strukture.

4. Chain of responsibility - predstavlja listu objekata, ukoliko objekat ne može da odgovori prosljeđuje zahtjev narednom u nizu.

   Za obradu objave oglasa možemo imati niz „procesora“: validacija → formatiranje → zapis u bazu → notifikacija. Svaki objekat u lancu obrađuje zahtjev ili ga prosljeđuje dalje. Na ovaj način, lako se može ubaciti nova logika bez potrebe da se mijenja kompletan tok (npr. uvođenje dodatne provjere prije objave oglasa).

5. Command - razdvaja klijenta koji zahtjeva operaciju i omogućava slanje zahtjeva različitim primaocima.

   Kada radnik klikne na „Prijavi se na oglas“, ta radnja se može predstaviti kao Command objekat (npr.PrijaviseCommand) koji se može logirati, poništiti, zakazati ili ponovo izvršiti. Time se odvaja logika izvršenja akcije od dijela aplikacije koji zahtjeva akciju. Ovo omogućava implementaciju undo/redo funkcionalnosti.

6. Iterator - omogućava pristup elementima kolekcije sekvencijalno bez poznavanja interne strukture.

   Pomoću ovog pattern-a mogli bismo lahko prolaziti kroz različite kolekcije elemenata poput oglasa, notifikacija, recenzija na zahtjev korisnika što bi poboljšalo čitljivost koda i održivost, jer ne moramo se brinuti kako su ti podaci spremljeni.

7. Mediator – enkapsulira protokol za komunikaciju među objektima dozvoljavajući da objekti komuniciraju bez međusobnog poznavanja interne strukture objekta.

   Implementacija ovog pattern-a bi mogla poslužiti npr. Za komunikaciju između oglasa i notifikacija, ako se oglas završi, mediator brine o mijenjanju njegovog statusa, vidljivosti, šalje notifikacije, bez da se metode oglasa i notifikacije međusobno pozivaju što smanjuje zavisnosti klasa jednim o drugima i čini kod preglednijim i jednostavnijim.

8. Observer - uspostavlja relaciju između objekata takvu da kad se stanje jednog objekta promijeni svi vezani objekti dobiju informaciju.

   Observer pattern omogućava da jedan objekat (oglas) obavještava druge objekte (npr. Notifikacije) automatski o promjenama svog stanja, bez da ih direktno povezuje. Na primjer, kad se status oglasa promijeni, svi registrovani objekti odmah dobiju obavijest i mogu reagovati, recimo prikazati update korisniku ili poslati notifikaciju radnicima.

9. Visitor - definira i izvršava nove operacije nad elementima postojeće strukture ne mijenjajući samu strukturu.

   Kad želimo da izvršimo više različitih operacija nad istim objektima bez da ih mijenjamo (npr. Oglas, Korisnik, Recenzija), možemo koristiti Visitor pattern. Tako možemo dodati funkcionalnosti poput generisanja izvještaja, slanja podsjetnika, slanja e-mailova ili notifikacija bez mijenjanja osnovnih klasa modela.

10. Interpreter - podržava interpretaciju instrukcija napisanih za određenu upotrebu.

    Ako radnik želi napraviti složeniji upit, npr. „pronađi oglase koji su aktivni, sa rokom većim od 5 dana, a cijena veća od 100 KM“, možemo implementirati jednostavan jezik za pretragu. Interpreter pattern omogućava parsiranje i izvršavanje takvog jezika unutar aplikacije, što povećava fleksibilnost pretrage bez komplikovane logike unutar UI-a.

11. Memento - omogućava spašavanje internog stanja nekog objekta van sistema njegovo ponovno vraćanje.

    Za funkcionalnosti vraćanja starih verzija – npr. opis oglasa, stariji CV, prethodna verzija profila – Memento pattern omogućava spremanje „snapshota“ objekta. Time se omogućava korisnicima da se vrate na prethodno stanje bez uticaja na ostatak sistema. Ovaj pattern također se može iskoristiti za poništavanje izmjena.