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

   Ovaj pattern bismo mogli implementirati u procesu objave oglasa koji sadrži više koraka: validacija podataka, snimanje oglasa u bazu i slanje notifikacija. Ovim bismo dobili jednostavnu strukturu, a mogli bismo pojedine dijelove implementacije jednostavno mijenjati.

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

Slično kao kod TemplateMethod pattern-a, Chain of responsibility bi se mogao implementirati za proces objave oglase, s tim da ovaj pattern ima prednost da jednostavnije možemo ubacivati potrebne procese među postojeće.

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

   Kada se radnik prijavi na neki oglas, umjesto da to direktno radi kontroler, imamo objekte koji brinu o tim radnjama. Na taj način, radnje se mogu lahko logirati, zakazati ili poništiti.

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.

   U našoj aplikaciji bismo mogli iskoristit ovaj pattern za slanje obavijesti radniku na rok izvršavanja posla ili klijentu za rok isplate urađene usluge itd.

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

    Radnik bi mogao napisati neki složeniji upit za filtriranje oglasa,a interpreter bi to protumačio i vratio odgovarajuće rezultate.

11. Memento - omogućava spašavanje internog stanja nekog objekta van sistema njegovo ponovno vraćanje. Može se iskoristiti za razne stvari poput, spašavanja starijih verzija opisa nekog oglasa koji klijent napiše, CV-a koji radnik postavi na svoj profil.