Strukturalni paterni

Naš postojeći sistem obuhvata entitete poput korisnika, termina, medicinskih usluga i nalaza, ali kako bismo postigli veću fleksibilnost i ponovnu iskoristivost koda, poboljšali modularnost, smanjili međuzavisnost klasa, uvodimo dva strukturalna paterna:

 Facade - koristi se kada sistem ima više podsistema (subsystems) pri čemu su apstrakcije i implementacije podsistema usko povezane

 Decorator - omogućava dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima

Facade

Facade je strukturalni patern koji pruža pojednostavljen interfejs za složen skup klasa i sistema, omogućavajući lakšu upotrebu kompleksnih podsistema. Implementacija u projektu, služi za pojednostavljenje složenih procesa kao što je zakazivanje termina. Umjesto da klijent direktno komunicira sa klasama poput Korisnik, Termin ili Lokacija, koristi se jedinstveni interfejs ZakazivanjeFacade koji pruža jednostavne metode poput zakaziTermin(). Na taj način se smanjuje kompleksnost, poboljšava modularnost sistema i olakšava povezivanje korisničkog interfejsa s poslovnom logikom.

Decorator

Decorator je patern koji omogućava dinamičko dodavanje novih funkcionalnosti postojećim objektima, bez izmjene njihovog izvornog koda. Implementacija ovog paterna u našem projektu omogućava da se osnovnom nalazu dinamički dodaju nove funkcionalnosti, bez potrebe za mijenjanjem postojećeg koda. U sistemu se koristi za obogaćivanje nalaza dodatnim elementima kao što su potpis doktora, komentari ili prevod. Na primjer, možemo kombinovati dekoratore tako da jedan nalaz istovremeno ima i potpis i komentar, što omogućava fleksibilno proširivanje bez dupliranja klasa i koda.

Adapter

Adapter je patern koji omogućava povezivanje klasa s nekompatibilnim interfejsima tako što prilagođava jedan interfejs drugome. Implementacija ovog paterna omogućava povezivanje sa servisima za autentifikaciju poput Google OAuth. Kroz AuthAdapter, koji prilagođava interfejs vanjskog servisa onome što naš sistem očekuje, omogućujemo korisnicima da se prijavljuju koristeći moderne sigurnosne protokole bez izmjena postojećih klasa.

Bridge

Bridge je patern koji razdvaja apstrakciju od njene implementacije kako bi obje strane mogle nezavisno evoluirati. Implementacija ovog paterna bi dozvolila da se razdvaja sadržaj nalaza od načina njegovog prikaza. Umjesto da svaka verzija nalaza ima sopstvenu implementaciju za prikaz u PDF-u, HTML-u ili drugom formatu, koristi se

struktura gdje Nalaz koristi različite implementacije prikaza kroz interfejs FormatImplementor. Na ovaj način se lako može dodati novi format bez izmjene osnovne logike sistema.

Proxy

Proxy je patern koji djeluje kao zamjenski objekat i kontroliše pristup drugom objektu, često dodajući sigurnosnu ili kontrolnu logiku. Implemetacija ovog paterna kreira sloj sigurnosti između korisnika i osjetljivih podataka kao što su medicinski nalazi. Kroz NalazProxy možemo kontrolisati da li određeni korisnik ima pravo da vidi ili izmijeni konkretni nalaz. Ovaj patern omogućava implementaciju pravila pristupa bez uticaja na samu klasu Nalaz, čime se povećava sigurnost sistema.

Composite

Composite je patern koji omogućava da se pojedinačni objekti i njihove kolekcije tretiraju na isti način kroz zajednički interfejs. Implementiranjem ovog paterna u sistem omogućava kreiranje paketa medicinskih usluga (npr. sistematski pregled koji sadrži više pojedinačnih pregleda). Kroz zajednički interfejs UslugaComponent, i PojedinacnaUsluga i PaketUsluga se tretiraju jednako, što pojednostavljuje upravljanje kompleksnim uslugama.

Flyweight

Flyweight je patern koji minimizira upotrebu memorije dijeljenjem što je moguće više zajedničkih podataka između sličnih objekata Korištenjem ovog paterna, umjesto da se za svaku pojavu lokacije kreira nova instanca, koristi se LokacijaFactory koja vraća postojeće instance za iste adrese. Time se značajno smanjuje potrošnja memorije, što je posebno važno u sistemima sa velikim brojem termina i pacijenata.