Paterni ponašanja

Potreba za implementacijom paterna ponašanja u sistem BookMyStyle

Implementacija paterna ponašanja u sistem BookMyStyle je ključna radi povećanja fleksibilnosti, skalabilnosti i jasnoće interakcija između komponenti sistema.

Pregled dizajn paterna ponašanja:

Strategy pattern

Strategy pattern izdvaja algoritam iz matične klase i uključuje ga u posebne klase. U našem slučaju, može se koristiti za fleksibilno sortiranje usluga po različitim kriterijima poput cijene, lokacije ili dostupnosti. Ovaj pattern sigurno odgovara za klasu recenzija i usluga u našem sistemu. Ovaj pattern omogućava korisnicima jednostavan odabir najpogodnijeg načina sortiranja usluga, bez puno zahtjeva i izmjena UI. Implementacija različitih strategija sortiranja ne utiče na postojeći kod. To značajno povećava modularnost sistema i omogućava jednostavniju implementaciju novih algoritama. Strategy pattern time olakšava buduće proširenje sistema i dodavanje novih funkcionalnosti po potrebi.

State pattern

State pattern mijenja način ponašanja objekata na osnovu trenutnog stanja. Pogodan je za implementaciju različitih stanja rezervacija (npr. potvrđena, otkazana, završena) i u modelu obavijesti gdje se prikazuju obavijesti i rezravacije za svakog korisnika. Koristeći State pattern, objekti rezervacija jasno komuniciraju svoje trenutno stanje, omogućavajući jasnije upravljanje rezervacijama. Ovaj pattern implementiran je u termin modelu, te na taj način olakšana je rezervacija termina. Implementacija ovoga paterna omogućava lakše održavanje i dodavanje novih stanja bez promjene postojeće logike. Također olakšava debugging i održavanje sistema jer stanja imaju izoliranu logiku.

Template Method pattern

Template Method omogućava algoritmima izdvajanje pojedinih koraka u podklase. Prikladan je za implementaciju različitih načina obrade rezervacija, zadržavajući osnovni postupak zajedničkim. Ovaj pristup omogućava jednostavnu promjenu specifičnih koraka obrade bez izmjene cijelog algoritma. U našem sistemu rezeravacija se odvija preko modela termina, uskuge, obavijesti i na taj način je smanjena složenost same rezravacije koja se izvršava preko ove tri klase. Na primjer, osnovna obrada rezervacija može ostati ista, dok se posebni koraci prilagođavaju pojedinačnim potrebama korisnika ili salona. Time se postiže visoka razina ponovne upotrebe i modularnosti sistema.

Mediator pattern

Mediator pattern enkapsulira komunikaciju između objekata i omogućava lakšu koordinaciju akcija između klijenata, frizera i administratora. Centraliziranjem komunikacije smanjuje se složenost interakcija među objektima. Omogućava lako upravljanje promjenama u sistemu jer se izmjene u komunikaciji vrše samo na jednom mjestu. U našem slučaju može biti upravljanje rezervacijama gdje mediator koordinira akcije korisnika, salona i sistema obavještavanja. Time se povećava fleksibilnost i održivost sistema.
Npr. Klasa BookingMediator upravlja interakcijama između korisnika, termina i frizera. Kada korisnik zakaže termin, BookingMediator obavještava sve uključene strane (npr. frizera, korisnika, šalje email obavijest)

Memento pattern

Memento omogućava spašavanje i ponovno vraćanje stanja objekta, što može biti korisno za oporavak stanja rezervacija ili uređivanja korisničkih računa. Svaki put kada korisnik ili admin uređuje termin ili profil, stanje se sprema kao memento. Ako dođe do greške sistem jednostavno pozove restore().
Korištenjem mementa moguće je kreirati sigurnosne kopije stanja objekata. Time se omogućava jednostavan oporavak u slučaju grešaka ili neplaniranih promjena. Memento pattern povećava pouzdanost sistema i korisničko zadovoljstvo. Omogućava i pregled povijesti promjena na intuitivan i siguran način.

Chain of Responsibility

Ovaj pattern predstavlja listu objekata koji obrađuju zahtjev. Pogodan je za rješavanje zahtjeva korisnika ukoliko osnovni objekt ne može izvršiti akciju, zahtjev se prenosi sljedećem objektu. Omogućava lako proširenje sistema novim objektima za obradu bez promjena u postojećem kodu. Primjer primjene može biti u validaciji zahtjeva za rezervacijom termina, gdje se zahtjev šalje kroz više provjera. U našem slučaju zahtjev i uneseni podaci se validiraju prilikom unosa, a iako se na neki način unseu neispravni podaci takvi se ne unose u bazu zbog constrainta. Chain of Responsibility pattern povećava fleksibilnost i smanjuje potrebu za tijesnom spregom između objekata.

Command pattern

Command pattern pretvara zahtjev u samostalni objekt omogućavajući jednostavno rukovanje, red čekanja i opcije undo/redo, što je korisno za operacije zakazivanja i uređivanja termina. Koristeći Command pattern, moguće je jasno pratiti, izvršavati i opozivati akcije korisnika. Svaka akcija korisnika se može predstaviti kao klasa koja implementira zajednički interfejs Command, s metodama execute() i undo(). Kreiramo konkretne komande npr. ZakaziTerminCommand, OtkaziTerminCommand, koje upravljaju instancom Termin. Također omogućava lakšu implementaciju funkcionalnosti povijesti izmjena ili revizije akcija korisnika. Svaka komanda može biti spremljena, a zatim ponovno izvršena ili opozvana. To značajno povećava korisničko iskustvo i stabilnost sistema.

Iterator pattern

Iterator omogućava pristup elementima kolekcije bez otkrivanja interne strukture. Pogodan je za pregled liste dostupnih usluga ili termina. Omogućava da korisnički interfejs pristupi podacima na jednostavan i konzistentan način. Time se postiže bolja organizacija i jasnoća u prikazu podataka korisnicima. Implementacija iteratora olakšava eventualne izmjene u strukturi kolekcije bez potrebe za izmjenama koda koji je koristi.

Observer pattern

Observer uspostavlja relaciju između objekata tako da promjene stanja automatski obavještavaju sve zainteresovane strane (ujedno svaki akter sistema). Izuzetno koristan za automatsko obavještavanje korisnika o statusu rezervacija. Korištenjem ovog paterna, promjene u sistemu kao što su rezervacija ili otkazivanje termina mogu automatski slati notifikacije korisnicima. Observer pattern smanjuje spregu između subjekata i posmatrača te omogućava jednostavno dodavanje novih oblika obavještavanja. Time sistem postaje dinamičniji i prilagodljiviji.

Način primjene:

- Korisnik i Frizer su (posmatrači) termina.
- Kada se termin promijeni, otkaže ili potvrdi, termin automatski šalje notifikaciju svim observerima.

Visitor pattern

Visitor definiše i izvršava nove operacije nad elementima postojeće strukture bez promjene same strukture. Može biti primijenjen za generisanje izvještaja ili analiza usluga i korisnika. Omogućava lako dodavanje novih operacija poput izvještaja, statistike ili analize podataka bez izmjene originalnih klasa. Može se koristiti za generisanje izvještaja (npr. najtraženije usluge, broj termina po korisniku). StatisticVisitor može obići sve termine i korisnike, generirati mjesečne statistike. Visitor pattern održava postojeći kod čistim i dobro strukturiranim. Time omogućava lakše održavanje i proširenje funkcionalnosti.

Interpreter pattern

Interpreter podržava interpretaciju specifičnih korisničkih zahtjeva, omogućavajući fleksibilniju pretragu i filtriranje termina. Pogodan za parsiranje i obradu kompleksnih korisničkih upita. Npr. Implementacija pretraživača koji razumije izraze poput: "frizer AND vikend AND slobodno prije 17h". Klase poput AndExpression, OrExpression, BeforeTimeExpression mogu se kombinovati za parsiranje i izvršavanje logike pretrage.

Interpreter omogućava jednostavnu implementaciju novih formata ili pravila pretrage bez izmjena u postojećoj logici. Time se značajno olakšava prilagodba sistema specifičnim potrebama korisnika. Implementacija ovoga paterna olakšava dodavanje novih funkcija i opcija filtriranja.