Univerzitet u Sarajevu
Elektrotehnički fakultet
Sarajevo

Deveti projektni zadatak

Paterni ponašanja

Objektno orijentisana analiza i dizajn

Naziv grupe: Booking
Članovi: Ajdin Dželo
Nejra Vatrić
Amna Glamoč
Adna Bajramović

Sarajevo, maj 2025

Paterni ponašanja u booking sistemu

Strategy pattern

Strategy pattern omogućava dinamičku promjenu načina na koji se izvršava neka funkcionalnost. Umjesto korištenja komlikovanijih if-else grananja za filtriranje i sortiranje smještaja, sistem koristi odvojene klase – strategije – koje implementiraju isti interfejs. Korisnik odabere kriterij, a odgovarajuća strategija se automatski primjenjuje.

U našem sistemu kod pretraživanja smještaja korisnik može odabrati da vidi najjeftinije smještaje, smještaje po određenoj lokaciji, vrsti smještaja itd. Svaka od tih metoda pretraživanja je posebna strategija. Sistem dakle koristi onu koju korisnik odabere.

State

State pattern omogućava objektima da promijene ponašanje kada im se promijeni stanje. Umjesto da se na više mjesta provjerava da li je rezervacija potvrđena ili otkazana, koristimo klase koje predstavljaju ta stanja. Konkretno u sistemu, rezervacija može biti u više stanja; kreirana, potvrđena, otkazana. Svako stanje ima drugačije ponašanje – npr. potvrđena se više ne može uređivati, otkazana može da se refundira itd. Svako stanje implementira ponašanje rezervacije na svoj način.

TemplateMethod pattern

Template Method definiše osnovu algoritma, ali omogućava podklasama da konkretne korake prilagode. Pa kod generisanja izvještaja (npr. broj rezervacija po lokaciji, prosječne ocjene, najaktivniji korisnici) svi izvještaji imaju sličnu strukturu: učitaj podatke, obradi ih, prikaži. Svaki tip izvještaja popunjava te korake drugačije.

Chain of Responsibility

Chain of Responsibility omogućava da se zahtjev (npr. zahtjev za rezervaciju) prosljeđuje nizom objekata dok ga neki ne obradi. To omogućava fleksibilnu obradu bez potrebe za „if-else“ grananjem.

Kod rezervacije, sistem može imati više provjera: dostupnost smještaja, validnost korisnika, uslovi plaćanja itd. Svaka provjera je član lanca – ako jedna ne prođe, rezervacija se ne nastavlja. Tako se logika validacije lijepo raspoređuje u nezavisne dijelove.

Command pattern

Command pattern enkapsulira zahtjeve kao objekte, što omogućava npr. undo/redo funkcionalnosti, logovanje komandi i planiranje izvršavanja. Dakle, kada korisnik kreira ili otkaže rezervaciju, te akcije se zapisuju kao komande. Tako možemo čuvati historiju akcija, dodati opciju "poništi zadnju rezervaciju" ili omogućiti izvršavanje komandi kasnije.

Iterator pattern

Iterator omogućava pregled elemenata kolekcije bez poznavanja njene unutrašnje strukture.

Konkretno, kada prikazujemo listu smještaja, recenzija ili korisnika, koristimo iterator koji omogućava da aplikacija ide redom kroz elemente – bez direktnog rada s listama. Pomoćno kod implementacije beskonačnog scrolla.

Mediator pattern

Mediator centralizuje komunikaciju između klasa tako da one ne komuniciraju direktno, već preko posrednika. Kada se korisnik prijavi, LogController se ne treba direktno povezivati sa drugim kontrolerima. Umjesto toga, svi kontroleri komuniciraju preko Mediatora koji zna kako ih povezati. To smanjuje kompleksnost među klasama i povećava fleksibilnost.

Tvoj sistem ima više pogleda (RegisterView, LoginView, RezervacijaView…). Umjesto da svaka klasa zna za sve druge, centralni „mediator“ može povezivati sve zahtjeve i odgovore između njih. Na primjer, kada korisnik završi registraciju, mediator obavještava LoginView da ga automatski prijavi.

Observer pattern

Observer obrazac omogućava da više objekata bude obavješteno kada se stanje nekog objekta promijeni. Recimo ako administrator doda novi smještaj, korisnici koji su se prijavili za obavijesti mogu automatski dobiti informaciju o tome (email notifikacija ili prikaz na početnoj stranici).

Kad korisnik ostavi recenziju za smještaj, vlasnik (ili admin) može biti obaviješten automatski. Takođe, ukupna ocjena se odmah ažurira i prikazuje na listi smještaja. Sve to može da se odvija preko Observer pattern-a.

Visitor pattern

Visitor omogućava dodavanje nove operacije postojećim klasama bez da ih mijenjamo, tako što sve operacije smjestimo u „posjetioca“. Npr. ako želimo dodati funkcionalnost poput izračuna ukupne zarade po korisniku, ili generisanje statistike po smještaju – a da ne diramo postojeće klase Korisnik, Smjestaj itd – koristit ćemo Visitor. On obilazi te objekte i obavlja potrebne kalkulacije.

Interpreter pattern

Interpreter definiše gramatiku za jezik i koristi je za interpretaciju izraza u tom jeziku. Ako korisnik ima mogućnost napredne pretrage koristimo Interpreter koji razumije i izvrši takve izraze, umjesto da ručno parsiramo sve uslove.

Memento pattern

Memento pattern omogućava da sačuvaš i vratiš prethodno stanje objekta bez otkrivanja njegove unutrašnje strukture. Recimo da korisnik uređuje rezervaciju, moguće je omogućiti mu "vrati na prethodno stanje". Prije nego se promjena sačuva, sistem kreira „memento“ trenutnog stanja, koji se kasnije može vratiti ako korisnik odustane.