PATERNI PONAŠANJA

1. Strategy pattern

- U našem sistemu je moguće upotrijebiti Strategy patern na sljedeći način:
S obzirom na to da imamo opciju filtriranja pretrage po recenziji i cijeni, umjesto da dodajemo if-else blok pomoću kojeg ćemo znati koja opcija filtriranja pretrage je izabrana, mi možemo kreirati dvije posebne klase: filtrirajPoCijeni i filtrirajPoRecenziji, koje će implementirati interfejs IFiltriranjeSmjestaja. Taj interfejs ima metodu filtriraj (S: List<Smjestaj>) void koja će biti implementirana u te dvije klase za filtriranje. Također, taj intefejs treba da naslijedi i neka nova klasa FiltriraniSmjestaj koja će u sebi imati listu smještaja kao i atribut tipa tog interfejsa. Od metoda koje će ona imati su promijeniNacinFiltriranja (i:IFiltriranjeSmjestaja) i filtriraj(). Metodu filtriraj() pozivamo direktno nad objektom FiltriraniSmjestaj, dok za promjenu smještaja je također pozivamo nad objektom tipa FiltriraniSmjestaj, i kao argument šaljemo novi objekat tipa intefejsa koji će predstavljati novi način filtriranja.
Klasa FiltriraniSmjestaj bi trebala biti povezana sa kontrolerom za filtriranje smještaja, preko DBContext klase.

2. State pattern

- U zavisnosti od strategy pattern-a, kod state patterna će se automatski prelaziti u drugo stanje u zavisnosti od nekog uslova, koji je najčešće izražen preko neke varijable. Recimo ako je korisnik online, varijabla će imati vrijednost 1, a ako je offline imat će 0. U sistemu imamo opcije dijeljenja lokacije i prikaza pinovanih hotela u blizini te korisnikove trenutne lokacije, međutim nije moguće da korisnik dijeli lokaciju i uopće dobija pinovane hotele ako nema internetske konekcije. Zbog toga bi bilo dobro da ovaj pattern upotrijebimo tako što ćemo imati interfejs sa metodama pretraziSmjestaj(), prikaziPinovaneHotele(). Klasa korisnik će implementirati interfejs IState koji sadrži navedene metode. Također u klasi korisnik će se nalaziti atributi “stanje” tipa bool i atribut tipa interfejsa IState. U zavisnosti od bool atributa iz klase Korisnik, te metode će ili baciti izuzetak, gdje ćemo kroz pogled dati korisniku do znanja da je došlo do greške radi internet konekcije, ili će korisniku biti prikazani pinovani hoteli.

3. Template method

- Template method pattern možemo u našem sistemu iskoristit u vezi sa smještajem. Smještaj klasa bila bi nam apstraktna klasa koja bi posjedovala metode koje bi naravno bile implementirane u klasama koje je nasljeđuju. Klase koje bi nasljeđivale apstraktnu klasu Smještaj su klase Motel, Hostel, Hotel, Apartman. Smisao je da bi svaki smještaj imao različitu implementaciju za dodavanje atributa naslijeđenih iz klase smještaj.

4. Observer

- Observer pattern s obzirom na svoju primjenu, u našem sistemu bi se mogao upotrijebiti u smislu automatskog pinovanja hotela na mapi koja se prikazuje korisniku prilikom ulaska na našu stranicu. Smisao je da se automatski poziva metoda pinujHotele() prije koje će se svakako morati podijeliti lokacija koja će uzrokovati poziv navedene metode. Također mora postojati metoda koja će provjeravati da li je zahtjev za pinovanje hotela uputio RegistrovaniKorisnik ili Gost korisnik jer u slučaju da Gost koristi aplikaciju onda neće biti dozvoljeno pinovanje hotela, a u suprotnom metoda pinujHotele() će raditi očekivano.

5. Iterator

- Iterator pattern u našem sistemu bismo mogli upotrijebiti za "prolazak" kroz smještaje, ali moglo bi se upotrijebiti i u admin panelu za prolazak kroz korisnike koji su registrovani u sistemu. Što se tiče prvog slučaja, imali bismo klase koje bi nam predstavljale kakav je to način prolaska kroz smještaje pa bismo imali: PoCijeniIterator, PoRecenzijiIterator, PoAbecediIterator i sl. Što se tiče drugog slučaja, implementacija bi bila slična samo bi nam se drugačije zvale neke klase za iteriranje, npr.: BrojRezervacijaIterator, PoAbecediIterator, PoUtrošenomNovcuIterator, PoAktivnimRezervacijamaIterator itd.

6. Chain of responsibility

- Što se tiče Chain of responsibility patterns, u našem sistemu mogli bismo ga iskoristiti u slučaju plaćanja rezervacije za neki smještaj. To bi se uradilo na način da korisnik prvo mora odabrati smještaj, pa onda rezervisati, a nakon toga bi mu se prikazale opcije za plaćanje gdje bi mogao izabrati željenu opciju. Nakon odabira načina plaćanja, korisniku će biti ponuđena opcija za plaćanje ili za poništavanje rezervacije.

7. Medijator

- Medijator pattern u našem sistemu bismo mogli iskoristit za komunikaciji klijenta sa support osobljem, a to je u stvari naš administrator. Ukoliko dođe do neke greške pri rezervaciji smještaja ili ukoliko je potrebno refundirati novac, a to se nije desilo automatski preko sistema plaćanja ili nešto slično, tada bismo mogli ovo iskoristiti, tj. ostvariti komunikaciju sa support osobljem. To bi se implementiralo na način da imamo interfejs IMedijator koji će biti povezan sa klijentom i administratorom i koji će implementirati metode koje će provjeravati da li se nalazi neki govor mržnje ili pravi zahtjev korisnika za pomoć.