PATERNI PONAŠANJA

1. Strategy pattern

Koristili bi se za filtriranje vozila po različitim kriterijima, kao što su tip goriva, brend, boja, godina proizvodnje, transmisija i slično. Svaka od ovih logika filtriranja bila bi implementirana kao posebna strategija koja implementira zajednički interfejs, npr. VoziloFilterStrategy. Prilikom pretrage, korisnički interfejs (UI) bi slao parametre koji određuju koju strategiju filtriranja treba koristiti. Sistem bi dinamički birao i primjenjivao odgovarajuću strategiju bez potrebe za pisanjem if-else logike.

2. State pattern

Koristili bi se za upravljanje ponašanjem korisničkog interfejsa i dostupnim funkcionalnostima u zavisnosti od uloge korisnika. Na primjer, Korisnik objekat bi mogao imati atribut state, koji bi referencirao instancu stanja (npr. AdministratorState, VlasnikState, KupacState). Svaka uloga bi omogućavala ili zabranjivala određene radnje: npr. administrator može upravljati korisnicima, vlasnik može objavljivati oglase i upravljati vozilima, dok običan korisnik može samo pretraživati i rezervisati vozila. Time bi se ponašanje kontrolisalo kroz stanje umjesto ručnog provjeravanja uloge.

3. Template Method

Koristili bi se za definisanje šablona toka rada koji se ponavlja kod sličnih akcija. Na primjer, kod slanja poruke, kreiranja recenzije ili objave oglasa. Apstraktna klasa Akcija bi definisala korake kao što su: validiraj(), autorizuj(), izvrsi(), loguj(). Konkretnim klasama bi se ostavilo da implementiraju specifične detalje svakog od ovih koraka, ali bi sam redoslijed izvršavanja bio fiksiran. To bi osiguralo konzistentnost i ponovnu upotrebljivost koda.

4. Chain of Responsibility

Koristili bi se za obradu zahtjeva za rezervaciju, gdje bi se niz uzastopnih koraka izvršavao – svaki handler provjerava određeni uslov. Na primjer:

- ValidacijaKorisnikaHandler provjerava da li je korisnik prijavljen.
- ProvjeraDostupnostiHandler provjerava da li je vozilo dostupno za odabrani period.
- ProvjeraDatumaHandler provjerava validnost unesenih datuma.

Svaki handler odlučuje da li može sam završiti obradu ili da li treba proslijediti zahtjev sljedećem handleru u lancu. Ovo omogućava fleksibilno dodavanje i reorganizaciju koraka bez promjene glavne logike.

5. Command pattern

Koristili bi se za enkapsulaciju korisničkih zahtjeva kao objekata, čime bi se omogućilo jednostavno upravljanje operacijama kao što su kreiranje rezervacije, otkazivanje, dodavanje recenzije itd. Na primjer, komanda KreirajRezervacijuCommand bi imala metodu execute(), dok bi OtkaziRezervacijuCommand mogla imati i undo(). Kontroler bi upravljao redoslijedom izvršavanja komandi i po potrebi omogućio poništavanje posljednjih akcija, npr. kod greške ili ako korisnik odustane.

6. Observer pattern

Koristili bi se za automatsko obavještavanje dijelova aplikacije kad dođe do promjene u stanju. Na primjer, kada se kreira nova rezervacija ili recenzija, svi prikazi vozila ili profila koji zavise od tih podataka bi se automatski ažurirali. Rezervacija i Recenzija bi bile *Subject*, dok bi korisnički interfejs (npr. ProfilView, OglasView) bio *Observer* koji se registruje i reaguje na promjene. To bi omogućilo automatsko osvježavanje sadržaja bez ručnog poziva.

7. Visitor pattern

Koristili bi se za dodavanje dodatnih funkcionalnosti postojećim klasama kao što su Vozilo, Rezervacija i Recenzija, bez potrebe da se mijenja njihov kod. Na primjer, ako se želi dodati statistička analiza (prosječna ocjena, broj rezervacija, najpopularnije vozilo), to bi se moglo implementirati kao poseban Visitor, npr. StatistickiIzvjestajVisitor. Taj objekat bi obilazio sve entitete i prikupljao podatke bez da ih ti entiteti sami moraju računati.

8. Interpreter pattern

Koristili bi se za interpretaciju jednostavnog jezika pretrage koji bi korisnik unosio u aplikaciju. Na primjer: "dizel i automatik i godina > 2018". Svaki izraz bi bio predstavljen kao komponenta u stablu (AST – Abstract Syntax Tree), a evaluator bi tumačio logiku izraza. Time bi korisnicima bio omogućen moćan način pretrage oglasa uz podršku za logičke operatore i uslove.

9. Memento pattern

Koristili bi se za spremanje stanja objekta prije nego što se izvrši izmjena. Na primjer, kada korisnik uređuje svoj profil ili detalje rezervacije, trenutno stanje objekta bi se sačuvalo kao *memento*. Ako korisnik odluči da odustane od izmjena, sistem bi mogao vratiti prethodno stanje bez potrebe za dodatnom logikom za ručno resetovanje polja. Također bi bio koristan za „poništi“ funkcionalnosti.