STRUKTURALNI PATERNI

1. Bridge Pattern

U kreiranom klasnom sistemu nema nasljeđivanja, posebno ne višestrukog nasljeđivanja. Umjesto toga, postoji jedna klasa Korisnik koja kroz svoj enum definiše tipove korisnika: Administrator, Tehnička podrška, Vozač i Putnik. Ovakav pristup pojednostavljuje strukturu i izbjegava složenost i probleme koje višestruko nasljeđivanje može donijeti.

Međutim, ako bi u budućnosti postojale različite vrste putnika ili vozača sa značajno različitim ponašanjem ili svojstvima, tada bi korištenje Bridge paterna imalo smisla. On bi omogućio da apstrakcija (npr. korisnik) i konkretne implementacije (različite vrste putnika ili vozača) budu odvojene i nezavisno proširive, čime bi se sistem učinio fleksibilnijim i održivijim.

2.Adapter Pattern

Adapter pattern se koristi kada dvije klase imaju nekompatibilne interfejse, ali treba da rade zajedno. U trenutnom sistemu, sve klase i komponente su dizajnirane unutar iste aplikacije, sa međusobno usklađenim interfejsima, pa Adapter nije bio potreban.

Međutim, Adapter bi mogao biti koristan ako bi se povezivao vanjski sistem koji koristi drugačiji način rada.

3. Decorator Pattern - Dodavanje više kanala/načina za slanje notifikacija

Sistem je koristio klasu Notifikacija za slanje obavještenja korisnicima. Međutim, u budućnosti korisnik može odabrati da želi poslati/primiti obavještenje putem:

- Email notifikacije
- SMS poruke
- Email i SMS

Umjesto da dodajemo sve ove funkcionalnosti direktno u Notifikacija klasu (što krši SRP), mogli bismo iskoristiti **Decorator pattern**. Ovaj primjer je dodan na dijagramu klasa.

4. Proxy Pattern

Trenutno sistem ne koristi Proxy pattern, jer nema potrebe da se nešto posebno štiti ili odgađa učitavanje podataka. Ipak, postoje situacije u kojima bi ovaj obrazac mogao biti koristan.

Jedan primjer je korištenje Google Maps servisa za prikaz lokacije ili izračun rute. Pozivanje ovog servisa može biti sporo ili ograničeno, pa bi Proxy mogao:

- privremeno sačuvati rezultate (keširanje),
- kontrolisati koliko često se servis poziva,
- pojednostaviti rad sa Google Maps iz ostatka sistema.

Drugi primjer je prijava korisnika u sistem. Proxy bi se mogao iskoristiti da:

- spriječi neulogovanog korisnika da pristupi važnim funkcijama,
- dozvoli samo administratorima da npr. brišu korisnike ili upravljaju vožnjama.

Dakle, iako sada nije neophodan, Proxy pattern bi bio koristan ako se sistem proširi i počne koristiti vanjske servise ili zahtijevati veću sigurnost i kontrolu pristupa.

5.Composite Pattern

Uvođenje Composite paterna u našem klasnom sistemu bi imalo smisla izmedju klasa Lokacija i Ruta, pa je on jedan od paterna kog smo izabrali predstaviti na dijagramu klasa.
Klasa Ruta se sastoji od više lokacija, mogli bismo posmatrati kao da svaka od njih ima i svoje ‘podlokacije’ i to nam je bila motivacija za uvođenje ovog paterna. U slučaju da Ruta ima hijerarhijsku sturkturu(npr. Ruta unutar rute), ovakav obrazac bi pomogao pri upravljanju složenijom strukturom.

6. Facade Pattern

U sistemu ne bi imalo smisla primijeniti Facade pattern jer je većina klasa (Korisnik, Lokacija, Ruta, Vožnja, Notifikacija, Ocjena) relativno jednostavna i imaju jasno definirane odgovornosti. Svaka od tih klasa već obavlja specifične zadatke, a korisnici sistema trebaju pristupiti direktno tim klasama za obavljanje operacija (npr. pronalaženje rute, rezervacija vožnje, ocjena vožnje).

Facade pattern se obično koristi kada imamo složeni sistem s mnogo međusobno povezanih klasa i želimo pojednostaviti interfejs za korisnika, skrivajući unutrašnje komponente. U ovom kontekstu implementacija Facade patterna mogla bi jedino stvoriti nepotrebne složenosti u sistemu.