SOLID PRINCIPI

1. **Single responsibility principle - SRP (princip pojedinačne odgovornosti)**
   Prema ovom principu, svaka klasa treba imati samo jedan razlog za promjenu, tj. treba biti odgovorna za jednu jedinu funkcionalnost u sistemu.

Svaka klasa u našem sistemu osmišljena je na način da implementira tačno određeni dio poslovne logike aplikacije.

- **Klasa `Korisnik`**
  Apstraktna bazna klasa - uloga je isključivo čuvanje zajedničkih atributa za klase Administrator, Vožnja, Putnik, TehničkaPodrška.
- **Klasa `Vožnja`**
  Zadužena isključivo za upravljanje procesom vožnje – uključuje naručivanje, prihvatanje, praćenje statusa i ocjenjivanje vožnje.

- **Klasa `Putnik`**
  Implementira korisničke funkcionalnosti sa stanovišta putnika – naručivanje vožnje, slanje poruka vozaču i davanje ocjene nakon vožnje.

- **Klasa `Vozač`**
  Fokusirana je na funkcionalnosti relevantne za vozača – prihvatanje vožnje i komunikacija sa putnicima.

- **Klasa `TehničkaPodrška`**
  Bavi se isključivo komunikacijom sa korisnicima i sistemom radi rješavanja tehničkih problema i pružanja potrebne pomoći.

- **Klasa `Ruta`**
  Njena jedina odgovornost je izračun procijenjenog vremena dolaska na osnovu trenutne lokacije i udaljenosti.

- **Klasa `Lokacija`**
  Implementira funkcionalnosti određivanja lokacije vozača u realnom vremenu.

- **Klasa `Notifikacija`**
  Odgovorna isključivo za slanje i prikazivanje notifikacija korisnicima, uključujući obavještenja o narudžbama i statusima vožnji.

- **Klasa `Ocjena`**
  Služi isključivo za obradu i upravljanje ocjenama koje putnici ostavljaju nakon završene vožnje.

2. **Open Closed Principle - OCP (Otvoreno zatvoren princip)**
   Entiteti softvera (klase, moduli, metode) trebali bi biti otvoreni za nadogradnju, ali zatvoreni za modifikacije.

Ispod je opisan način kako se Open/Closed Principle (OCP) primjenjuje na svaku od klasa u našem sistemu — s naglaskom na to da su sve one otvorene za proširenje, ali zatvorene za izmjenu postojećeg koda:

- **Klasa `Vožnja`**

  Otvorena za proširenje: Mogu se recimo dodati novi tipovi vožnji (npr. hitna vožnja, grupna vožnja) nasljeđivanjem osnovne klase `Vožnja`

  Zatvorena za izmjene: Osnovna logika naručivanja, prihvatanja vožnje i praćenja kretanja vozača ostaje nepromijenjena

- **Klasa `Putnik`**

  Otvorena za proširenje: Moguće je dodati nove funkcionalnosti za korisnika kao što su "omiljeni vozači", "ponovna narudžba prethodne vožnje", ali bez izmjene postojećih metoda.
  Zatvorena za izmjene: Funkcionalnosti poput `naruciVoznju()`, `ocijeniVoznju()`, `posaljiPoruku()` ostaju nepromijenjene.

- **Klasa `Vozač`**

  Otvorena za proširenje: Moguće je proširiti sistem da podrži različite statuse vozača (npr. zauzet, na odmoru, van mreže) kroz dodatne metode, bez

mijenjanja osnovne logike već prisutnih.
Zatvorena za izmjene: Postojeće funkcionalnosti nrp. `prihvatiVoznju()`, `posaljiPoruku()` ostaju neizmjenjene.

- **Klasa `TehničkaPodrška`**

  Otvorena za proširenje: Može se dodati novi način komunikacije izmedju korisnika i tehničke podrške (npr. poziv uživo) koristeći interfejs npr. `IPodrska`.
  Zatvorena za izmjene: Ne treba mijenjati postojeći način kako je implementirana, tj. Komunikacija sa korisnikom slanjem poruka.

- **Klasa `Ruta`**

  Otvorena za proširenje: Moguće je dodati različite algoritme za izračunavanje rute (najbrža, najkraća, najjeftinija), koristeći strategijski obrazac. Također, moguće bi bilo uračunati stanje u saobrćaju prilikom izračuna.
  Zatvorena za izmjene: Osnovna logika za izračunavanje procijenjenog vremena dolaska ostaje stabilna.

- **Klasa `Lokacija`**

  Otvorena za proširenje: Klasa omogućava proširenje dodavanjem alternativnih izvora podataka o lokaciji (npr. mobilna mreža), ili unapređenja poput korištenja naprednih servisa Google Maps API-ja za preciznije pozicioniranje i praćenje u stvarnom vremenu.
  Zatvorena za izmjene: Osnovna logika za praćenje kretanja vozača ostaje stabilna i ne mora se mijenjati prilikom dodavanja novih izvora ili načina određivanja lokacije.

- **Klasa `Notifikacija`**

  Otvorena za proširenje: Različite vrste notifikacija (push, SMS, email) mogu se implementirati kroz zajednički interfejs `INotifikacija`, bez izmjene postojeće logike.
  Zatvorena za izmjene: Ne treba mijenjati baznu klasu da bi dodao novu vrstu obavještenja.

- **Klasa `Ocjena`**

  Otvorena za proširenje: Mogu se uvesti različiti kriteriji za ocjenjivanje (vozač, vozilo, tačnost), proširujući klasu.
  Zatvorena za izmjene: Postojeće metode za unos i prikaz ocjene ostaju iste.

3. **Liskov Substitution Principle-LSP (Liskov princip zamjene)**
   Podtipovi moraju biti zamjenjivi njihovim osnovnim tipovima.

U analizi sistema i kreiranju dijagrama,a klasa nasljeđivanje je imalo smisla izvršiti jedino uvođenjem apstraktne bazne klase Korisnik, koja čuva zajedničke atribute za klase Vozac, Putnik, TehničkaPodrška i Administrator. LSP princip jeste zadovoljen, jer se u svakom kontekstu gdje se očekuje primjerak bazne može iskoristiti primjerak bilo kojenod izvedenih klasa. Za ostale klase u sistemu nasljeđivanje nije imalo smisla koristiti jer su sve klase jasno definisane i specijalizovane za jednu svrhu, u skladu s principom pojedinačne odgovornosti (SRP). Nijedna klasa ne dijeli dovoljno zajedničkog ponašanja s drugima da bi opravdala nasljeđivanje.

4. **Interface Segregation Principle - ISP (princip izolacije interfejsa)**
   Klijenti ne treba da ovise o metodama koje neće upotrebljavati.

U implementaciji sistema zadovoljen je princip izolacije interfejsa, jer su klase dizajnirane tako da implementiraju samo one metode koje su im zaista potrebne. Jedini interfejs koji ima smisla u ovom kontekstu jeste interfejs za komunikaciju (`IKomunikacija`), jer funkcionalnost slanja poruka koristi više međusobno nezavisnih klasa: `Putnik`, `Vozač` i `TehničkaPodrška`. Na taj način, ove klase ne moraju zavisiti od drugih metoda koje ne koriste, čime se ostvaruje jednostavna struktura. S druge strane, funkcionalnosti poput naručivanja, prihvatanja vožnje, ocjenjivanja ili praćenja kretanja vozača su specifične za pojedine klase, te bi uvođenje posebnih interfejsa za te operacije bilo nepotrebno i suprotno duhu ISP principa, jer bi prisiljavalo klase da implementiraju metode koje ne koriste.

5. **Dependency Inversion Principle - DIP (princip inverzije ovisnosti)**
   Moduli visokog nivoa ne bi trebali zavisiti od modula niskog nivoa. Oba modula trebaju zavisiti od apstrakcija. Moduli ne bi trebali zavisiti od detalja, već detalji trebaju zavisiti od apstrakcija.

Iako je Dependency Inversion Principle koristan princip koji omogućava veću fleksibilnost i manje povezanosti između modula u sistemu, u projektu nismo mogli pronaći mnogo prilika za njegovu primjenu. Razlog tome je što većina klasa ima jasno definisane odgovornosti i zavisi od konkretnih implementacija koje su vezane za njene funkcionalnosti, kao što su slanje poruka, praćenje lokacije i izračunavanje ruta. Ovaj princip je djelimično zadovoljen uvođenjem apstraktne klase Korisnik, iz koje nasljeđuju konkretne klase kao što su Putnik, Vozač, TehničkaPodrška i Administrator. Time se omogućava da ostale komponente sistema zavise od apstrakcije Korisnik, a ne od konkretnih implementacija.