SOLID PRINCIPI

1. Single responsibility principle (SRP)

- *Klasa bi trebala imati samo jedan razlog za promjenu.*

Svaka klasa ima jasno definisanu odgovornost. Na primjer klase imaju samo jednu odgovornost:

-klasa Vozilo se bavi isključivo informacijama o vozilima (marka, model, cijena, dostupnost) i funkcijama poput jeDostupno() i toString()

-klasa Rezervacija upravlja rezervacijama – datumima, statusom i cijenom, i sadrži metode izracunajCijenu(), otkaziRezervaciju() itd

-klasa Plaćanje se brine samo o plaćanjima i verifikaciji kartica

2. Open Closed Principal (OCP)

- *Entiteti softvera (klase, moduli, funkcije) trebali bi biti otvoreni za nadogradnju, ali zatvoreni za modifikaciju.*

Klase su dizajnirane tako da ih je moguće proširiti bez izmjena postojećeg koda.

Enumeracije kao što su StatusRezervacije, StatusPlacanja, Dostupnost i StatusZahtjeva omogućavaju proširivanje statusa bez izmjene postojećih klasa.

Dodavanje novih tipova statusa (npr. novi status za plaćanje) ne zahtijeva promjenu logike u klasama koje ih koriste.

3. Liskov Substitution Principle (LSP)

- *Podtipovi moraju biti zamjenjivi njihovim osnovnim tipovima.*

Hijerarhija klasa je jednostavna i dosljedna, što omogućava da se objekti podklasa mogu koristiti gdje god se očekuje objekat nadklase (Osoba može imati više uloga).

Klasa Osoba koristi enumeraciju Uloga koja može biti registrovani korisnik, neregistrovani korisnik, administrator . Svaka uloga se može tretirati na isti način pri radu s objektom tipa Osoba, bez potrebe za provjerom tipa, čime se poštuje Liskov princip.

4. Interface Segregation Principle (ISP)

- *Klijenti ne treba da ovise o metodama koje neće upotrebljivati.*

Odgovornosti su podijeljene kroz metode specifične za svaku klasu, izbjegavajući nepotrebno opterećenje klasama nebitnim funkcijama.

Klasa Podrška sadrži samo metode vezane za korisničku podršku (posaljiNotifikaciju(), evidentirajInterakciju()), bez dodatnih metoda koje joj nisu potrebne.

Nema nepotrebnog „nasljeđivanja funkcionalnosti“ iz klasa koje nemaju direktne veze s njenom odgovornošću.

5. Dependency Inversion Principle (DIP)

- *Moduli visokog nivoa ne bi trebali ovisiti od modula niskog nivoa. Oba bi trebalo da ovise od apstrakcija.*
- *Moduli ne bi trebali ovisiti od detalja. Detalji bi trebali biti ovisni od apstrakcija.*

Komunikacija među klasama vrši se kroz apstrakcije kao što su enumeracije i jasno definirane metode, a ne kroz direktnu zavisnost od konkretnih implementacija.

Klasa Osoba ne zavisi direktno od implementacije rezervacije ili plaćanja, već koristi metode kao rezervisiVozilo() i platiOnline(), koje kao argumente primaju apstraktne podatke (objekte tipa Vozilo, broj kartice itd.).