SOLID PRINCIPI

SOLID principi predstavljaju skup od pet osnovnih pravila objektno orijentisanog dizajna koji pomažu u kreiranju softverskih sistema koji su lako proširivi, održivi i fleksibilni. U nastavku su objašnjeni svi SOLID principi uz konkretne primjere njihove primjene u okviru dizajna našeg sistema.

1. Princip pojedinačne odgovornosti (SRP - Single Responsibility Principle)

"Svaka klasa treba da ima samo jedan razlog za promjenu."

Ovaj princip nalaže da svaka klasa treba imati tačno jednu odgovornost, odnosno da bude fokusirana samo na jednu funkcionalnost sistema. Time se postiže bolja modularnost i lakše održavanje koda.

- Klasa Korisnik je odgovorna isključivo za upravljanje podacima korisnika.
- Klasa Proizvod rukuje podacima o proizvodima.
- Klasa Apoteka se bavi podacima vezanim za apoteku.
- Ostale klase poput Korpa, ObradaNarudžbe, NarudžbaProizvoda, StavkaNarudžbe, i E-račun imaju jasno definisane i jedinstvene funkcionalnosti u sistemu.

Na ovaj način, svaka klasa ima samo jednu funkcionalnost, što je u potpunosti u skladu sa SRP principom.

2. Otvoreno-zatvoreni princip (OCP - Open/Closed Principle)

"Softverski entiteti trebaju biti otvoreni za proširenje, ali zatvoreni za izmjene."

To znači da bi trebalo moći nadograđivati funkcionalnost klase bez potrebe da se mijenja njen postojeći kod. Time se smanjuje rizik od grešaka u postojećem funkcionalnom dijelu sistema. Korištenjem enumeracija i posebnih klasa poput TipKorisnika i KategorijaProizvoda, omogućena je jednostavna nadogradnja (dodavanje novih tipova korisnika ili kategorija) bez mijenjanja postojećeg koda. Time se osigurava stabilnost sistema i istovremeno omogućava njegova buduća proširivost.

3. Liskov princip zamjene (LSP - Liskov Substitution Principle)

"Objekti podtipova moraju se moći zamijeniti objektima bazne klase bez narušavanja tačnosti sistema."

Ovaj princip se odnosi na pravilno korištenje nasljeđivanja. Podklase moraju zadržati ponašanje osnovne klase i moraju moći da je zamijene u bilo kojem kontekstu. Iako se trenutno ne koristi opsežna hijerarhija, dizajn sistema omogućava buduće proširenje klase Korisnik putem njenih podtipova. U svakom dijelu sistema gdje se očekuje objekat klase Korisnik, moguće je koristiti bilo koji od njenih podtipova bez narušavanja funkcionalnosti.

Ovim je zadovoljen LSP princip, dok su ostale klase ostale jednostavne i fokusirane, u skladu sa SRP principom.

4. Princip segregacije interfejsa (ISP - Interface Segregation Principle)

"Klijenti ne bi trebali biti prisiljeni da zavise od metoda koje ne koriste."

Cilj ovog principa je da klase implementiraju samo one metode koje su im stvarno potrebne, čime se izbjegava suvišna funkcionalnost i nepotrebna zavisnost. Klase su dizajnirane tako da sadrže samo one metode koje su im potrebne za njihovu specifičnu funkcionalnost. Npr, samo klasa E-račun sadrži i implementira metode za slanje e-maila, dok ostale klase nisu opterećene ovom funkcionalnošću. Time se korisnici interfejsa štite od metoda koje im nisu potrebne, što je u skladu sa ISP principom.

5. Princip inverzije zavisnosti (DIP - Dependency Inversion Principle)

"Moduli višeg nivoa ne bi smjeli zavisiti od modula nižeg nivoa; oboje bi trebali zavisiti od apstrakcija."

Ovaj princip omogućava veću fleksibilnost i zamjenjivost dijelova sistema tako što promoviše upotrebu apstrakcija umjesto konkretnih implementacija. Ovaj princip zadovoljavamo implemetacijom klase Korisnik koja funkcioniše kao apstrakcija nad podtipovima.

Zaključak

Dizajn sistema uspješno prati svih pet SOLID principa objektno orijentisanog dizajna.
Klase su jasno strukturirane, sa jasno definisanim odgovornostima, međusobno povezane preko apstrakcija gdje god je to bilo moguće.