Primjena SOLID principa u dizajnu klasa

📌 Uvod

Model sistema je dizajniran tako da podrži sve ključne funkcionalnosti fitness platforme: upravljanje korisnicima, generisanje plana ishrane i treninga, praćenje napretka, upravljanje terminima i plaćanjem. Kroz pažljivo razdvajanje odgovornosti i povezivanje entiteta preko jasnih veza, osigurana je usklađenost sa svih pet SOLID principa objektno-orijentisanog dizajna.

1. S — Single Responsibility Principle (SRP)

Svaka klasa ima jednu jasno definisanu odgovornost.
- Korisnik se bavi ličnim podacima i ciljevima korisnika.
- Trener predstavlja profesionalni profil trenera.
- PlanIshraneITreninga sadrži informacije o prehrambenom i trening planu.
- Termin služi za rezervaciju termina između korisnika i trenera.
- Plaćanje upravlja transakcijama i statusima popusta.
- StatistikaNapretka omogućava praćenje BMI-ja, težine i kalorijskog unosa.

🟢 Zadovoljeno: nema klase koja preuzima više odgovornosti ili miješa uloge.

2. O — Open/Closed Principle (OCP)

Klase su otvorene za proširenje, ali zatvorene za izmjene.
- Nove funkcionalnosti mogu se dodavati npr. nasljeđivanjem Korisnik za različite tipove korisnika.
- enum vrijednosti (npr. Spol, Cilj, NacinPlacanja) omogućavaju proširenje bez izmjene same klase.
- Novi tipovi plaćanja ili ciljeva mogu se dodati bez mijenjanja postojećih klasa.

🟢 Zadovoljeno: proširenja su moguća bez mijenjanja osnovnog koda.

3. L — Liskov Substitution Principle (LSP)

Objekti izvedenih klasa mogu zamijeniti objekte bazne klase bez narušavanja funkcionalnosti.
- Ako bismo naslijedili Korisnik u klase Student, Gost, svi bi se mogli koristiti na mjestu baznog Korisnik bez problema.
- Tipovi se jasno poštuju, npr. Trener ne nasljeđuje Korisnik, što bi narušilo LSP.

🟢 Zadovoljeno: model podržava buduće podklase bez narušavanja ponašanja.

4. I — Interface Segregation Principle (ISP)

Klijenti ne bi trebali biti prisiljeni implementirati interfejse koje ne koriste.
- Iako interfejsi nisu eksplicitno definisani (jer radimo na nivou UML modela), dizajn implicira da npr.:
  - Korisnik koristi metode za upravljanje ličnim podacima.
  - Trener koristi metode za upravljanje terminima i planovima.
- Funkcionalnosti nisu zbijene u jednu "sveobuhvatnu" klasu.

🟢 Zadovoljeno: klasama su dodijeljene samo funkcionalnosti koje su im relevantne.

5. D — Dependency Inversion Principle (DIP)

Aplikacija treba zavisiti od apstrakcija, a ne od konkretnih implementacija.
- Zavisan kod (npr. funkcionalnosti koje kreiraju planove, obrađuju statistike itd.) može raditi prema interfejsima/servisima, a ne direktno nad klasama.
- Veze između klasa ostvarene su preko primarnih i stranih ključeva, čime se omogućava fleksibilno povezivanje entiteta bez čvrstog spajanja.

🟢 Zadovoljeno: model omogućava odvajanje logike od strukture, čime podržava DIP.

📌 Zaključak

Dizajn sistema u potpunosti poštuje SOLID principe:
- svaka klasa ima svoju ulogu,
- sistem se može lako proširivati bez izmjena,
- jasno su definisane granice odgovornosti,
- pripremljen je za buduće proširenje interfejsima i servisima.

Takav dizajn omogućava skalabilnost, jednostavno održavanje i povećava dugoročnu stabilnost sistema.