Solid principi

1. Single Responsibility Principle (SRP)

Sve klase zadovoljavaju navedeni princip.
Princip je ispunjen jer svaka klasa u sistemu ima **tačno jednu odgovornost**:

- `Korisnik` čuva osnovne informacije o korisnicima sistema.
- `Donacija` predstavlja konkretnu transakciju pomoći.
- `ZahtjevZaPomoc` opisuje potrebu korisnika za pomoći.
- `Notifikacija` predstavlja obavijest koju korisnik prima.
- `Akcija` opisuje volonterske aktivnosti.
- `VolonterAkcija` služi kao međutabela između korisnika i akcija.

Svaka klasa se bavi isključivo podacima koji su vezani za njen kontekst, što znači da postoji **jedan razlog za promjenu** u svakoj klasi.

---

2. Open-Closed Principle (OCP)

Sve klase zadovoljavaju navedeni princip.
Princip je ispunjen jer je sistem projektovan tako da je **otvoren za proširenje, ali zatvoren za izmjene**.
Na primjer:

- Ako se želi dodati novi tip notifikacije (npr. push notifikacija), moguće je to uraditi proširenjem postojeće strukture kroz implementaciju interfejsa `INotifikacijaService`, bez izmjene postojećih klasa.

- Novi tip korisnika (npr. `Admin`, `Volonter`, `Donator`) može se kreirati kao podklasa `Korisnik`, čime ne dolazi do izmjene postojeće logike.

---

3. Liskov Substitution Principle (LSP)

Sve klase zadovoljavaju navedeni princip.
Ukoliko se uvedu naslijeđene klase iz `Korisnik` (npr. `Donator`, `Volonter`), one mogu **bez problema biti korištene na mjestu gdje se očekuje osnovna klasa**, bez narušavanja funkcionalnosti.
Npr. metoda `posaljiNotifikaciju(Korisnik k)` će raditi isto bez obzira da li je korisnik instanca osnovne klase ili podklase.

---

4. Interface Segregation Principle (ISP)

Model zadovoljava navedeni princip.
Iako trenutno model ne koristi interfejse eksplicitno, pri eventualnom implementiranju servisnog sloja preporučeno je kreirati više **specijalizovanih interfejsa** (npr. `IDonacijaService`, `INotifikacijaService`, `IAkcijaService`) umjesto jednog velikog servisnog interfejsa.
Tako će klase implementirati **samo ono što im je potrebno**, bez suvišnih metoda.

---

5. Dependency Inversion Principle (DIP)

Sistem zadovoljava navedeni princip.

Kod planiranja sloja logike i servisa, klase kao što su `Izvjestaj` ili `Notifikacija` **trebaju zavisiti od apstrakcija**, a ne konkretnih implementacija.

Na primjer, `IzvjestajGenerator` može zavisiti o interfejsu `IDonacijaRepozitorij` umjesto o konkretnoj bazi podataka.

Na ovaj način omogućeno je **lako testiranje**, zamjena izvora podataka i lakše održavanje.