

SOLID principi

Single Responsibility Principle – ovaj princip je ispoštovan zato što je svaka klasa (uključujući kontrolere) odgovorna samo za jednu stvar, odnosno upravlja samo jednom stvari (view-om). Npr. `RezervacijaController` je zadužena isključivo za `RezervacijaView`, a i `RezervacijaView` i `RezervacijaController` se bave samo rezervacijama (ne bave se mijenjanjem postavki accounta). Korisnik je prethodno u sebi imao atribut tipa `int vrijemeIstekaRezervacije`, ali sada je to izbačeno, jer se vremenom isteka rezervacije treba baviti samo klasa `Rezervacija` zato što to ne opisuje korisnika.

Open Close Principle – iz prethodno navedenih razloga u SRP može se vidjeti da je i ovaj princip ispoštovan. Ukoliko promijenimo jednu klasu, to neće dovesti do promjene druge klase. Npr. funkcija `spasiPromjene` u `EditAccountController`u poziva funkcije `provjeriIme` samo jednom i `provjereLozinke` samo jednom, a nijednu od njihovih funkcionalnosti ne radi sama, tj. ukoliko se promijeni način provjere imena ili provjere lozinke, ova funkcija ostaje ista. Isto tako funkcija `promijeniZauzetostParkinga` poziva funkciju `pratiVrijemeSvakogParkinga` te postupa u skladu sa tom funkcijom, ali za ovu funkciju nije bitno kako `pratiVrijemeSvakogParkinga` provjerava vrijeme (Close dio principa). Svaka od klasa je krajnje jednostavna sa veoma malom međuovisnosti kada su u pitanju funkcije tako da je vrlo lahko dodati nove funkcionalnosti bez većih izmjena (Open dio principa).

Liskov Substitution Principle – ovaj princip je ispoštovan jer u ovoj aplikaciji ne postoji vrsta nasljeđivanja gdje bi se bazna klasa trebala moći koristiti gdje i izvedena. Štaviše, postoji samo jedna izvedena klasa, a to je klasa `RegistrovaniKorisnik` izvedena iz `Korisnik`, pri čemu klasa `Korisnik` se ne smije moći koristiti kao `RegistrovaniKorisnik`, jer nije svaki korisnik registrovan te samim tim nema pravo na popuste i nema potrebe pratiti njegovo uzastopno vrijeme na parkingu.

Interface Segregation Principle – ne koriste se interfaces u aplikaciji. Da postoji više podvrsta `Korisnika` i da npr. `RegistrovaniKorisnik` treba neku metodu koju ostale podklase klase `Korisnik` ne trebaju, bilo bi najbolje napraviti dodatni interface da na taj način ostale podklase ostanu nedirnute.

Dependancy Inversion Principle – s obzirom da u aplikaciji nema tolike kompleksnosti da bi se nešto moglo nazvati apstrakcijom pa da zavisi od detalje i s obzirom da se ne mogu primjetiti high-level i low-level moduli, ovaj princip je zadovoljen.