SOLID principi

Single Responsibilty 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 mijenjam 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 Prinicple – 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. funckija spasiPromjene u EditAccountControlleru 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 postoji samo jedna vrsta nasljeđivanja, a to je iz RegistrovaniKorisnik iz Korisnik. S obzirom da su jedini atributi Korisnik klase ime,prezime,email i jedine metode su odgovarajuci getteri i setteri za te atribute koje svakako ima svaki RegistrovaniKorisnik, klasa RegistrovaniKorisni se moze koristiti svugdje gdje se koristi i Korisnik.

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

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