# projektni zadatak 21.5:

## O klasnom dijagramu:

Odlučili smo iskoristiti singleton patern u našem klasnom dijagramu, te preko njega uvesti rad sa bazom podataka. Da bi dohvatali informacije iz baze ili upisivali informacije u bazu podataka, moramo ostaviti neki vid konekcije sa njom koristeći kod. Ukoliko ne bi učinili ovu klasu singletonom, mogli bismo pokušati napraviti više instanci klase baze, te pokušati ostvariti više konekcija na bazu što bi dovelo do kraha programa. S druge strane, i da je dopušteno napraviti dvije konekcije na bazu podataka, u slučaju da radimo na nekoliko threadova, mogli bismo pokušati pristupati istim podacima u bazi kroz dvije instance, te tako dovesti do konflikta, prijevremenih brisanja podataka i slično.

Da bi ispoštovali MVC pattern, u naš kalsni dijagram dodali smo kalse koje predstavljaju View-ove koji će u aplikaciji biti realizovani kroz pravljenje različitih vidova HTML (i vjerovatno css fajlova), te klase za Controller-e. Svaki kontroler u sebi ima instancu klase kaze, te ima konstruktor koji prihvata element tipa klase koja je zaduzena sa rad sa bazom, te na taj način ćemo samo prosljeđivati instancu baze koja je jedinstevena u svim kontrolerima. Izuzetak je klasa za kontrolor pri prijavi, s obzirom da se radi o kontroleru koji će se pri pokretanju aplikacije aktivirati, u njemu smo dodali i kontroler bez parametara.

## SOLID :

1) **Single responsiblity principle**: ✅

   - Princip je zadovoljen jer sve klase obavljaju jednu dužnost.

2) **Open-closed principle**: ✅

   - Ovaj princip je zadovoljen, jer kod je podložan nadogradnjama, bez potrebe da se kod značajno modifikuje. Uzmimo za primjer klasu “Objekat”. Iz ove klase su izvedene klase “Stan” i “Kuca”, te je ostavljeno dodatnog prostora za daljnja proširenja kao što su vikendice, hotelske sobe itd.

3) **Liskov substitution principle** ✅

   - Zahtjev da je nasljeđivanje odrađeno na način da se na svim mjestima na kojim se koriste instance osnovnih objekata mogu koristiti i instance izvedenih objekata. Uzmimo za primjer korištenje objekata tipa “Kuca” i “Stan” koji su nasljeđeni iz klase “Objekat”. Na svim mjestima se objekti ovog tipa mogu korisiti kao i sami objekti, tj. nasljeđivanje je ispravno implementirano.

4) **Interface segregation principle** ✅

   - Ovaj princip je također zadovoljen s obzirom na to da su interfejsi u našem kodu odvojeni tako da obavljaju zasebne logičke cjeline koje nisu prekompleksne, te zbog njih nije kod postao manje razumljiv. U interfejse su razdvojene samo

cjeline kao što su vrste pretraga razdvojene u jedan interfejs, ili npr korištenje interfejsa u svrhu provjere pristupa pojedinim podacima u zavisnosti od korisnika koji traži neke informacije. Također, iako smo bazu mogli realizovati u obliku interfejsa, odlučili smo iskoristiti pogodnost i napraviti DAO klasu, tj. Data Access Object klasu koja je zadužena za konekciju na bazu te svu komunikaciju s njom.

5) **Dependency inversion principle** ☑

- Princip je zadovoljen jer su bazne klase realizovane tako da budu apstraktne, tj. da se ne mogu instancirati objekti njihovog tipa, te dubina nasljeđivanja nije dublja od dva.