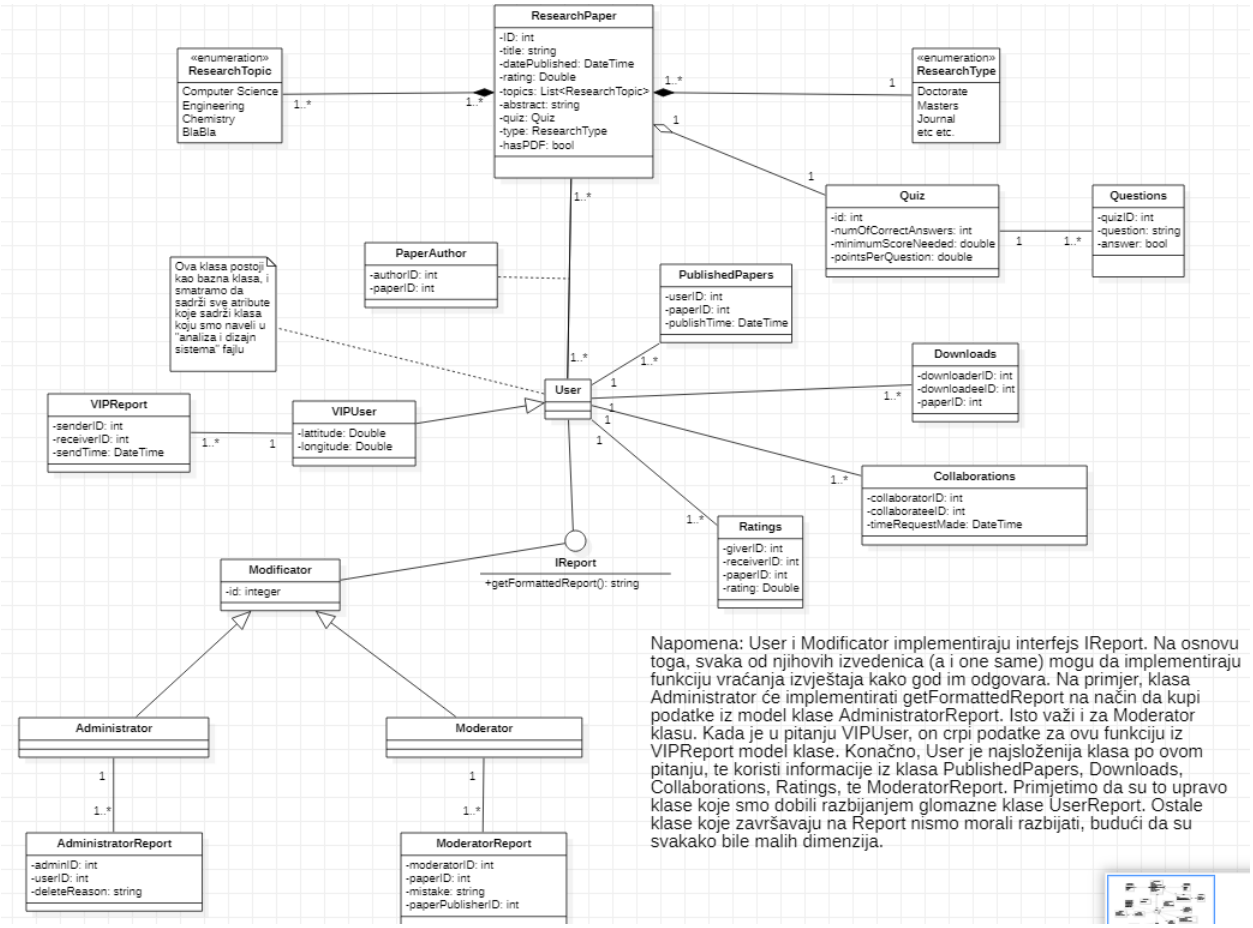
DIJAGRAM KLASA



Jedino još nisam postavio klasu za bazu podataka, jer bi bilo neuredno (ona treba biti u vezi asocijacije sa svim klasama koje su model klase). Također, ne znam da li je pametno napraviti hijerarhiju u radu sa klasama baze podataka (npr imamo baznu klasu Database, i onda izvodimo iz nje UserDB, RatingsDB, itd…) gdje će svaka od klasa biti zadužena za upite i rad sa odgovarajućim istoimenim klasama, dok bi glavna Database klasa bila odgovorna samo za uspostavljanje konekcije. Što se tiče ovog pitanja ako Vam nije problem ostavite odgovor u komentare, ili na google chat.

Poštivanje SOLID principa

Proći ćemo kroz svaki od principa ukratko:

- Single responsibility principle: Kao što se da primjetiti, sve klase su relativno male po svom broju atributa, što je dobar indikator da svaka od njih ima po jednu odgovornost. Naime, najveća klasa je ResearchPaper, ali je većina atributa informativne prirode, a ne funkcionalne. U odnosu na Analizu i Dizajn sistema

DIJAGRAM KLASA

(task), gdje smo imali dosta velike klase (pogotovo klasa UserReport), sada smo ih razbili na manje klase (kao što su Downloads, PublishedPapers, Collaborators), kako zbog SRP principa, tako i zbog baze podataka.

- Open/Closed princip: Njega koliko vidim i nemamo gdje sačuvati, ali recimo ukoliko želimo uvesti novi tip korisnika u sistem, dovoljno je da se naslijedi User (ili ako to ne želimo ne moramo, možemo samo naslijediti IReport), i tada smo u mogućnosti da sa nekim novim klasama dodanim za funkcionalnosti tog novog user-a, recimo realiziramo funkciju getFormattedReport(), čime ćemo bez mijenjanja starih stvari (samo nadogradnja, dodavanje novih klasa i nasljeđivanja) ostvariti upgrade sistema, u ovom primjeru sa novim tipom korisnika.

- Liskov princip zamjene: ovo možemo testirati na nasljeđivanjima, budući da se ovaj princip odnosi na ispitivanje pravilnog nasljeđivanja. Kod Modificator-a, i Administrator i Moderator zaista jesu jedna vrsta (specijalni tip) Modifictor-a (onoga koji mijenja sistem) sistema. Bilo koja funkcija koju obavlja Modificator, može je obavljati i Administrator/Moderator. Kada je u pitanju User i VIPUser, očito da je VIPUser samo specijalna vrsta običnog User-a. On ima sve kao i obični user, ali ima i još par dodatno omogućenih funkcionalnosti zahvaljujući svome statusu.

- Interface segregation princip: Ovdje se radi o tome da neki klijent u vidu klase ne treba da implementira interfejs kog neće koristiti. Kod nas ima samo jedan interfejs (IReport), i on ima samo jednu funkciju, i ta funkcija se koristi od strane svih korisnika sistema (VIPUser, User), kao i od strane svih Modifikatora sistema, tako da ovdje ne treba razdvajati ništa.

- Dependency inversion principle: ovo pravilo se ogleda u tome da bazne klase trebaju da budu apstraktne. Ovo sam ostvario kod Modificator-a sistema, on je apstraktni, i bazni za Administratora i Moderatora. Međutim, u slučaju User-a je stvar malo komplikovanija. Naime, prvobitno sam i naumio da User bude apstraktna klasa, koju će naslijeđivati GuestUser, a kojeg će opet naslijediti VIP korisnik. Međutim, ispostavlja se da GuestUser nema nikakvih konkretnih funkcionalnosti (sve se svodi na browsing, i pregled stranice), tako da je on ispao

DIJAGRAM KLASA

iz igre. Prijavljeni korisnik (User) se ne može napraviti apstraktnim, jer ima neke stvari koje mora obaviti (mora se instancirati), dok pravljenje neke apstraktne klase AbstractUser ne bi imalo puno smisla, jer bismo imali nasljeđivanje na tri nivoa: AbstractUser -> User -> VIPUser, pri čemu AbstractUser ne bi imao nikakvu konkretnu svrhu. Ukoliko griješim ispravite me, i na google chat ili ovdje u komentare ostavite neku ideju ukoliko smatrate da ovaj princip (ili bilo šta drugo) može biti zadovoljen na neki drugi, bolji način.