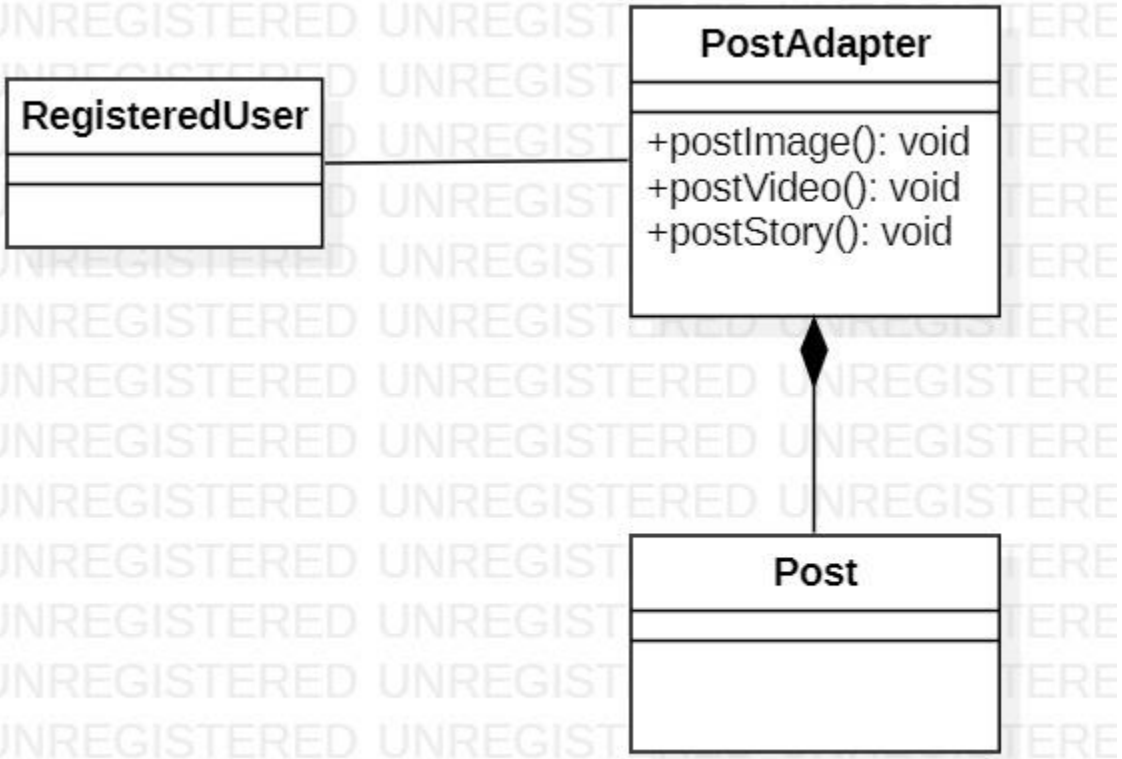
STRUKTURALNI PATERNI

1. Adapter pattern

Adapter pattern se koristi s ciljem da omogući širu upotrebu već postojećih klasa. Koristimo ga kada je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati već postojeću klasu. Adapter služi da se objekti prilagode nekim specifičnim upotrebama. Moguće je kreirati adapter klasu, koja služi kao posrednik između orginalne klase i željenog interfejsa.

U našem sistemu moguće je koristiti adapter pattern prilikom objavljivanja sadržaja. Adapter bi omogućio objavljivanje sadržaja nevezano bio taj sadržaj slika, tekst, videozapis ili 24-satna priča. Adapter bi omogućio da se prilikom objavljivanja slike omogući objava iz različitih formata(jpg, png...).



2. Facade pattern

Facade pattern olakšava korisniku korištenje sistema. Poenta je skrivanje struktura. Korisnik koristi gotov objekat, dok mu njegova struktura ostaje nepoznata. Dakle, on može biti korisnik bez da je detaljno upoznat o tome kako sistem funkcioniše.

Pattern je primjenjen u našem sistemu na više mjesta. Korisnik može koristiti sistem bez da je u potpunosti upoznat sa njim. U pozadini sistema kreira se profil, statistika, notifikacija, objava...

3. Dekorater pattern

Dekorater pattern služi da olakša različite nadogradnje objekata koje imaju istu osnovu. Umjesto povećavanja broja klasa, dovoljno je omogućiti različito dekodiranje objekata. Akcenat je na korištenju već postojećih klasa, a ne praviti nove.

Dekorater pattern bi se mogao postići samim tim da se korisniku omogući uređivanje profila na različite načine, kao na primjer promjena slike profila, nekih ličnih podataka...

4. Bridge pattern

Omogućava da se iste operacije primjenjuju nad različitim podklasama. Bridge pattern se postiže samim izbjegavanjem kreiranja novih metoda za postojeće funkcionalnosti. Upotrebom ovog patterna omogućava se da se apstrakcija nekog objekta odvoji od implementacije.

Možemo postići ovaj pattern time što u klasi RegisteredUser pišemo metode koje će i RegisteredUserPersonalProfile i RegisteredUserProfessionalProfile moći koristiti. Time ćemo omogućiti da se koristi ta metoda, a nećemo praviti nove metode za iste funkcionalnosti.

5. Composite pattern

Ovaj pattern služi za kreiranje hijerarhije klase. Bazira se na pozivanju iste metode nad različitim objektima sa različitim implementacijama.

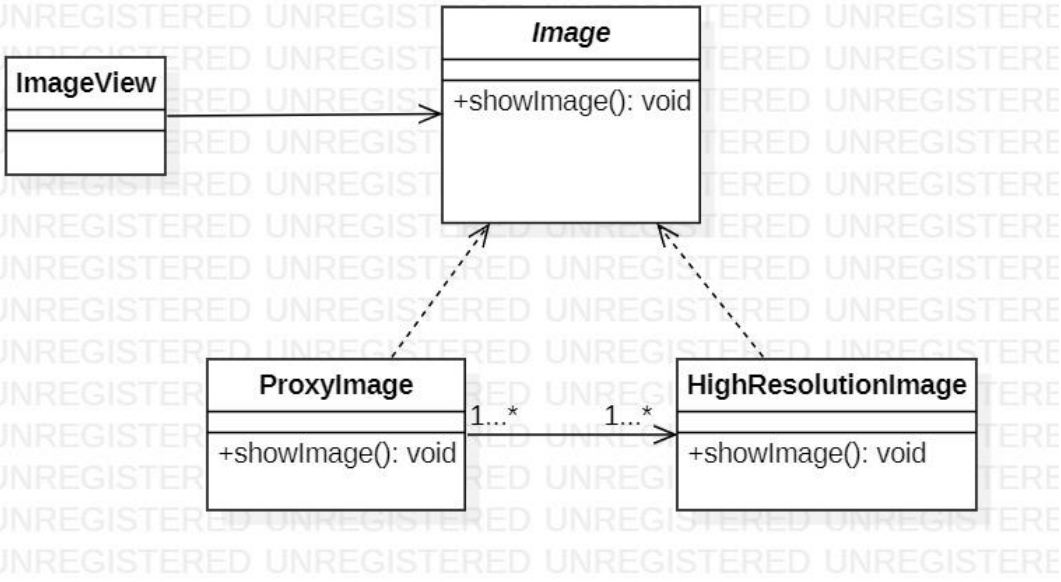
Ovaj pattern možemo iskoristiti na način da proširimo sistem. Recimo da uvedemo još jednu vrstu korisnika koja će imati sličnu funkcionalnost kao administrator, te metode koje služe za rad administratora mogu služiti i za rad drugog tog korisnika, uz neke manje razlike.

6. Proxy pattern

Proxy pattern ima dvije vrste upotrebe:

- U svrhu zaštite,
- U svrhu cachiranja.

Mi ćemo u našem sistemu kreirati Proxy klasu koja će omogućiti povećanje rezolucije slike. Ideja je da pritiskom na sliku dobija sliku sa većom rezolucijom preko Proxy klase.



7. Flyweight pattern

Flyweight pattern se koristi kako bi se onemogućilo bespotrebno stvaranje ogromnog broja instanci objekata koji svi u suštini predstavljaju isti objekta. Dakle, stvara mogućnost ponovne upotrebe objekata da bi se smanjila potrošnja RAM memorije.

Mi u našem projektu bismo mogli koristiti ovaj pattern prilikom kreiranja statistike, na način da se jedna već kreirana statistika samo ažurira tokom vremena, a ne da se pravi nova. Tako bi svaki Post imao samo jednu statistiku koja bi se vremenom mijenjala.