**Dizajn paterni i analiza**

**1. Strukturni dizajn paterni**

1) Adapter pattern

Adapter patern smo mogli iskoristiti kod slika za korisnika i artikal. S obzirom da oba imaju svoja posebna ograničenja veličine i slično implementirani su odvojeno, no mogli smo koristiti isti wrapper objekt slika i uraditi refactoring u adapter patern. Korisnik manager i Registration manager smo također mogli kombinovati koristeći ovaj patern.

2) Fascade patern

Fascade pattern smo mogli koristiti u slučaju kada bi implementirali cijeli sistem prodaje, komunikacije sa bankom i dostave robe(a ne samo mock service) pri čemu bi Fascade predstavljao UI koji skriva interkonekcije ovih sistema od korisnika.

3) Decorator patern

Ovaj patern nije pretežno koristan za našu aplikaciju, s obzirom da nema mnogo mjesta gdje je dinamično ponašanje neophodno, no mogli smo ga koristiti možda npr. u interakciji sa mapama pri čemu bi dodali neke dodatne mogućnosti koje bi zahtjevale određenu dinamiku.

4) Bridge patern

Ponovno, ne posebno koristan model za našu aplikaciju. No, mogli smo ga koristiti na nekoliko mjesta, kao što je klasa Lokacija čiju smo implementaciju mogli varirati zavisno od toga šta nam treba, te Cart koji takođet ima sposobnost iskoristivosti u više različitih obilka(offline, online, itd.)

5) Proxy

Poprilicno beskoristan za aplikaciju. Mogao bi se, mada ne vidimo stvarno svrhu, uraditi refactoring kod informacija o korisniku tako da preko proxy objekta se vrsi promjena tih informacija i slično. Štaviše ovo izgleda kao anti-pattern u slučaju takvog korištenja.

6) Composite patern

Composite patern smo mogli iskoristiti na više mjesta ali ne sa pretežno velikom motivacijom. Mogli smo grupirati korisnike po tipu ili štaviše artikle i iteme po tipu. Budući da ne implementiramo neke složene povezane kolekcije, ovaj patern nije nešto posebno prikladan za upotrebu.

7) Flyweight patern

Ovaj patern ima iznenađujuće solidan potencijal za upotrebu. Naime, svakog od glavnih tipova podataka(Artikal, User, Item, Kreditna Kartica i sl.) možemo refactorati u ovaj patern. Pri tome poprilično zgodna bi bila npr. Kreitna Kartica gdje zajednička polja predstavljaju

**2. Kreacijski dizajn paterni**

1) Singleton patern

Singleton je naroćito primjenjiv na našu aplikaciju jer imamo puno klasa koje rade interkaciju sa bazom ili validaciju i za koje je sasvim dovoljna samo jedna instantacija.

2) Prototype patern

Prototype patern smo mogli koristiti recimo za kloniranje objekata tipa Item, pri čemu bi radili plitko kopiranje. No, vidimo da ovaj patern nije pretjerano interesantan za nas.

3) Factory method patern

Factory je iskoristiv ukoliko bi implementirali dodatne funkcionalnosti u aplikaciju koje se tiču dostave robe i slično. Kako sada stvari stoje, nije pretežno interesantan i nije iskoristiv osim npr. za implementaciju klase slika sa različitim tipovima(iako ovo već graniči anti-patternu).

4) Abstract Factory patern

Prilicno slično kao za prethodni patern, ovaj patern nam nije pretežno interesantan.

5) Builder patern

Builder patern je prilično interesantan i pojedine klase kao što su Korisnik i Artikal bi se mogle implementirati koristeći ovaj patern zbog svoje složenosti. Isto vrijedi i za klase kao što je Item, no njihova implementacija je relativno jednostavna i takva da nema neke potrebe za ovakvim tipom paterna.