Kreacijski paterni

1. Singleton pattern

- *Singleton patern pruža mogućnost da se klasa instacira samo jednom i da osigura globalni pristup kreiranoj instanci klase. Moguće je da postoji više objekata koje je potrebno samo jednom instancirati i nad kojim je potrebna jedinstvena kontrola pristupa.*

- U našem DressCode sistemu, Singleton obrazac se može koristiti za klase koje trebaju imati samo jednu instancu tokom izvršavanja. Na primjer, klasa koja predstavlja administratora odnosno menadžera sistema trebala bi biti jedinstvena kako bi se izbjegle nekonzistentnosti i višestruki pristupi koji mogu dovesti do konflikata. Također, centralizovana sesija korisnika može se implementirati kao Singleton kako bi sve komponente aplikacije imale pristup istim informacijama o trenutnom korisniku.

2. Prototype pattern

- *Prototype patern pruža mogućnost kreiranjaobjekata koristeći neki postojeći prototip postojećeg objekta. Prednost ovog paterna je što se mogu kreirati prilagođeni objekti bez poznavanja detalja kako je objekat kreiran ili njihove klase.Također, još jedna prednost ovog paterna je i što kloniranje već postojećeg objekta olakšava u slučaju da su veliki troškovi za kreiranje novog objekta i/ili je kreiranje objekta resursno zahtjevno.*

- U sistemu kao što je DressCode, veoma je bitno da radnik ima mogućnost korištenja šablona za nove proizvode, mijenjajući samo određene informacije. Umjesto da se svaki atribut proizvoda unosi ispočetka, koristi se kloniranje već postojećeg objekta.

3. Factory method pattern

- *Factory Method Patern pruža mogućnost kreiranja objekata tako da podklase odluče koju klasu instancirati.Patern instancira odgovarajuću izvedenu klasu,tj podklasu na osnovu tekućeg stanja ili od strane klijenta.*

- Factory patern bi bio izuzetno koristan za kreiranje različitih vrsta korisnika na osnovu njihove uloge, poput korisnika, administratora ili radnika. Umjesto da imamo više if-ova ili duplikata koda po cijeloj aplikaciji, Factory omogućava centralizovanu i jednostavno proširivu logiku za instanciranje objekata. Na osnovu ulaznog parametra, vratili bi odgovarajuću instancu klase Korisnik. Ovo čini kod preglednijim, lakšim za održavanje i fleksibilnijim za buduće proširenje.

4. Abstract factory pattern

- *Abstract Patern pruža mogućnost da se kreira familija povezanih objekata i na osnovu toga se kreiraju konkretne fabrike različitih kombinacija i tipova.*

- Abstract Factory bi imao primjenu kada želimo da grupišemo više međusobno povezanih objekata koji zajedno formiraju jedan kontekst. Recimo, za svaku korisničku ulogu, mogli bismo imati odgovarajući interfejs, meni, prikaz narudžbi i druge komponente specifične za tu ulogu. Umjesto da ih instanciramo ručno svuda, Abstract Factory omogućava kreiranje čitavog seta tih povezanih objekata preko jednog interfejsa. Time se olakšava konfiguracija sistema i smanjuje zavisnost među komponentama.

5. Builder pattern

- *Builder patern odvaja specifikacije kompleksnih objekata od njihove stvarne konstrukcije.*

- Builder patern je idealan za kreiranje objekata koji imaju mnogo opcionalnih ili složenih svojstava, poput narudžbi ili korisnika sa detaljnim podacima. Na primjer, kada kreiramo novu narudžbu, možda ne znamo odmah sve informacije kao što su opis, adresa dostave, itd. Builder omogućava postepeno i fleksibilno konstruisanje takvih objekata bez potrebe za beskonačnim konstruktorima ili masivnim parametrima. Takođe, čini kod čitljivijim i olakšava buduće izmjene.