

# KREACIJSKI PATERNI

Postoje 5 vrsta kreacijskih paterna za koje ćemo navesti način implementacije u našem projektu:

1. Singleton
2. Factory Method
3. Abstract Factory
4. Builder
5. Prototype

## 1. Singleton

Singleton pattern osigurava da imamo samo jednu instancu i pruža *globalnu* tačku pristupa toj instanci.

U našem projektu Singleton pattern bi mogli iskoristiti tako da bi kreirali *singleton* klasu za upravljanje prijavljenim korisnikom (UserSessionManager). Ovo je korisno implementirati jer nakon što se korisnik/ici loginuju na aplikaciju sistem u svakom trenutku mora da zna osnovne podatke o korisniku (ko je trenutno prijavljen, koja je uloga (admin ili obični korisnik), koje dozvole ima itd. ) a te informacije su potrebne na više mjesta u aplikaciji.

Dakle *Singleton* klasa bi na početku dobavila podatke o korisniku i na taj način spriječila višestruko dohvaćanje informacija o korisniku od strane drugih dijelova programa.

## 2. Abstract Factory

Abstract Factory pattern nam omogućava kreiranje povezanih objekata bez navođenja njihovih konkretnih klasa.

U našem projektu Abstract Factory pattern koristili bi tako što bi kreirali apstraktnu klasu *Delivery Factory* koja definiira porodicu objekata za dostavu. S obzirom da imamo više načina dostave dokumenata (digitalno putem maila, ličnim preuzimanjem, dostavom putem pošte) definirali bi respektivne klase za svaki vid dostave, a svaka bi naslijedila *Delivery Factory*.

### 3. Builder

Builder pattern omogućava postepenu konstrukciju složenijih objekata. U našem projektu *Builder pattern* bi mogli iskoristiti pri inicijalnom kreiranju korisnika. Naime, korisnik ima dosta atributa od kojih su neki složenijeg tipa i to bi mogli segmentirati kako bi samo kreiranje bilo dosta jednostavnije.

Međutim, kreiranje korisničkog naloga u našem projektu se dešava isključivo uz fizičko prisustvo nadležne osobe pa se iz tog razloga nismo odlučili za ovaj pattern.

### 4. Factory Method

*Factory Method* pattern omogućava definisanje interfejsa za kreiranje objekta tako da podklase mogu da odluče koji objekat da instanciraju.

U našem projektu bi napravili klasu *ZahtjevFactory* koja bi se koristila za instanciranje objekata/zahtjeva. Ovo nam omogućava veliku skalabilnost prilikom potencijalnih dodavanja novih zahtjeva. Također, ovime postizemo centraliziranje prilikom odlučivanja o kojem tipu zahtjeva je riječ.

### 5. Prototype

Prototype pattern koristimo kada nam je potrebno kloniranje objekata. To se obično radi kada je novo instanciranje izuzetno skupo.

U našem projektu Prototype pattern koristili bi prilikom kreiranja novih šablona od strane administratora. S obzirom da se šabloni često uveliko preklapaju smatramo da je ovaj pattern idealan za ovu svrhu. Dakle svaki šablon bi bio implementiran sa mogućnošću da se klonira i taj klon administrator može bez bilo kakvih problema da modifikuje po potrebi. Ovim načinom implementacije želimo prvenstveno da održimo konzistentnost u izgledu šablona, uštedimo vrijeme prilikom dodavanja novih šablona te enkapsuliramo logiku kreiranja.

U okviru našeg projekta odlučili smo se za korištenje *Prototype* i *Factory Method* patterna jer smatramo da je njihovo korištenje neophodno za svrhe navedene iznad.