**Kreacijski:**

*1. Singleton*   
Singleton pattern ćemo iskoristiti u kontejnerskoj klasi ETFKupon koja sadrži sve liste i ostale podatke potrebne za validan rad aplikacije.

*2. Prototype*

Ovaj pattern smo uzeli u obzir kod dizajna klasa za nasljeđivanje kupca i firme iz apstraktne klase osoba. Nismo iskoristili jer kupac i firma nemaju toliko zajedničkih atributa.

*3. Factory*

Factory smo iskoristili za odobravanje registracije kupca ili firme od strane administratora (korisnik aplikacije traži od aplikacije instancu objekta te nudi svoje informacije, a administrator vrši validaciju svih podataka te odobrava/poništava zahtjev od korisnika aplikacije).

**Strukturalni:**

*1. Adapter*

Adapter pattern ćemo iskoristiti kod dodavanja opcija za kupca, npr. premium kupci (oni koji imaju člansku karticu posebnog tipa) ne moraju imati iste opcije kupovine kao i regularni kupci, zato možemo implementirati adapter koji to sve radi. Također adapter pattern je iskorišten kod validacije, gdje smo sve metode implementirali u jednoj klasi i kasnije u ovisnosti od potrebe koristili.

*2. Flyweight*

Ovaj pattern smo iskoristili kod određivanja role kupca, tj. podaci na formi se vežu direktno za logovanog kupca ali ostali objekti koji se mogu nalaziti u kontejnerskoj klasi i sl. ne moraju nužno imati veze sa tim kupcem, jer možda kupac nije ni stavio interes za određene kategorije, pa ima interakciju sa samo određenim brojem artikala.

*3. Proxy*

Proxy nismo iskoristili u našem projektu iz jednog razloga; mogli smo pri registraciji napraviti proxy objekt za obradu unešenih podataka te takve obrađene podatke proslijediti administratoru, međutim nismo jer želimo da se komunikacija administrator – korisnik vrši direktno a ne 3rd person.

**Patterni ponašanja:**

*1. Command*

Command smo iskoristili u projektu: poziva se prije nego što se pozove bilo koja metoda tipa registracija, dodavanje artikla i sl. Prvo pozivamo interface ICommand da se provjeri da li su svi parametri ispravni, te da li ima smisla pozivati odgovarajuću metodu za željenu akciju. Nakon toga canExecute i Execute donose odluku o pozivanju funkcije za tu radnju.

*2. Strategy*

Ovaj pattern smo iskoristili da odredimo rolu i skup formi koje otvaramo. Kako su procesi registracije za kupca i firmu različiti, na osnovu toga određujemo odgovarajuću strategiju logovanja, određivanja role, skupa formi i skupa mogućih radnji koje korisnik sistema može raditi na sistemu.

*3.Observer*

Observer pattern smo iskoristili jer lista kupaca i lista firmi mora konstantno biti ažurna sa unesenim podacima na formama. Tako da konstantno ažuriranje podataka u listama je ono što sistem konstantno radi.