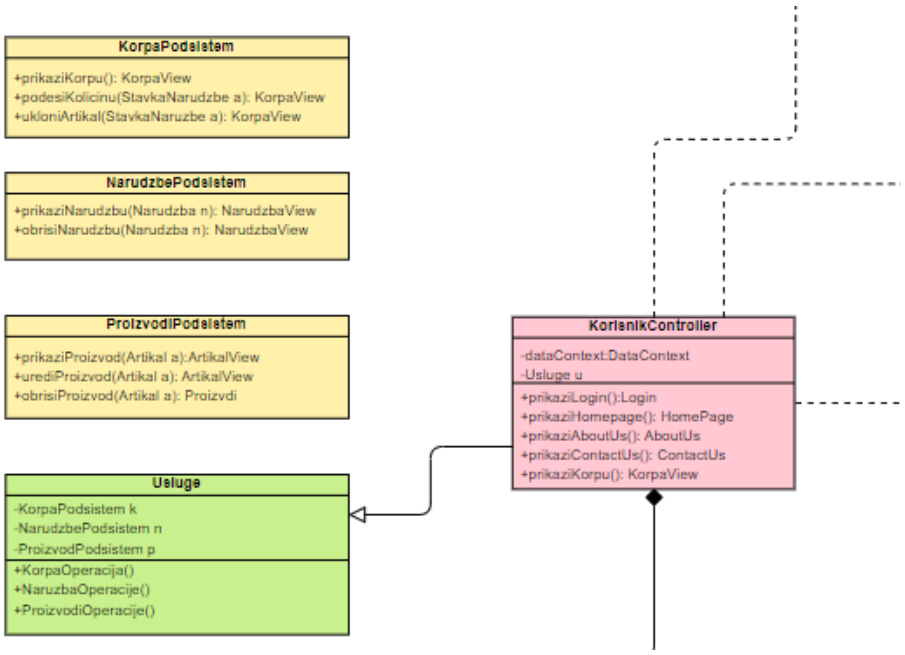
# Strukturalni paterni

1. ***Adapter patern*** – ovaj tip paterna kreira novu adapter klasu koja služi kao posrednik između originalne klase i željenog interfejsa. Ovime omogućavamo implementaciju neke funkcionalnosti bez ugrožavanja integriteta cijele aplikacije. Adapter patern bismo mogli primijeniti za obradu kartičnog plaćanja, pošto će se za obradu koristiti neki vanjski servis. Možemo implementirati adapter koji će preuzeti odgovornost za obradu detalja transakcije, tj. kada se unesu podaci vezani za kartično plaćanje, adapter ih prilagođava formatu koji taj vanjski servis zahtijeva.

2. ***Facade patern*** – ovaj patern se koristi kada je potrebno da se osigura više pogleda visokog nivoa na podsisteme čija je implementacija sakrivena od korisnika. U našem slučaju, ovaj patern bi nam bio mnogo koristan, budući da su backend sistemi poprilično kompleksni, te je samu implementaciju potrebno sakriti od korisnika. Mogli bismo imati podsistem za upravljanje korpom, narudžbama, te za sam pregled proizvoda, i na osnovu tih podsistema formirati facade interfejs UslugeInterfejs. Ova ideja je prikazana na slici ispod.



3. ***Decorator patern*** – koristi se kada je potrebno da se omogući dinamička dodjela novih elemenata i funkcionalnosti postojećim objektima. Ovaj paterni bismo mogli primijeniti u svrhu proširenja funkcionalnosti prilikom narudžbe artikla, kao npr. mogućnost unosa nekog koda za popust na naručene artikle, ili recimo mogućnost ostavljanja detaljnih naznaka vezanih za način isporuke.

4. ***Bridge patern*** – osnovna namjena ovog paterna jeste da omogući odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više implementacija za pojedine apstrakcije. Ovaj patern bi nam poslužio kada bismo klasu Artikal implementirali kao apstraktnu klasu, a svaki pojedini tip namještaja (stolice, fotelje, stolovi, kreveti i sl.) naslijedili iz ove klase.

5. **Composite patern** – omogućava formiranje strukture drveta pomoću klasa, u kojoj se individualni objekti (listovi stabla) i kompozicije individualnih objekata (korijeni stabla) jednako tretiraju. U našem slučaju, ovaj patern bismo mogli primijeniti recimo na klase Narudzba, StavkaNaruzbe i Artikal, i to na način da Narudzba predstavlja složeni objekt koji sadrži StavkaNarudzbe, dok StavkaNarudzbe u sebi sadrži informacije o artiklu.
6. ***Proxy patern*** – virtuelni proxy se koristi kada želimo odgoditi stvaranje stvarnog objekta kada on nije potreban; remote proxy rješava prblem kada se objekt ne može instancirati direktno, dok zaštitni proxy omogućava pristup i kontrolu pristupa nad stvarnim objektima. Konkretno za naš slučaj, proxy objekat bismo mogli koristiti za autentifikaciju i autorizaciju administratora za obavljanje određenih zadataka, recimo uređivanje artikala, pregled i brisanje narudžbi i sl. Proxy objekat će različitim korisnicima omogućiti različit nivo pristupa, ne narušavajući integritet same aplikacije. Proxy objekat će biti posrednik između korisnika i stvarne implementacije objekta, osiguravajući da se samo korisnici koji imaju odgovarajuće privilegije mogu vršiti određene akcije.

7. ***Flyweight patern*** – osigurava brzinu dohvaćanja svakog objekta na zahtjev, koristeći činjenicu da postoje situacije u kojima su pojedini dijelovi klase isti za sve instance te klase. Odabrali smo ovaj patern budući da nam optimizira memoriju i brzinu odgovora na pojedine zahtjeve, budući da ćemo imati poprilično mnogo različitih artikala. Smanjujemo zauzeće memorije tako što dijelimo zajedničke podatke. Uzmimo recimo krevete koji su dio našeg asortimana, pa ćemo u bazi imati samo jednu instancu artikla tipa krevet koji ćemo dalje koristiti za kreiranje narudžbi, i na taj način ostvariti značajnu uštedu memorije.