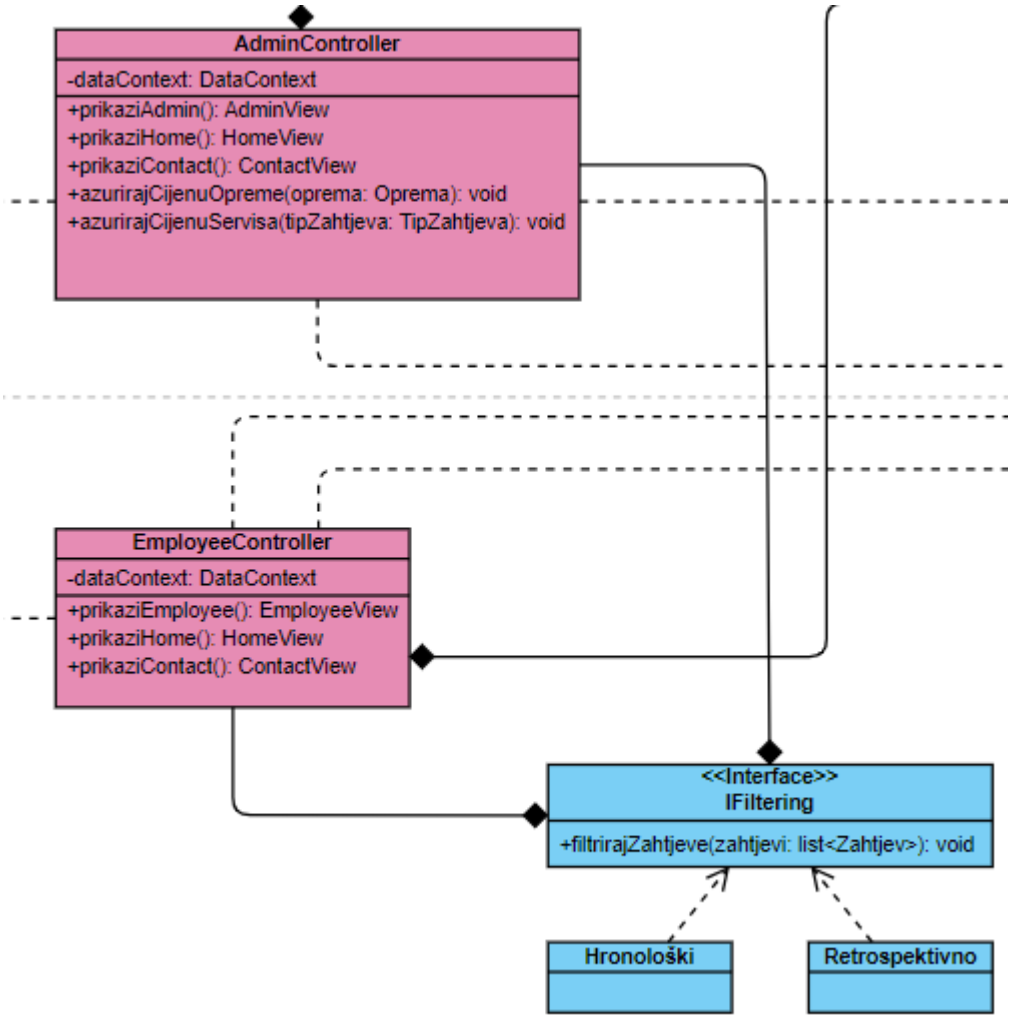
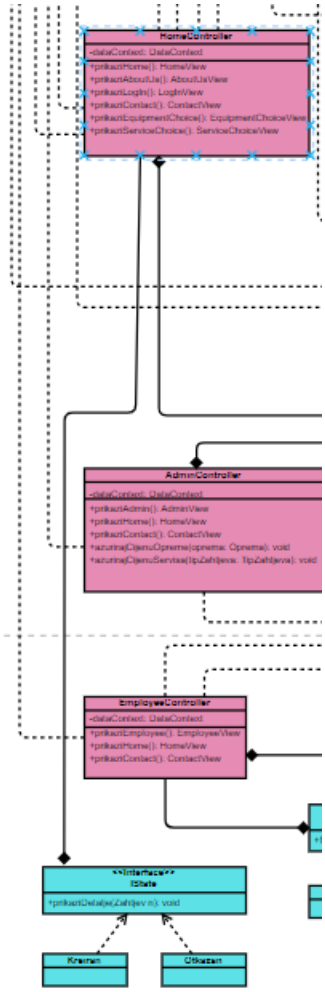
Patterni ponašanja

Strategy pattern – izdvaja algoritam iz matične klase i uključuje ga u posebne klase. Pogodan je kada postoje različiti algoritmi za neki problem i omogućava korisniku odabir jednog od implementiranih algoritama. U našem konkretnom primjeru ćemo iskoristiti strategy pattern prilikom filtriranja zahtjeva hronološki i retrospektivno zavisno od vremena kada su napravljeni. Napravit ćemo interfejs IFiltering koji će biti povezan sa kontrolerima koji rade sa filtriranjem. Također, ovaj pattern se mogao iskoristiti prilikom algoritma preporuke gdje sistem na osnovu unesene visine i vještine korisnika poreporuči određeni set opreme.



State pattern – primjenjuje se kada objekat ima više mogućih stanja. U našem primjeru, zahtjev može biti takav da je napravljen ili otkazan. Zahtjev je napravljen kada su popunjena sva obavezna polja i korisnik je odabrao finaliziranje svog zahtjeva, a zahtjev je otkazan kada korisnik putem svog ID-a zahtjeva svojevoljno

otkaže isti. Ovo ćemo sada i implementirati. Imat ćemo interfejs IState koji će biti povezan sa odgovarajućim kontrolerima. Da ponovimo, mogućnost otkzivanja zahtjeva ima isključivo korisnik.



TemplateMethod pattern – omogućava algoritmima da izdvoje pojedine korake u podklase. U našem projektu bi se mogao ovaj pattern implementirati kreiranjem abstraktne klase prikaza zahtjeva iz koje bi se naslijeđivale klase AdminMenu i EmployeeMenu. Da ponovimo, admin ima mogućnost izmjene cijena opreme, dok zaposlenik to nema. Oboje imaju mogućnost pregleda zahtjeva ali ne na isti način.

Chain of Resposibility pattern – predstavlja listu objekata, ukoliko objekat ne može da odgovori prosljeđuje zahtjev narednom u nizu. U našem projektu bi se moglo iskoristiti prilikom obrade korisničkih zahtjeva za pomoć. Okvirno bi se dizajnirao sistem koji bi postavio zaposlenika za podršku korisnicima, te ukoliko on nije u stanju da riješi korisnički problem, on se prosljeđuje dalje menadžeru, pa vlasniku, dakle, hijerarhijski.

Command pattern – razdvaja klijenta koji zahtjeva operaciju i omogućava slanje zahtjeva različitim primaocima. Mogao bi se iskoristiti u našem projektu tako što bi kroz ICommand interfejs imali klase koje bi modificirali određene informacije u nekom zahtjevu. Najlogičnije i najprirodnije bi bilo da se korisnik predomisli i izmijeni svoju narudžbu i npr. umjesto prethodno odabranih skija odluči se za iznajmljivanje snowboarda.

Iterator pattern – omogućava pristup elementima kolekcije sekvencijalno bez poznavanja interne strukture kolekcije. Npr. U našem projektu bi se mogao ovaj pattern iskoristiti na način da admin ima pristupe kolekcijama korisnika koji su iznajmili skije i opremu, snowboard i opremu, isključivo popravak opreme itd. Omogućili bi jednostavniji pristup svojim korisnicima tako sto bi kategorizovali iste radi lakšeg targetovanja publike ili pregleda korisničkih trendova i sl.

Mediator pattern – enkapsulira protokol za komunikaciju među objektima dozvoljavajući da objekti komuniciraju bez međusobnog poznavanje interne strukture objekta. Npr. U slučaju gdje bi SkiRentify biznis postao poznata franšiza na svim poznatim skijalištima iz BiH, bilo bi korisno implementirati Mediator pattern zarad lakše komunikacije između pojedinačnih biznisa, eventualne razmjene inventara itd. Zaposlenici bi eventualne informacije o stanju inventara zapisivali i u slučaju krize u vidu nestanka skija, tu informaciju šalju mediatoru koji obavještava sljedeći najbliži SkiRentify biznis.

Observer pattern – uspostavlja relaciju između objekata takvu da kad se stanje jednog objekta promijeni svi vezani objekti dobiju informaciju. Poželjno je koristiti ovaj pattern prilikom promjene stanja raspoloživosti nekog proizvoda. Npr. U našem projektu bi se iskoristilo da obavijesti korisnika ukoliko je u SkiRentify skladištu ponestalo skija za iznajmljivanje. U tom slučaju bi korisnik bio obaviješten o tome i ne bi mogao iznajmiti nepostojeće skije.

Visitor pattern – definira i izvršava nove operacije nad elementima postojeće strukture ne mijenjajući samu strukturu. Ovaj pattern bi koristili za prolazak kroz listu gdje bi morali izvršiti neku jedinstvenu operaciju nad svakim elementom liste. Dakle kada želimo dodati neku novu funkcionalnost klasi bez prevelikog mijenjanja koda, visitor pattern je izuzetno koristan. Npr. ako bismo htjeli generisati izvještaj o prometu unutar biznisa.

Interpreter pattern – podržava interpretaciju instrukcija napisanih za određenu upotrebu. Za podršku višejezičnog prikaza web aplikacije, neophodno bi bilo implementirati interpreter pattern.

Memento pattern – omogućava spašavanje internog stanja nekog objekta van sistema i njegovo ponovno vraćanje. Poprilično self-explanatory, prilikom nekog kraha sistema bi imali spašeno posljednje ispravno stanje koji bi mogli uzeti za trenutno stanje.