

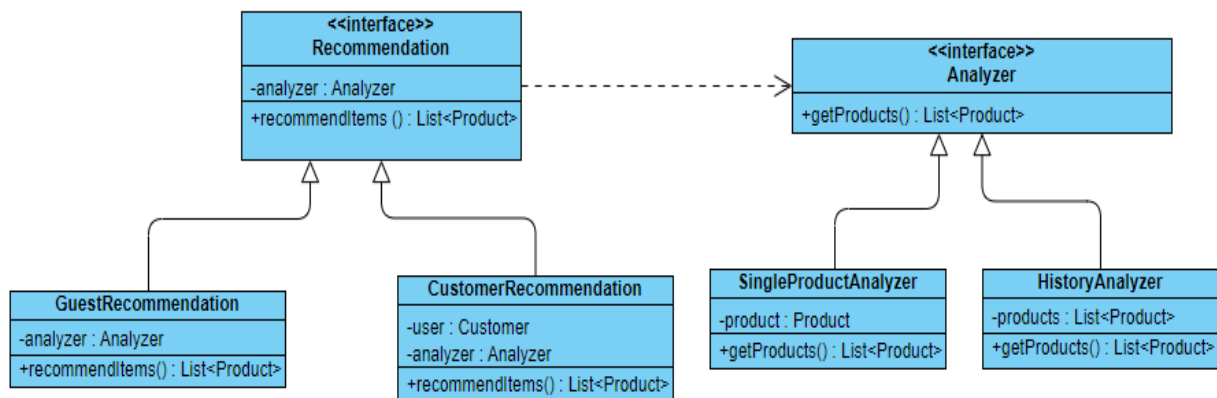
# Strukturalni paterni

## Adapter pattern

Adapter pattern bi se mogao iskoristiti u sortiranju proizvoda ako nekad budemo dodavali drugačije načine sortiranja. Trenutno je osmišljeno da postoji *Tip sortiranja* - highest price first, lowest price first itd i metoda Sort prima taj tip sortiranja. Ako bi nekad uveli drugačije sortiranje koje ne prima ni jedan Tip sortiranja nego nešto drugo onda bi konverziju umjesto mijenjanja postojećih metoda mogli uraditi pomoću ovog pattern-a.

## Bridge pattern

Bridge pattern ćemo iskoristiti prilikom preporuke proizvoda na sljedeći način. *Recommendation* je interface koji će imati dvije implementacije, jednu za *User* i jednu za *Customer* iz razloga što predlaganja za njih rade različito. S druge strane klasa *Analyzer* daje proizvode tako što ili analizira jedan proizvod pa vraća one slične njemu (kao što je primjenjivo u Product Details View-u), ili vraća proizvode na osnovu prijašnjih kupovina korisnika (kao što je na Home Page).



## Composite pattern

Composite pattern ima smisla jedino u hijerarhijskim **tree-like** strukturama. Ovakve strukture trenutno **ne postoje**(niti će) u našem projektu. Ovaj pattern međutim bi potencijalno mogli primijeniti ako bi recimo dodali menadžere kao administratore. U tom slučaju bi dodali `List<Administrator>` u našu već postojeću *Administrator* klasu što bi označavalo da je taj administrator menadžer drugim administratorima.

# Decorator pattern

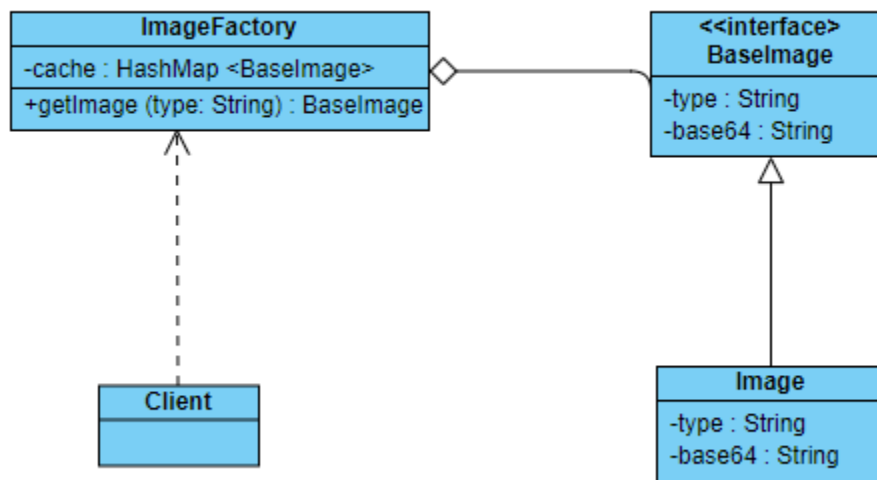
Decorator pattern bismo mogli iskoristiti za dodavanje novih funkcionalnosti u procesu registracije nekad u budućnosti na način da se npr. doda funkcionalnost slanja email-a nakon uspješne registracije ili slično, a što nije trenutno podržano.

# Facade pattern

Facade pattern bismo mogli iskoristiti prilikom implementacije validiranja kreditne kartice. Ako iskoristimo neku eksternu biblioteku za ovu svrhu ovaj pattern bi bio koristan da sakrije kompleksnost tog podsistema iza interfejsa.

# Flyweight pattern

Flyweight pattern **ćemo** iskoristiti prilikom učitavanja korisničkog interface-a jer će se koristiti veliki broj slika koji se ponekad ponavljaju (imat ćemo jednu sliku za jedan tip podataka npr. svi laptopi imaju jednu sliku, svi mobiteli jednu itd.), tako da se može iskoristiti ovaj pattern u svrhu boljeg korištenja resursa.



# Proxy pattern

Proxy pattern bismo mogli iskoristiti prilikom dodavanja novih proizvoda. Administrator umjesto da direktno dodaje proizvode u shop mora to raditi preko posrednika tj. Protection Proxy koji će prvo validirati stanje proizvoda tj. da li sve ima smisla, prije nego što pokuša dodati isti u bazu.