Adapter dizajn patern

Adapter patern dizajn služi da omogući da se interfejs već postojeće klase korsiti kao drugi interfejs, ali često omogućava i to da postojeće klase rade sa drugim bez izmjene njihovog izvornog koda. Osnovna namjena je da omogući širu upotrebu već postojećih klasa. Kreira novu adater klasu koja povezuje originalnu klasu i željeni interfejs. Na taj način se dobija željena funkcionalnost bez ikakvih izmjena na originalnoj klasi.

Ovaj patern se u našoj aplikaciji može pronaći prilikom izvršavanja naplate sa aplikacijom. Korisnik ima dvije mogućnosti izbora naplate, pouzećem ili preko svog bankovnog računa. Time dajemo aplikaciji dodatnu komunikaciju između ta dva sistema tako da prilagodi i prevodi zahtjeve i odgovore između njih.

Svaki adapter bi implementirao isti interfejs plaćanja koji očekuje ova aplikacija, ali bi interno koristio specifične interfejse za komunikaciju sa različitim procesorima plaćanja. Na ovaj način, aplikacija može da koristi isti kod za obradu plaćanja, bez obzira na to koju metodu naplate korisnik odabere.

Façade dizajn patern

Ovo je strukturalni obrazac dizajna koji obezbjeđuje pojednostavljeni interfejs za skup interfejsa u podsistemu čineći ga lakšim za korišćenje. Olakšava klijentima da koriste funkcionalnosti aplikacije bez potrebe za poznavanjem unutrašnjeg složenog sistema. Na primjer, u našem slučaju složeni interfejs za naručivanje i dostavu bi obuhvatao različite komponente poput upravljanja narudžbama, plaćanja, komunikacije sa korisnicima, upravljanje dostavom, itd. Ovaj patern bi omogućio korisnicima interakciju sa ovakvim sistemom kroz jednostavan i intuitivan interfejs. Dakle, umjesto što bi korisnici morali da pozivaju različite servise ili klase za kreiranje narudžbi, praćenje statusa istih ili komunikaciju s korisnicima, fasada bi pružila jednostavne metode koje obavljaju ove zadatke na osnovu korisničkog zahtjeva. Ovakav patern je koristan iz razloga što skriva kompleksnost podsistema od klijenata. Klijenti koriste jednostavan interfejs, a pri tome ne poznaju interne detalje implementacije.

Decorater dizajn patern

Služi da omogući dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima. Takođe, omogućava i da se funkcionalnost podijeli između klasa u zavisnosti od područja interesa. Ovaj patern nasljeđuje originalnu klasu, ali se ne oslanja na nasljeđivanje prilikom dodavanja novih atributa i objekata, već kreira nove. Na primjer, u našem slučaju ovaj patern bismo mogli iskoristi ukoliko želimo da korisniku pružimo mogućnost promjene njegovih podataka, npr. promjena emaila-a ili drugih podataka.

Proxy design patern

U našoj aplikaciji ovaj patern bi mogao da se odnosi na samo ograničenje pristupa. Može se koristiti za implementaciju ovog koncepta tako što će djelovati kao posrednik između korisnika aplikacije i resursa, provjeravajući i primjenjujući određena pravila ili ograničenja prije nego što omogući pristup resursima. Primjer može biti autentifikacija i autorizacija. Što se autentifikacije tiče, ovaj patern može da provjeri identitete korisnika prije nego što im omogući pristup određenim funkcionalnostima ili podacima. Nakon autentifikacije, ovaj patern može odobriti ili odbiti pristup određenim resursima ili funkcionalnostima na temelju njihovih privilegija. Proxy može da provjeri je li korisnik administrator ili obični korisnik prije nego što im dopusti pristup administratorskim funkcijama.

Bridge design patern

Kada bi naša aplikacija podržavala različite načine prikaza informacija, na primjer na web stranici, mobilnoj aplikaciji, itd. Svaki od ovih načina prikaza bi zahtijevao različitu implementaciju, ali bi dijelio zajedničke apstrakcije poput liste restorana, prodavnica, ili prikaza detalja restorana i mogućnosti naručivanja hrane, odnosno namirnica. Obzirom da pravimo samo web aplikaciju ovaj patern neće postojati u našem primjeru, zbog kompleksnosti implementacije.

Flyweight patern

Strukturalni patern koji bismo mogli iskoristiti kada bi naši modeli imali osobinu koja se naknadno postavlja, te joj se dodjeljuje neka defaultna vrijednost koja se nalazi u samom sistemu radi smanjenog korištenja memorije. Flyweight obrazac može biti posebno koristan za optimizaciju upravljanja stavkama menija, detaljima restorana, korisničkim profilima i drugim ponavljajućim podacima. To može dovesti do efikasnije upotrebe memorije i boljih performansi, posebno u scenarijima gdje je potrebno upravljati velikim brojem sličnih objekata istovremeno. Spomenuta osobina nije značajna samo za razvoj toka aplikacije. Trenutno dizajnirani sistem

kojeg imamo, koristi podatke koji su zaista potrebni aplikaciji, te iz tog razloga ne vidimo neku potrebu za ovim paternom. Međutim, mogli bismo ovaj patern iskoristiti za default popust prilikom kreiranja nove narudžbe starog korisnika.

Composite patern

Ovaj patern služi da bismo postigli hijerarhiju objekata, tako što ćemo kreirati strukturu stabla pomoću klasa, u kojoj se kompozicije individualnih objekata, kao i oni sami ravnopravno tretiraju, odnosno moguće je pozvati zajedničku metodu nad svim klasama. Korištenje kompozitnog obrasca omogućuje nam da jednostavno manipuliramo i obrađujemo cijele grupe objekata na isti način kao i pojedinačne objekte, što olakšava upravljanje složenim strukturama podataka. Osim toga, omogućuje nam fleksibilnost u dodavanju ili uklanjanju novih elemenata u hijerarhiji bez potrebe za promjenom koda koji se koristi za obradu tih elemenata. Kao ideju za ovaj patern možemo omogućiti da se korisniku pokaže konačni račun narudžbe, te se cijena računa na osnovu popusta(ako postoji), dalje obrađuje na osnovu odabira načina plaćanja. Potrebno je naglasiti da bi se onda morale praviti posebne klase vrste plaćanja koje bi se na različit način implementirale.

Jednostavniji način implementacije ovog paterna jeste taj da smo omogućili vrlo lako dodavanje dodataka narudžbi u vidu sosa, priloga, itd. Na primjer, začinima vlo lako možemo dodati neki kojeg nemamo, a da ne mijenjamo ostatak koda.