KREACIJSKI PATERNI

1. SINGLETON

Obzirom da naš API obavlja glavni dio posla – kreiranje planova ishrane, a on je u vezi sa jedinstvenom klasom ApiWrapper koji komunicira sa svim klasama i na taj način prikuplja sve potrebne informacije za samo kreiranje konkretnog plana ishrane, potrebno je osigurati jedinstveno instanciranje klase ApiWrapper pomoću Singleton paterna. Singleton patern će biti implementiran u klasu ApiWrapper, a sadržavat će privatnu static varijablu koja čuva jednu/jedinstvenu instancu klase, javnu static metodu (getInstance) preko koje će se pristupati Singleton klasi i privatni static objekat koji se interno instancira korištenjem privatnog konstruktora.

2. PROTOTYPE

Klijent prilikom registracije unosi svoje specifikacije koje kasnije pomažu nutricionisti da kreira jedinstven plan ishrane. Neki planovi ishrane mogu se razlikovati samo u jednoj namirnici, konkretan primjer bi bio da imamo klijenta sa alergijom na orašaste plodove te bi se njegov pojedini obrok razlikovao samo u tome da ne bi uključivao neki od plodova koji mu izazivaju alergiju. Stoga vidimo da bi se mnogi planovi ishrane razlikovali u minimalnim stvarima, tako da bi se neki planovi ishrane mogli iskoristiti za više klijenata uz neke minimalne modifikacije tj. mogao bi postojati neki osnovni obrok koji bi odgovarao svim/mnogim korisnicima, a onda bi se takav obrok nadograđivao namirnicima koje određenom klijentu ne smetaju. Nakon kopiranja objekta (obroka), nutricionista može mijenjati njegove karakteristike bez da to utiče na originalni objekat. Mogli bismo napraviti interfejs IKopiraj sa metodom kopiraj() koji bi implementirao klasu Meal. Klasa Meal će implementirati metodu kopiraj za ranije objašnjene potrebe.

3. FACTORY METHOD

Za objašnjenje načina na koji ćemo dodati ovaj patern u naš sistem, nadovezat ću se na ideju izloženu u objašnjenju prethodnog paterna. Naime imat ćemo neki bazni obrok. Sada bismo mogli dodati klase koje dodaju/oduzimaju sastojke u zavisnosti od načina prehrane koji je klijent odabrao kao prihvatljivi. Tako bi recimo postojao bazni obrok, a onda bi se u slučaju da se radi o klijentu koji ima netoleranciju na gluten, mogle dodati još neke namirnice koje će obogatiti obrok, ali se neće kositi sa njegovom intolerancijom. Dakle ideja je da se tvorcu plana ishrane (u našem slučaju je to API pa ne moramo ulaziti u detalje načina na koji će API izvesti neke akcije) olakša kreiranje dnevnog/sedmičnog plana ishrane tako da se odvoje različite kategorije koje prate

različite načine ishrane. Također dodavanje ovog paterna u naš sistem bi bilo od koristi u slučaju da se pojave neki novi načini ishrane koje je potrebno dodati u postojeći sistem.

4. ABSTRACT FACTORY

Za dodavanje ovog paterna, također ćemo se osvrnuti na različite načine prehrane i specifikacije klijenta. Iako za nas API obavlja kreiranje plana ishrane uz korištenje informacija koje mu prikupimo, možemo zamisliti jedan od mogućih scenarija kako bi to API mogao raditi. Kako smo ranije objasnili, naš klijent prilikom registracije ispunjava 5 stranica o ličnim specifikacijama – fizička sprema, težina i visina, način prehrane, BMI. Mogli bismo kreirati apstraktne fabrike – FactoryBMI, FactoryActivityLevel itd. iz kojih bi bile naslijeđene konkretne fabrike npr HighActivityLevelFactory. Recimo razne klase koje bi predstavljale različite "teme" u zavisnosti od načina ishrane – Gluten Free, Ketogenic itd. bi se bavile različitom problematikom i pomagale nutricionisti da kreira plan ishrane. Ukoliko odlučimo u nekom trenu da dodamo još dodatnih specifikacija s ciljem što preciznijeg kreiranja plana ishrane, samo bismo dodali još apstraktnih fabrika te vidimo da bi se dodavanje ovog paterna moglo pokazati korisnim jer postojeće specifikacije neće biti narušene.

5. BUILDER

Ovaj patern bismo mogli iskoristiti kod kreiranja objekata tipa User jer klasa User sadrži veći broj atributa koji bi se mogli odvojeno definisati, ali idalje ne prevelik broj atributa tako da nije kompleksno ni naše postojeće rješenje. No to bi se moglo promijeniti ukoliko se klasa User nadogradi sa mnogo više informacija o korisniku i u tom slučaju bi ovaj patern bio itekako od koristi. U slučaju implementacije Builder paterna, postojao bi interfejs IBuilder sa različitim dijelovima kreiranja korisnika, npr. metode dodajImeIPrezime(String ime, String prezime), dodajMjestoRodjenja(String drzava, String mjestoRodjenja), dodajSliku(String url) itd. Također postojali bi i različiti builderi - BuilderUser bi pozvao samo osnovne metode, dok bi BuilderUserVip pozvao i dodatne metode koje bi bile u skladu sa mogućnostima tog VIP korisnika (trenutno takav korsnik nije predviđen, no za potrebe dodavanja ovog paterna u naš sistem, ta funkcionalnost bi se mogla uvesti).