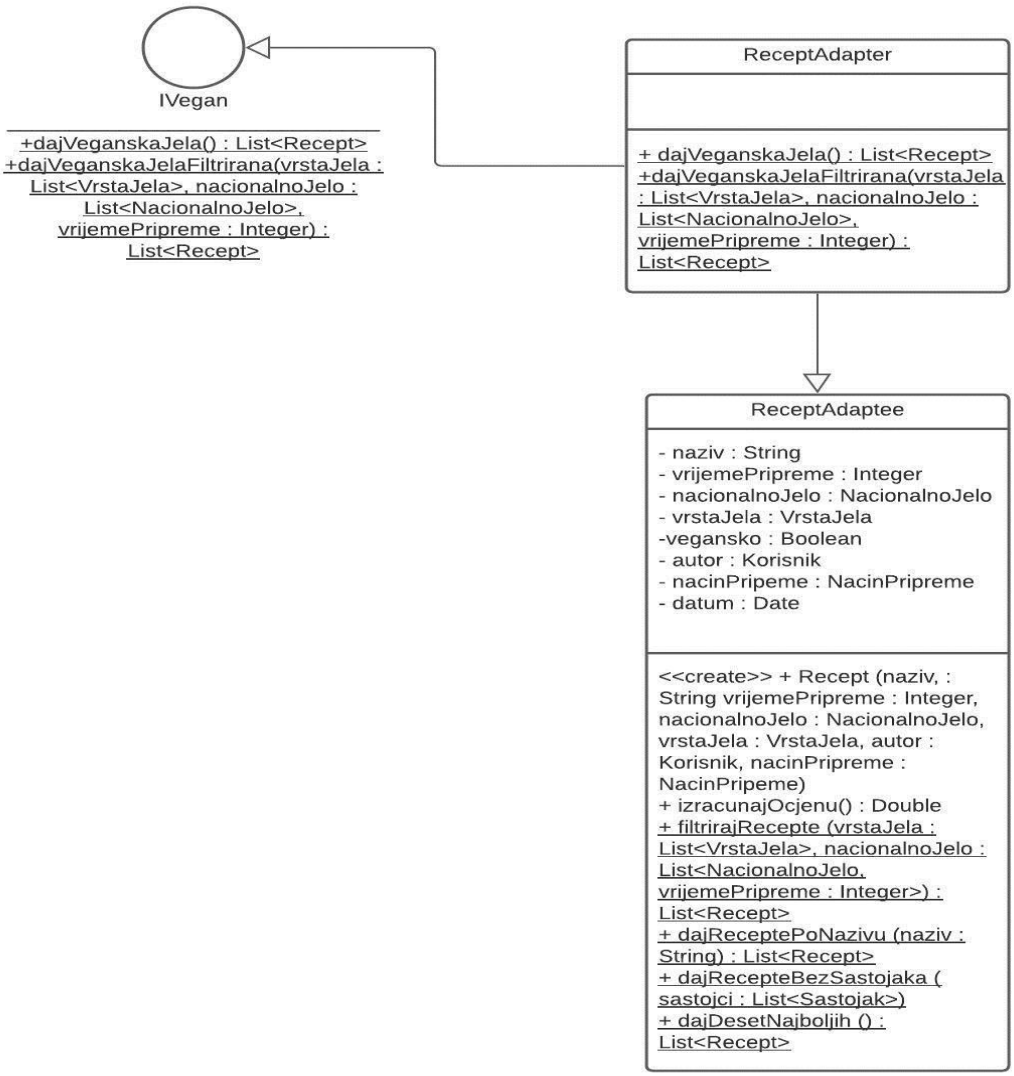
Strukturalni paterni

Adapter patern

Adapter patern omogućava širu upotrebu već postojećih klasa. Ukoliko je potreban drugačiji interfejs već postojeće klase, a ne želimo mijenjati postojeću klasu, koristi se Adapter patern koji kreira novu adapter klasu.

Zaključili smo da, u našem projektu, postoji potreba za dodavanjem novih metoda koje vraćaju veganske recepte iz sistema. Stoga smo odlučili da implementiramo Adapter pattern nad klasom Recept. Klasa Recept je preimenovana u ReceptAdaptee i to je upravo naša Adaptee klasa. Dodana je klasa ReceptAdapter koja nasljeđuje ReceptAdaptee, te također implementira interface IVegan sa dvije metode:

- + dajVeganskaJela() : List<Recept> - ovo je statička metoda koja iz postojećih recepata u sistemu izdvaja veganske. Interno ona poziva metodu Recept (sada ReceptAdaptee) klase *filtrirajRecepte* sa dummy vrijednostima svih parametara (postavljaju se na null) obzirom da nam ti parametri u ovom slučaju nisu potrebni.
- + dajVeganskaJelaFiltrirano(vrstaJela : List<VrstaJela>, nacionalnoJelo : List<NacionalnoJelo>, vrijemePripreme : Integer) – ovo je također statička metoda koja vrši filtriranje recepata u svrhu dobavljanja isključivo veganskih, no osim tog kriterija, može filtrirati i po ostalim kriterijima (vrsta jela, nacionalno jelo, vrijeme pripreme), stoga kada se interno poziva metoda *filtrirajRecepte*, ona se ne poziva s dummy parametrima, već se prosljeđuju parametri kojima je pozvana ova metoda.



Facade patern

Facade patern osigurava više pogleda visokog nivoa na podsisteme (implementacija podsistema skrivena od korisnika).

Kada bi omogućili da korisnik može dati ocjenu i komentar istovremeno, a ne odvojeno, mogli bi napraviti fasadnu klasu Recenzija. Unutar te klase bile bi instance klase Ocjena i klase Komentar i ukinuli bi metode dodajOcjenu, dodajKomentar, ukloniKomentar, ukloniOcjenu i sl., a umjesto njih bi imali samo dodajRecenziju i ukloniRecenziju. Dakle, izgledalo bi kao da postoji samo klasa Recenzija, iako su u njoj sadržane klase Ocjena i Komentar. Međutim, u našoj aplikaciji data je mogućnost davanja ocjene bez komentara i obrnuto, tako da nema potrebe za tim.

Decorator patern

Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima. Objekat pri tome ne zna da je urađena dekoracija što je veoma korisno za ponovnu upotrebu komponenti softverskog sistema.

Recimo da želimo omogućiti korisnicima da mogu unutar naše aplikacije vršiti uređivanje svojih slika (za profil ili za recept). Tada je potrebno kreirati klasu Slika, te interface ISlika koji definira metodu urediSliku(). Zatim, možemo dodati dvije Decorator klase (u budućnosti možemo dodati i više) koje služe za dvije različite vrste uređivanja slika. Klasa SlikaOsnovnePromjene pruža opcije rezanja i rotiranja slike, dok druga Decorator klasa SlikaFilteri pruža mogućnosti primjene crno-bijelog filtera ili izoštravanja slike. Obje ove klase implementiraju interface ISlika i interno sadrže instancu klase Slika. Ovim je omogućeno kreiranje instance klase Slika, što predstavlja originalnu sliku bez ikakvih promjena, a naknadno se mogu izvršiti promjene nad tom slikom, bilo da su to samo osnovne promjene ili samo filteri, bilo da se vrši kombinacija ovih decoratora.

Bridge patern

Bridge patern omogućava odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije.

Recimo da smo omogućili korisnicima da prilikom registracije odaberu preferirani tip ishrane: vegetarijanska, veganska, bez preferenci. Na osnovu odabranih preferenci, vrši se odabir recepata koji će se preporučiti korisniku (dodali bismo novi tab na Home pogledu koji bi sadržavao preporučene recepte). Mogli bismo iskoristiti Bridge pattern tako što bismo kreirali klasu Preporuka koja interno sadrži instancu interface-a IPreporuka. Ovaj interface bi se sastojao od metode dajPreporuceneRecepte() koja uzima sve recepte u bazi/sistemu i vraća one koje bi preporučila korisniku. Preporuka bi se vršila na osnovu ocjene recepata, ali i odabranom tipu ishrane. Prema tome, imali bi tri različite implementacije u vidu klasa Vegetarijanac, Vegan, Omnivore koje bi implementirale interface IPreporuka. Tako bi se prilikom odabira preporučenih recepata koji će se pružiti korisniku instancirala klasa Preporuka sa odgovarajućom internom klasom koja implementira interface IPreporuka, na osnovu odabranog preferiranog tipa ishrane korisnika.

Proxy patern

Namjena Proxy paterna je da omogući pristup i kontrolu pristupa stvarnim objektima.

Recimo da sistem kreira mjesečni izvještaj o prometu u sistemu i želimo omogućiti pristup ovim izvještajima samo administratorima koji su zaposleni prije više od godinu dana. Tad bi trebali dodati atribut koji govori o datumu zaposlenja Administratora, te klasu Izvjestaj koja upravo predstavlja generirani izveštaj. Da bi se ispoštovao Proxy pattern, trebao bi se dodati i interface IIzvjestaj koji pruza metodu prikaziIzvjestaj(). Ovaj interface implementira klasa Izvjestaj. Dodala bi se i Proxy klasa koja implementira metodu pristupiIzvjestaju() i interno sadrzi instancu Iizvjestaj. Unutar metode za pristup se provjerava da li osoba koja želi pristupiti određenom izvještaju to smije da uradi, odnosno da li joj je to dozvoljeno (na osnovu datuma zaposlenja) i ukoliko jeste, poziva se metoda prikaziIzvjestaj() nad internom instancom klase Izvjestaj, odnosno nad 'pravim' izvještajem.

Composite patern

Osnovna namjena Composite paterna je da omogući jednako tretiranje individualnih objekata i kompozitnih objekata koji sadrže te individualne objekte, tako što se kreira zajednički interface za ove dvije vrste objekata.

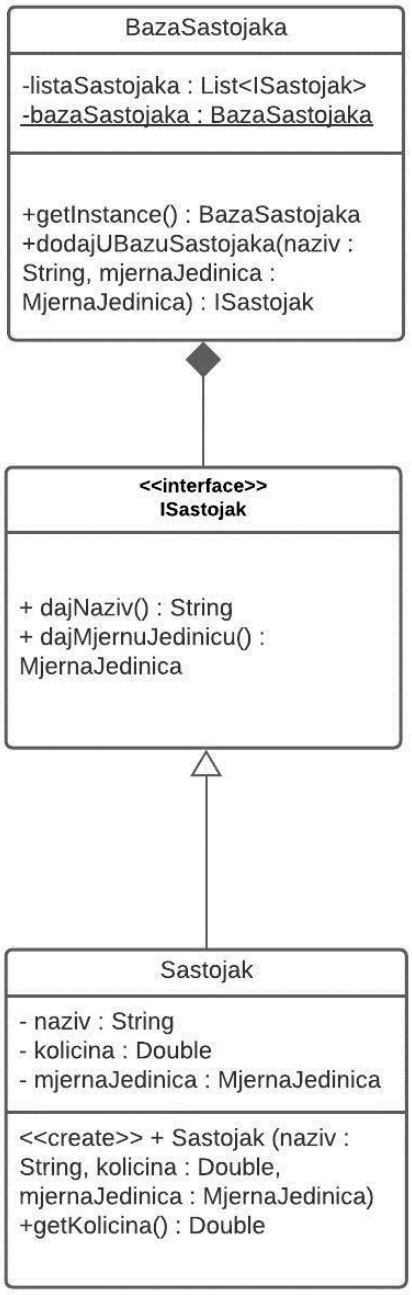
Recimo da želimo pružiti mogućnost da korisnici osvajaju nagrade u vidu besplatnog članstva i na osnovu prosječne ocjene recepata koje su objavili. Tada bi bilo potrebno proći kroz sve recepte jednog korisnika i izračunati njihove ocjene, a zatim i prosječnu ocjenu. Ovo je najjednostavnije implementirati upotrebom Composite paterna. Kreirali bi klasu VlastitiRecept u kojoj bi čuvali instancu korisnika na kog se odnosi, te listu njegovih recepata, dakle to bi bila kompozitna klasa. Također bi kreirali interface IOcjena koji bi se sastojao od metode izracunajOcjenu(). I klasa Recept i klasa VlastitiRecept bi implementirale ovaj interface (klasa Recept ionako već implementira tu metodu, pa tu ne bi bilo potrebe za bilo kakvim izmjenama). U implementaciji ove metode u klasi VlastitiRecept, nad svakim receptom u listi bi se pozvala istoimena metoda (ali ona koja je implementirana u klasi Recep), a na kraju bi se sračunala prosječna ocjena na osnovu dobivenih.

Flyweight patern

Flyweight patern omogućava da više različitih objekata dijele isto glavno stanje, a imaju različito sporedno stanje.

Uočili smo potrebu za implementacijom ovog paterna kako bi osigurali univerzalna imena i mjerne jedinice, u suštini istih, sastojaka. Dodali smo novu klasu BazaSastojaka koja interno sadrži listu sastojaka (tip List<Isastojak>). Ovo će ujedno biti i Singleton klasa, obzirom da je potrebna samo jedna instanca ove klase za čitav sistem. Također imamo i interface ISastojak koji implementira klasa Sastojak i koji prosto sadrži metode dajNaziv() koja vraća String koji predstavlja naziv sastojka, te dajMjernuJedinicu() koja vraća mjernu jedinicu korištenu za sastojak.

Prilikom kreiranja sopstvenog recepta, korisnik unosi sastojke za taj recept. Kada se unesu podaci o sastojku (naziv, mjerna jedinica, količina), poziva se metoda dodajUBazuSastojaka() klase BazaSatojaka, kojoj se upravo prosljeđuje uneseni naziv. Ukoliko se ustanovi da u listi sastojaka koja je interno sadržana u instanci klase BazaSastojaka postoji sastojak sa istim ili sličnim imenom (razlika u velikim/malim slovima), onda se taj sastojak ne dodaje u bazu, a njegov naziv i mjerna jedinica se eventualno usklađuju s nazivom i mjernom jedinicom ISastojak-a u listi, tako što se vraća ISastojak iz liste sa 'ispravnim' nazivom i 'ispravnom' mjernom jedinicom, a onda mu se atribut količina modifikuje da je u skladu s unesenom količinom. Ukoliko se pak radi o novom sastojku/nazivu, naziv se 'ispravlja' tako da prvo slovo naziva postaje veliko, a ostala su mala, te se unosi nova instanca u listu sastojaka, te taj novi ISastojak se i vraća iz metode. Instanca koja se dobije pozivom ove metode se privremeno čuva, dok se eventualno ne unese u bazu (misli se na pravu bazu, ne na našu Singleton klasu) zajedno s ostalim sastojcima i podacima o receptu, nakon što korisnik potvrdi dodavanje recepta klikom na odgovarajuće dugme.


BazaSastojaka
-listaSastojaka : List<ISastojak>
-bazaSastojaka : BazaSastojaka
+getInstance() : BazaSastojaka
+dodajUBazuSastojaka(naziv : String, mjernaJedinica : MjernaJedinica) : ISastojak
<<interface>>
ISastojak
+ dajNaziv() : String
+ dajMjernuJedinicu() : MjernaJedinica
Sastojak
- naziv : String
- kolicina : Double
- mjernaJedinica : MjernaJedinica
<<create>> + Sastojak (naziv : String, kolicina : Double, mjernaJedinica : MjernaJedinica)
+getKolicina() : Double