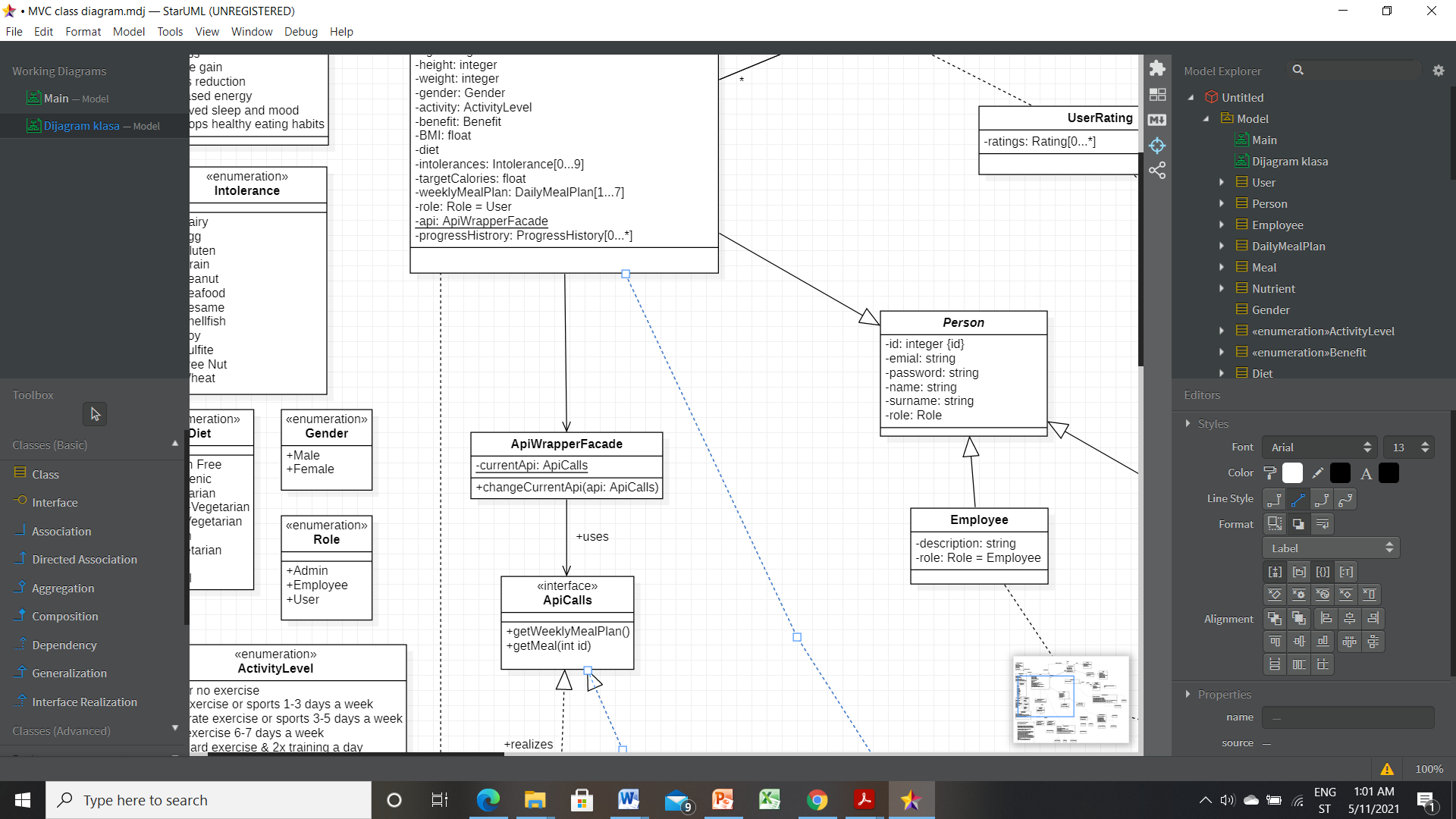
STRUKTURALNI PATERNI

1. ADAPTER

Za potrebe dodavanja ovog paterna, uvest ćemo funkcionalnost plaćanja u naš sistem. Zamislimo da se pored kartičnog plaćanja, klijentu želi omogućiti i plaćanje putem PayPal sistema. Tu bismo mogli dodati ovaj patern na način da kreiramo interfejs IPlaćanje koji će naslijediti adapter klasu , te kreiramo klasu vrstaPlaćanja, kako bismo u budućnosti mogli dodati još vrsta plaćanja (sada dodajemo PayPal plaćanje). Kreirat ćemo i VrstaPlaćanjaAdapter klasu koja će izvršiti adaptiranje vrste plaćanja iz kartičnog u PayPal (ukoliko je to ono što nam treba).

2. FACADE

Pozivom API-ja prave se jedinstveni dnevni planovi ishrane za svakog korisnika. Svaki dnevni plan se sastoji od više obroka: doručka, ručka i večere gdje svaki obrok sadrži određene vrste namirnica i nutrijenata, te također poštuje način prehrane klijenta i njegove specifične osobine. Vidimo da naša klasa *APIWrapper* već pojednostavljuje kreiranje planova ishrane nutricionisti jer komunicira sa klasama *Nutrient*, *DailyMealPlan* *dailyMealPlanMeals*, *Meal* i *User* te tako prikuplja sve gore potrebne informacije pomoću kojih kreira jedinstveni plan ishrane, stoga ćemo klasu *APIWrapper* proglasiti fasada klasom. Nije nam potrebno da znamo internu implementaciju ili neke pogodnosti korištenog API-ja, dovoljno ga je samo “pozvati”.



3. DECORATOR

Za potrebe implementiranja ovog strukturalnog paterna, dodat ćemo novi korisnički zahtjev – svaki korisnik će imati svoju profilnu sliku, te će mu biti omogućena modfikacija (rotacija, rezanje) iste. Cilj dodavanja ovog paterna je da se krajnjem korisniku olakša i unaprijedi korištenje sistema. Kreirat ćemo zasebnu klasu *Slika* koja će sadržavati atribute ime:String i slika:Bitmap. Potrebno je dodati interfejs *IslikaProfila*, kao i tri nove klase *SlikaUpdate, SlikaRezanje, SlikaRotacija.* U interfejs je potrebno implementirati metode uredi i dajSliku, a ostale tri klase će naslijediti interfejs.

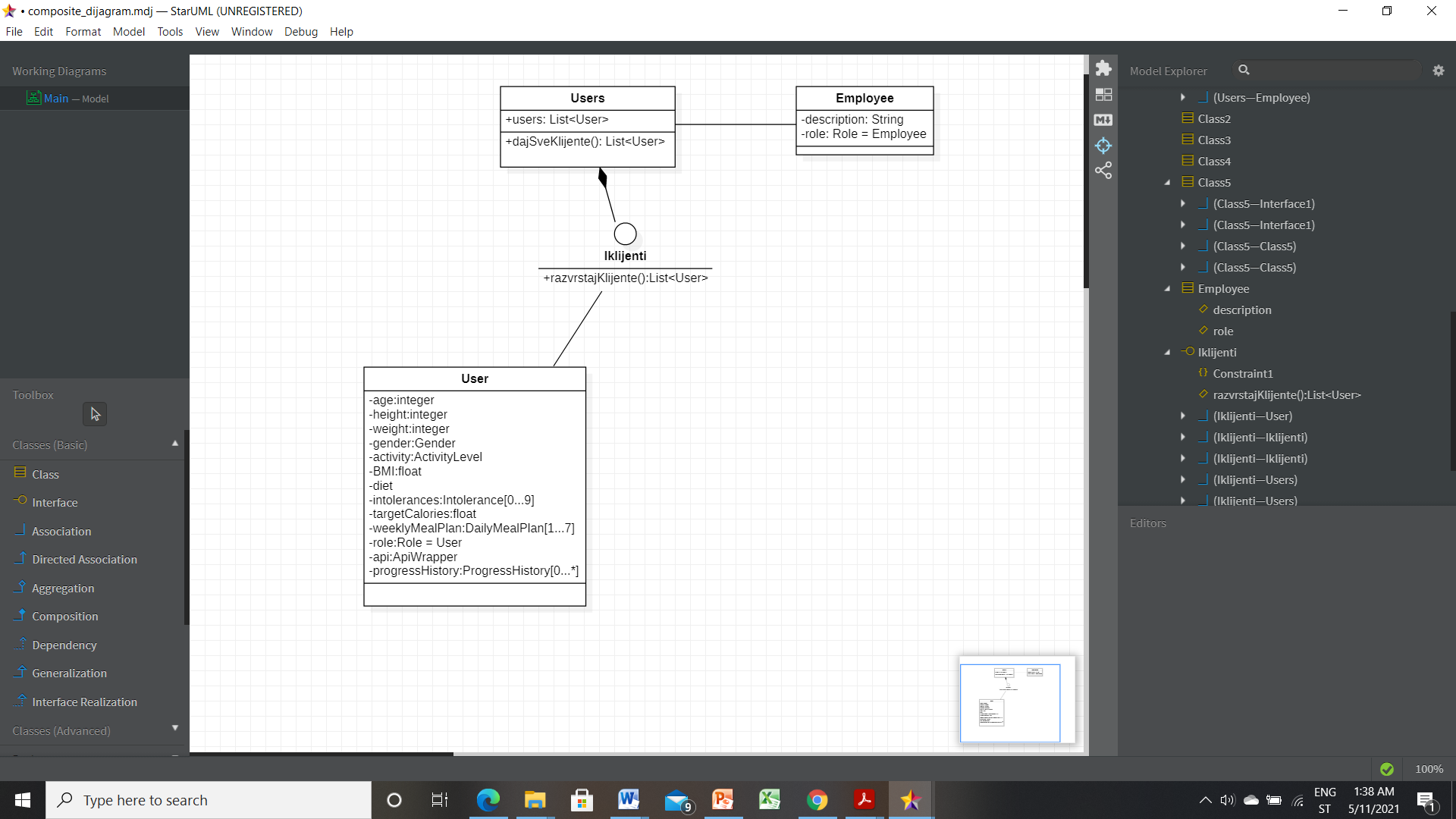
Bitno je napomenuti da bi osnovna vrsta slike imala atribut tipa Slika, a ostale bi imale tipa ISlika, čime bi se osigurao tok akcija.

4. BRIDGE

Za potrebe implementacije ovog paterna, uvest ćemo novu funkcionalnost. U momentu registracije, klijentu se nudi mogućnost uplaćivanja jednog mjeseca ili plaćanja istovremeno za naredna 2/3 mjeseca. Oni klijenti koji se odluče za plaćanje više od jednog mjeseca istovremeno, ostvaruju određene pogodnosti u vidu popusta na cijenu koju plaćaju za korištenje sistema(za plaćanje 2 mjeseca istovremeno ostvaruje se 20% popusta na ukupnu cijenu, dok za plaćanje 3 mjeseca istovremeno ostvaruje 30% popusta na ukupnu cijenu). Tada bi bilo potrebno dodati novi interfejs *IdodatnePogodnosti,* koji će sadržavati definiciju metode za izračun cijene koju plaćaju klijenti. Također, bilo bi potrebno dodati klasu *Bridge*, koja će sadržavati apstrakciju i kojoj će klijenti jedino imati pristup, ukoliko žele pregledati cijenu pogodnosti na osnovu njihovog korisničkog računa. Te bilo bi potrebno dodati atribut koeficijent: double u, već postojeću klasu, *User*, na osnovu kojeg bi se vršio izračun cijene.

5. COMPOSITE

Ovaj patern u našem sistemu možemo implementirati na sljedeći način. Ukoliko želimo dopustiti nutricionisti pregled svih korisnika u aplikaciji podijeljenih na osnovu naziva prehrane(Gluten Free, Ketogenic, Vegeterian, Lacto-Vegeterian, Ovo-Vegeterian, Vegan, Pesceterian, Paleo, Primal), trebali bismo za potrebe ovog paterna, kreirati klasu *Users* koja će imati atribut users:List<User> i u koju ćemo dodati medotu dajSveKlijente:List<User> *(primjer vraćenog rezultata: „Gluten Free: Sara Sarić, Bakir Bakirić,... Ketogenic: Harun Harunić,...“)*, koju će nutricionista moći pozvati i koja će vratiti sve klijente registrovane u sistemu sortirati po vrsti prehrane. Zatim, kreirati interfejs *IKlijenti* u koji ćemo implementirati definiciju metode razvrstajKlijente za sortiranje klijenata po vrsti prehrane. Klasa *Users* će naslijediti interfejs kako bi se kreirala hijerarhija objekata. Klasa User će zadržati atribut diet (koji će čuvati informaciju o načinu prehrane) i koji će koristiti pri sortiranju.



6. PROXY

Ovaj patern ćemo u našem sistemu implementirati na sljedeći način. Prava pristupa za pregled svih planova ishrane su ograničena. Svim planovima ishrane mogu pristupiti samo zaposleni - nutricionisti, ali ne i klijent ili admin. Vrši se provjera pristupnih podataka, te se na osnovu njih određuje da li se radi o klijentu ili nutricionisti i shodno tome omogućuje pristup planovima ishrane. Potrebno je definirati interfejs IDailyMealPlans, te definirati novu klasu Proxy koja će sadržavati atribute nivoPristupa:int (za potrebe određivanja da li pregled zahtijeva admin, nutricionista ili klijent), te planove ishrane dailyMealPlans:IDailyMealPlan. Ova klasa će naslijediti interfejs i njegove metode.

7. FLYWEIGHT

Za potrebe dodavanja ovog pattern-a, uvest ćemo novu funkcionalnost. Poznato nam je da prilikom registracije klijent bira način prehrane. Mogli bismo sve načine prehrane razdvojiti u različite kategorije koje bi nutricionista mogao pregledati i uz pomoć određenih funcionalnosti dodati/izbaciti pojedine namirnice iz plana ishrane/obroka u zavisnosti od same kategorije u kojoj se nalazi. Na taj način postigli bismo da npr. kategorija Gluten Free može sadržavati isti obrok kao i neka druga kategorija uz zamjenu običnih peciva bezglutenskim i koištenje bezglutenskog brašna u pripremi obroka. Nutricionista nakon prikupljanja informacija o klijentu može pristupiti različitoj kategoriji u zavisnosti od klijentovih želja i potreba i tako pripremiti njegov unikatni plan ishrane. Nećemo previše ulaziti u detalje, jer su isti objašnjeni u dokumentu Kreacijski Paterni, no za potrebe dodavanja ovog paterna (kao i nekih kreacijskih paterna) mogli bismo i dodati postojanje takozvanog baznog obroka koji bi nutricionisti uveliko olakšao kreiranje plana ishrane, jer bi se bazni obrok mogao lahko modificirati u zavisnosti od načina prehranepojedinog klijenta.