**Strukturalni paterni**

Potreba za dodavanjem strukturalnih paterna u sistem se prvobitno ogleda u cilju što veće standarizacije samog sistema koji se pravi, kako bi se dovelo do lakšeg korištenja tog sistema i korisnicima i programerima. Također već postojeći strukturalni paterni su dobro istraženi, te su njihove mane i prednosti unaprijed znate, što će uveliko pridonjeti mogućnosti predviđanja efikasnosti i uspješnosti sistema.

U našem slučaju strukturalne paterne od navedenih koje čemo primjeniti su:

* Adapter patern
* Composite patern

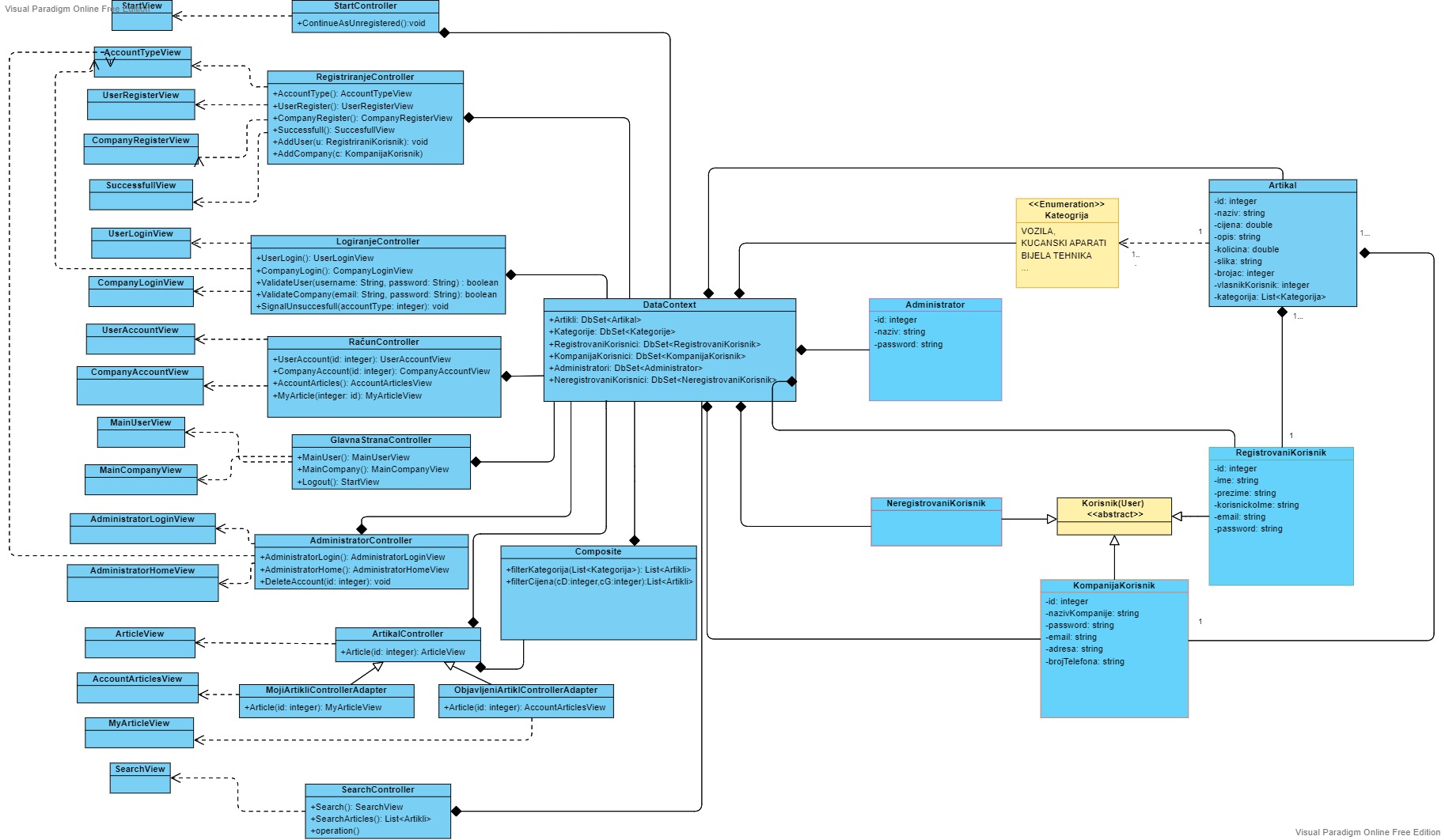
*1. Adapter patern*

Adapter patern će biti realizovan na problemu različitog prikaza interfejsa artikala zavisno od konteksta. Ovaj problem se ogleda u tome što se prikazani intefejs klase Artikli drastično mjenja od konteksta u kojem je upotrebljen, a ti konteksti su “Korisnik želi objaviti artikal”, “Korisnik želi izmjeniti artikal koji je on objavio” i “Korisnik želi da pregleda i kupi artikle drugih korisnika”. Kako svi ovi konteksti zahtjevaju klasu Artikal ali njihovi odgovarajući intefejsi predstavlju te informacije na različite načine, koristit čemo adapter patern. On će biti implementiran tako što će kontroler interfejsa za kontekst “Korisnik želi da pregleda i kupi artikle drugih korisnika” biti proglašen osnovnim, te će biti formirane nova klase MojiArtikliControllerAdapter i “ObjavljeniArtiklControllerAdapter” koje će biti nasljeđena iz te klase, te će biti povezana sa potrebnim viewovima.

*2. Composite patern*

Composite patern će biti realizovan prilikom vršenja funkcionalnosti sortiranja, pretraživanja, pregladavanja i filtriranja artikala. Korisnik će moći izabrati način prikaza artikala, podtipove artiklala koji želi da pregleda itd. Ova funkcionalnost će biti realizovana preko “Composite” klase koja će nam služiti za dobavljanje novih lista artikla ili modifikacija trenutne liste na osnovu informacija koje imamo. Composite klasa će biti povezana sa DBContext klasom od koje će dobivati podatke te sa ArtiklomController kojoj će proslijeđivati podatke po potrebi.

MVC nakon dodavanja ovih paterna će biti sljedeći:



**Ostali strukturalni paterni:**

*3. Facade patern*

Facade patern bi mogao biti vrlo efikasno i lagano realizovan na ovom sistemu, zbog toga što se sam sistem može u podpunosti podjeliti na autonomne cjeline ili podsistema. Ti podsistemu su “Kupovina artikla, “Objava artikla” i “Korisnički račun”. Ovaj patern bi bio implementiran tako što u sistem uvedemo glavni interfejs ePijaca, kojem čemo dodati odgovarajuće atribute za svaki podsistem, a koje čemo također predstaviti putem različitih interfejsa. Dalje čemo dodati odgovarajuće operacije koje će nam olakšati rad i dozvoliti da manipulišemo svakim podsistemom zasebno, što će rezultovati sa veoma preglednim sistemom.

*4. Decorator patern*

Decorator pattern bi najbolje mogao biti realizovan na klasi Artikal, kako bi dozvolili što veću modifikaciju samog prikaza artikala i informacija koje sistem čuva o različitim tipovima artikala. Neke od mogućih modifikacija su dodavanje sposobnosti uređivanja osobina objavljene slike, dodavanje videa itd. Patern bi bio realizovan na način da kreiramo klasu “Decorater” koja će biti povezana sa našom klasom “Artikal” i koja će implementirati dinamičko prošireni interfejs.

*5. Bridge patern*

Bridge patern bi najbolje bio realizovan na klasi Korisnik i njenim izvedenim klasama, tako što patern primjenimo na samoj klasi Korisnik. Ovo će rezultovati čišćom implementacijom i boljim razdvajanjem svih podtipova Korisnika, kao i podršku za dodatno dodvanje novih podklasa “Korisnik”. Ovo bi bilo implementirano na način da razdvojimo klasu Korisnik od njene implementacije “KorisnikImplementacija” koja predstavlja most koji će sadržavati sve implementacije te klase koje sistem sadrži.

*6. Proxy patern*

Proxy patern bi mogao biti realizovan na klasi Administrator, tako što toj klasi ne bi trebalo moći direktno pristupiti, a ona sama ima poprilično velike efekte na čitav sistem. Ovo će rezultovati boljom specifikacijom funkcionalnosti administratora, te učiniti implementacijom dosta lakšom i sigurnijom prilikom korištenja funkcionalnosti ove klase.

*7. Flyweight patern*

Flyweight patern bi mogao imati svrhu u problemima sljedeće prirode. Recimo u sistemu je moguće da korisnik prilikom pretraživanja artikala, otkrije svoj artikl. U tom slučaju otvaranje tog artikla ne smije rezultirati istom funkcionalnošću kojom bi rezultiralo otvaranje tog artikla od strane korisnika koji ne posjeduje taj artikl. To jeste naš artikal bi tada ušao u sporedno stanje i nastavio operirati unutar njega. Implemenacija ovog paterna bi se mogla realizirati putem nasljeđivanja svih mogućih podstanja iz osnovnog stanja, te po potrebi pozivati i smjenjivati ta stanja.