STRUKTURALNI PATERNI

Adapter pattern

Adapter patern je design pattern čija je osnovna namjena da interfejs jedne klase pretvori u neki željeni interfejs kako bi se ona mogla koristiti u situaciji u kojoj bi inače problem predstavljali nekompatibilni interfejsi. Primjenom Adapter paterna dobija se željena funkcionalnost bez izmjena na originalnoj klasi i bez ugrožavanja integriteta cijele aplikacije. U našem sistemu nismo primijenili Adapter pattern, međutim on bi našao svoju primjenu ukoliko bismo željeli proširiti sistem sa nekom od sljedećih funkcionalnosti:
-omogućiti prebacivanje kartona pacijenta iz neke druge zdravstvene ustanove, koji je u nekom drugačijem formatu, u naš sistem.
-omogućiti upload skeniranih nalaza u različitim formatima
?(statistike o zaraženima koje mogu biti u različitim formatima, a koje dobijemo preko web servisa)

Facade pattern

Facade patern se koristi kada sistem ima više identificiranih podsistema pri čemu su apstrakcije i implementacije podsistema usko povezane. Osnovna namjena Facade paterna je da osigura više pogleda visokog nivoa na podsisteme. Ovaj patern sakriva implementaciju podsistema od korisnika i pruža pojednostavljeni interfejs putem kojeg korisnik pristupa sistemu.

TO DO:
Naša fasadna klasa: metode za kreiranje i brisanje računa, metode za rezervisanje termina testiranja...
?(Da li kreirati više fasadnih klasa za različite kategorije korisnika? Da li uopšte ima potrebe za facade paternom uz dosadašnji način implementacije?)

Decorator pattern

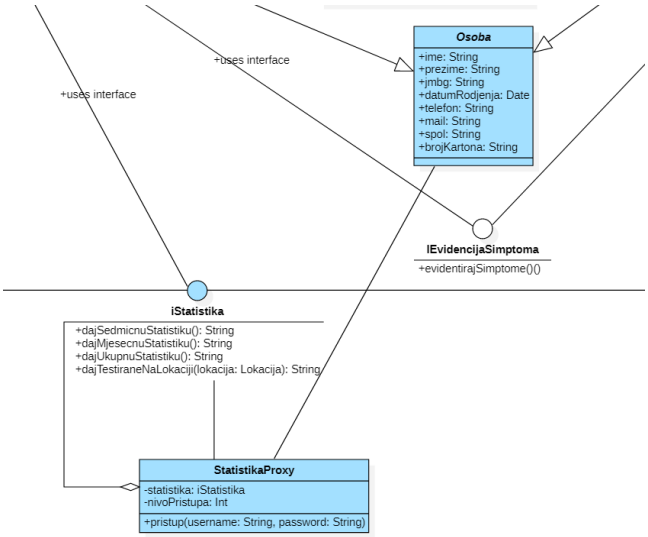
Osnovna namjena Decorator paterna je da omogući dinamičko dodavanje novih elemenata i ponašanja (funkcionalnosti) postojećim objektima.
Neki od načina na koje bismo mogli uključiti ovaj patern u svoj sistem su:
-ukoliko bismo imali potrebu za dodavanjem novog tipa kartona
-ukoliko bismo dodali više vrsta notifikacija, bilo bi korisno implementirati ovaj patern. Oobzirom da mi u našem sistemu nemamo puno vrsta notifikacija (samo in-app i email notifikacije), nema potrebe za implementacijom ovog paterna jer nema ni puno mogućih kombinacija notifikacija koje korisniku treba omogućiti

Bridge pattern

[NIJE IMPLEMENTIRAN (ZA SADA)]
Osnovna namjena Bridge paterna je da omogući odvajanje apstrakcije i implementacije neke klase tako da ta klasa može posjedovati više različitih apstrakcija i više različitih implementacija za pojedine apstrakcije.
Dakle, bridge patern razdvaja pojedinačnu klasu na više zasebnih klasa koje se mogu razvijati odvojeno jedna od druge.
Bridge patern nije primijenjen na naš sistem, međutim, u slučaju kada bi bilo potrebe računati cijenu testiranja na različite načine u zavisnosti od načina plaćanja (online plaćanje ili plaćanje na lokaciji), onda bismo primijenili ovaj strukturalni patern.
Za računanje cijene mogao se iskoristiti i Decorator patern, ali smatramo da je, kako kod nas nema potrebe za različitim izgledom korisničkog interfejsa u ovisnosti od toga da li postoji popust na cijenu ili ne, ovdje bolje primijeniti bridge patern za interno računanje cijene.

Proxy pattern

Proxy je strukturalni patern koji pruža objekat koji se ponaša kao "substitute" objekat za pravi objekat koji pruža neku uslugu. Proxy objekat prihvata korisničke zahtjeve i obrađuje ih u zavisnosti od prava pristupa korisnika. Na ovaj način onemogućen je slučajni pristup podacima od strane korisnika koji nema odgovarajuće permisije za pregled istih.



Proxy patern u našem sistemu svoju primjenu je našao kod dobijanja različitih vrsta statistika u zavisnosti od tipa korisnika koji im pristupa. Iako svi korisnici imaju pristup statistikama, određene kategorije korisnika imaju specifične statistike i izvještaje koji bi trebali biti dostupni samo njima.
Osim ove primjene, Proxy patern u našem sistemu može se primijeniti i za kontrolu pristupa bazi podataka (za pretragu korisnika i kartona - adminu omogućiti pretragu svih korisnika, medicinskom osoblju omogućiti pretragu samo pacijenata; adminu omogućiti pregled dostupnog osoblja, spisak testiranih po lokaciji testiranja, detaljnije izvještaje i sl.)

Composite pattern

Osnovna primjena Composite paterna jeste pravljenje hijerarhije klasa i omogućavanje pozivanja iste metode nad različitim objektima sa različitim implementacijama.
Ukoliko bismo željeli administratoru dopustiti pregled kartona svih korisnika (i pacijenata i medicniskog osoblja) na isti način, mogli bismo primijeniti Composite patern. Osim toga, još jedan potencijalni primjer primjene bio bi ukoliko bi pacijent i medicinsko osoblje vršili rezervaciju na isti način (što u našem sistemu nije slučaj).

Flyweight pattern

Flyweight patern koristi se kako bi se onemogućilo bespotrebno stvaranje velikog broja instanci objekata koji u suštini predstavljaju isti objekat.
U našem sistemu sve korisničke klase imaju jedinstvene identifikacione atribute (username i/ili jmbg, te lozinku), kao i klase kao što su klasa Test i klasa RezervacijaTestiranja, pa na njih nije moguće primijeniti ovaj pattern.
Jedinu iznimku predstavljaju objekti klase Guest, za koju bismo mogli primijeniti ovaj pattern tako što bismo ili grupisali zajedničke podatke za guest korisnike ili u potpunosti izbjegli kreiranje više instanci ove klase i koristili samo jednu.
?(problem istovremenog pristupa više guest korisnika)