STRUKTURALNI PATERNI

Za naš projekat odlučili smo se da dodamo dizajn paterne **Facade** i **Proxy**. Prvo da navedemo njihovu generalnu namjenu.

Zaštitni proxy omogućava pristup i kontrolu pristupa stvarnim objektima.

Facade patern se koristi kada sistem ima više podsistema (subsystems) pri čemu su apstrakcije i implementacije podsistema usko povezane. Osnovna namjena Facade paterna je da osigura više pogleda (interfejsa) visokog nivoa na podsisteme čija implementacija je skrivena od korisnika.

Pored ovih paterna, postoje i drugi poput: Bridge, Adapter, Decorator, i Composite.

Adapter pattern bismo mogli uvesti u sistem prilikom integracije eksternih servisa poput servisa za validaciju adresa, gdje bi adapter služio kao posrednik koji pretvara format eksternih podataka u naš interni model, omogućavajući nesmetanu komunikaciju između sistema. Na primjer, eksterni servis može vraćati adresu u obliku stringa, dok naš sistem koristi strukturisani objekat – adapter bi tada vršio mapiranje između ta dva formata.

Bridge pattern bi omogućio veću fleksibilnost pri generisanju dokumenata u više različitih formata (npr. PDF, DOCX, XML), razdvajanjem apstrakcije dokumenta od konkretne implementacije za formatiranje, čime bi se omogućilo dinamičko dodavanje novih formata bez izmjena postojećeg koda. U praksi bi to značilo da naša logika za kreiranje sadržaja dokumenta ostaje nepromijenjena, dok se samo mijenja implementacija za njegovo renderovanje.

Composite pattern bi mogao biti koristan u kontekstu grupisanja i upravljanja dokumentima kao cjelinama – npr. omogućavanje da korisnik izvrši operaciju (brisanje, potpisivanje, eksport) nad cijelom kolekcijom dokumenata na isti način kao i nad pojedinačnim dokumentom, što bi otvorilo mogućnosti za naprednije funkcionalnosti u budućim verzijama sistema.

Decorator pattern bismo mogli primijeniti u slučaju kada bismo željeli omogućiti slanje dokumenata putem više različitih kanala, kao što su email, SMS, in-app notifikacije ili čak push obavijesti. Umjesto da pravimo veliku klasu sa svim mogućim načinima slanja, koristili bismo dekoratore koji bi dodavali nove načine slanja bez mijenjanja postojećeg koda. Na primjer, bazni objekat bi sadržavao osnovnu logiku za slanje emaila, a zatim bismo mogli “umotati” taj objekat u dekorator koji dodatno šalje SMS, pa još jedan koji šalje push notifikaciju. Ovakav pristup omogućava fleksibilnu kombinaciju različitih metoda slanja i pojednostavljuje održavanje koda.

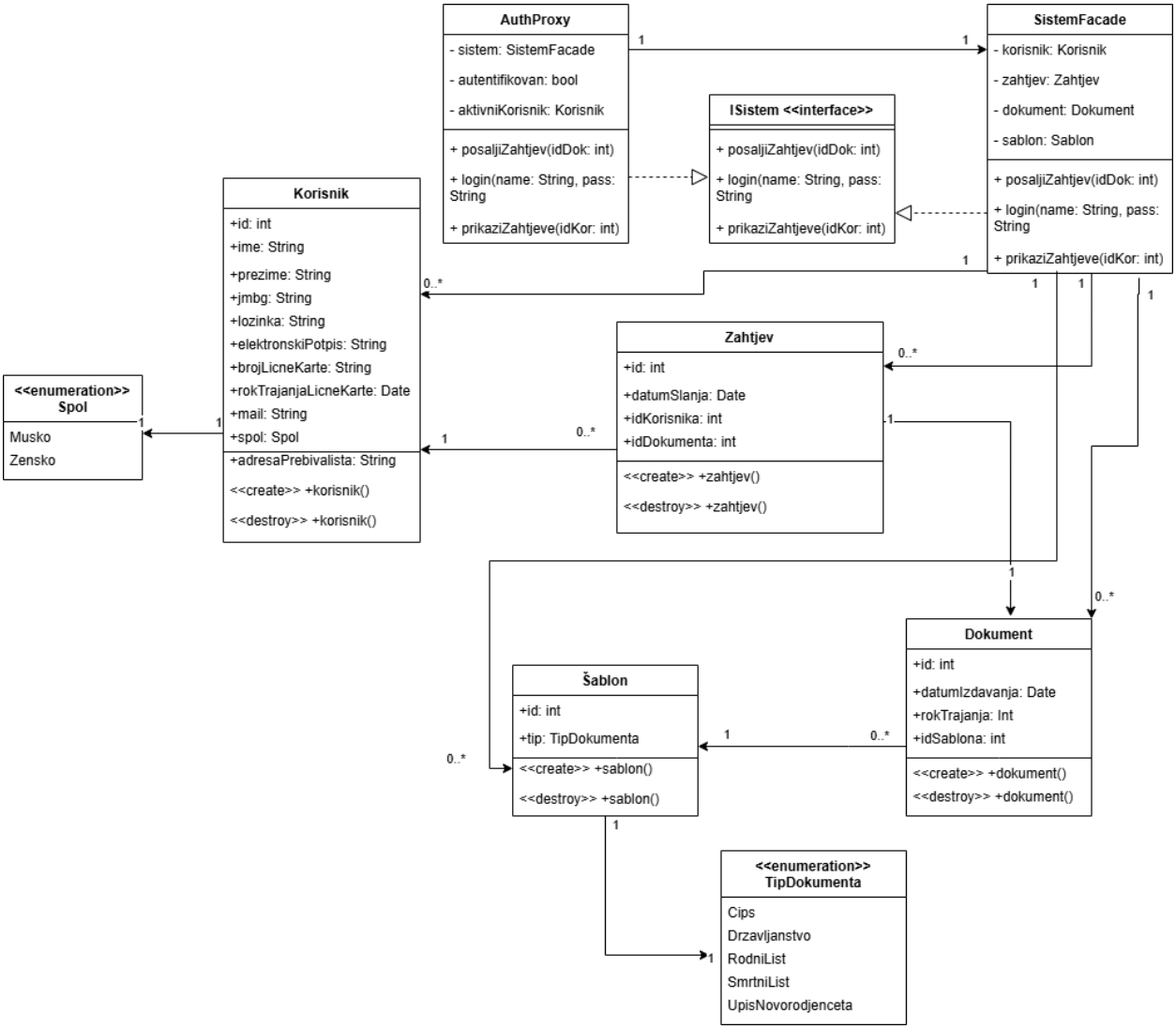
Sada navodimo konkretnu primjenu u našem sistemu.

Façade pattern ćemo iskoristiti da bismo povezali sve klase koje su neophodne za pružanje svih funkcionalnosti koje naš sistem nudi korisniku. Pošto korisnik ima više operacija koje može izvršiti a one su dio različitih klasa (npr. pregled zahtjeva iz Korisnik klase, ObradiZahtjev iz Zahtjev klase I sl.) iskoristićemo fasadu da napravimo jedan

high-level pogled na sistem da sakrijemo unutrašnje implementacije funkcionalnosti.

Zaštitni proxy ćemo iskoristiti jer naš sistem podržava login na korisničke račune, i podržava različite tipova korisnika sa različitim pravima pristupa, te moramo omogućiti kontrolu pristupa. Iskoristićemo zaštitni proxy kao način provjere ulaznih podataka prilikom pristupa računu, te omogućiti ili onemogućiti korištenje funkcionalnosti sistema na osnovu istih.

Novi dijagram klasa sa dodanim paternima izgleda kao na slici:


AuthProxy
- sistem: SistemFacade
- autentifikovan: bool
- aktivniKorisnik: Korisnik
+ posaljiZahtjev(idDok: int)
+ login(name: String, pass: String
+ prikaziZahtjeve(idKor: int)
ISistem <<interface>>
+ posaljiZahtjev(idDok: int)
+ login(name: String, pass: String
+ prikaziZahtjeve(idKor: int)
SistemFacade
- korisnik: Korisnik
- zahtjev: Zahtjev
- dokument: Dokument
- sablon: Sablon
+ posaljiZahtjev(idDok: int)
+ login(name: String, pass: String
+ prikaziZahtjeve(idKor: int)
Korisnik
+id: int
+ime: String
+prezime: String
+jmbg: String
+lozinka: String
+elektronskiPotpis: String
+brojLicneKarte: String
+rokTrajanjaLicneKarte: Date
+mail: String
+spol: Spol
+adresaPrebivalista: String
<<create>> +korisnik()
<<destroy>> +korisnik()
<<enumeration>>
Spol
Musko
Zensko
Zahtjev
+id: int
+datumSlanja: Date
+idKorisnika: int
+idDokumenta: int
<<create>> +zahtjev()
<<destroy>> +zahtjev()
Dokument
+id: int
+datumIzdavanja: Date
+rokTrajanja: Int
+idSablona: int
<<create>> +dokument()
<<destroy>> +dokument()
Šablon
+id: int
+tip: TipDokumenta
<<create>> +sablon()
<<destroy>> +sablon()
<<enumeration>>
TipDokumenta
Cips
Drzavljanstvo
RodniList
SmrtniList
UpisNovorodjenceta
1
0..*