Patterni ponasanja

- Strategy Pattern

Strategy dizajn patern izdvaja algoritam iz osnovne klase i omogućava njegovo dinamičko biranje putem različitih implementacija. U aplikaciji koja omogućava različitim korisnicima (studentima, profesorima, asistentima, adminima) da pregledaju statističke podatke, pristup tim statistikama se razlikuje:

- Student vidi broj svojih pitanja, komentara, odgovora i lajkova.
- Profesor vidi broj pitanja po predmetima koje predaje.
- Admin ima pristup sveobuhvatnoj statistici svih korisnika.

Ideja: Omogućava različite strategije prikaza statistika za različite tipove korisnika (student, profesor, admin).

Implementacija:

- Napraviti interfejs npr. IStatistikaStrategija s metodom PrikaziStatistiku(Korisnik user).
- Napravi implementacije: StudentStatistikaStrategija, ProfesorStatistikaStrategija, AdminStatistikaStrategija.
- U kontroleru (npr. StatistikaController), na osnovu tipa uloge korisnika (Korisnik.Uloga ili Identity rola), kreiramo statistiku po našoj volji, recimo za studenta broj sati proveden učeći sedmično I sl.

Korištenjem Strategy paterna, može se omogućiti biranje odgovarajuće strategije statistike bez dupliranja logike i bez izmjena glavnog kontrolera.

- State Pattern

State dizajn patern omogućava objektu da promijeni svoje ponašanje kada mu se promijeni stanje – kao da je riječ o drugom objektu. Svaki korisnik u sistemu ima status: aktivan ili neaktivan. Ovisno o tom statusu:

- Aktivan korisnik može dodavati pitanja, komentare i odgovore.
- Neaktivan korisnik ne može izvršavati nikakve interakcije – može samo gledati sadržaj.

Ideja: Ponašanje korisnika zavisi od njegovog statusa (Aktivan, Neaktivan).

Implementacija:

- U modelu Korisnik postoji status (bool ili enum).
- Napravimo state interfejs: IKorisnikState s metodama za akcije (DodajPitanje, DodajKomentar...).
- Implementacije: AktivanKorisnikState (dozvoljava akcije), NeaktivanKorisnikState (samo pregled).
- U kontrolerima (npr. QnAController), umjesto stalnog if (status == ...), koristi korisnik.state objekat korisnik.State.DodajPitanje()… I sl.

Korištenjem State paterna, sistem jasno odvaja ponašanja u zavisnosti od statusa, čime se izbjegavaja ifologija provjere i povećava fleksibilnost za eventualna nova stanja (npr. banovani korisnik).

- Template Method Pattern

Template Method patern omogućava da se definira kostur algoritma u osnovnoj klasi, dok se pojedini koraci prepuštaju podklasama. Prilikom generisanja izvještaja o aktivnostima korisnika, svi izvještaji sadrže slične dijelove (broj pitanja, odgovora, komentara...), ali se način obrade i prikaza razlikuje:

- Student izvještaj je fokusiran na ličnu aktivnost.
- Profesor izvještaj prikazuje angažman studenata po predmetima.
- Admin izvještaj sadrži detaljnu analitiku sistema.

Ideja: Kostur generisanja izvještaja je isti, ali se detalji razlikuju po tipu korisnika.

Implementacija:

- Abstraktna klasa IzvjestajGenerator s metodom GenerisiIzvjestaj(), a koraci kao virtual/abstract metode: GenerisiZaglavlje(), GenerisiSadrzaj(), GenerisiPodnozje().
- StudentIzvjestajGenerator, ProfesorIzvjestajGenerator, AdminIzvjestajGenerator nasljeđuju baznu klasu i override-aju potrebne korake, u ovisnosti od toga šta želimo staviti u izvještaju.
- U kontroleru (IzvjestajiController) instanciramo odgovarajuću podklasu.
- View može biti zajednički jer koristi generisani model.

Primjenom ovog paterna omogućava se fleksibilno dodavanje novih tipova izvještaja bez izmjena osnovne logike.

- Observer Pattern

Observer patern definiše zavisnost između objekata tako da se promjene u jednom objektu automatski propagiraju ka drugim zainteresovanim objektima. U scenarijima gdje se korisničke akcije trebaju proširiti kao obavijest:

- Kada neko odgovori na pitanje, autor pitanja automatski dobija notifikaciju.
- Kada neko komentariše odgovor, autor odgovora bude obaviješten.
- Kada se objavi novi predmet ili zadatak, određeni korisnici mogu biti pretplaćeni na obavještenja.

Ideja: Promjene na nekom objektu automatski prate zainteresovani objekti (npr. dostignuća).

Implementacija:

- Napraviti DostignućaHelper koji sadrži metode za provjeru svake vrste dostignuća
- Metode DostignućeHelpera se poziva u Pitanje, Odgovor, StudySession nakon svake promjene u bazi podataka.
- Ukoliko se desila promjena I ukoliko korisnik nema to dostignuće system mu automatski dodjeljuje isto.

Ovaj patern omogućava modularan i proširiv sistem dostignuća koji je lako povezati s drugim komponentama.

Command dizajn patern omogućava enkapsulaciju akcije kao objekta. Na taj način možeš fleksibilno upravljati izvršenjem komandi (npr. dodavanje, uređivanje, brisanje), kao i podržati mogućnost poništavanja (undo). Sistem ima različite radnje korisnika, posebno u vezi sa sadržajem:

- Dodavanje pitanja
- Lajkovanje odgovora
- Brisanje komentara
- Slanje poruke

Ideja: Svaka korisnička akcija je komanda koja se može izvršiti, poništiti (undo), logovati...

Implementacija:

- Napravimo interfejs ICommand s metodom Execute() i opciono Undo().
- Svaka akcija: DodajPitanje, LajkujOdgovor, ObrisiKomentar, PosaljiPoruku ima svoju klasu npr. DodajPitanjeCommand.
- U kontroleru umjesto direktnog izvršavanja odredjenih komandi pišemo:

```
ICommand komanda = new DodajPitanjeCommand(...);
komanda.Execute();
```

Svaka od ovih radnji može biti modelirana kao komanda. Time bi sistem:

- Imao centralizovan način upravljanja akcijama
- Moguće je undo/redo budućih koraka (npr. greškom izbrisan komentar)
- Moguće je logovati svaku korisničku akciju

- Iterator Pattern

Iterator patern omogućava sekvencijalni prolazak kroz kolekciju objekata bez otkrivanja njene unutrašnje strukture. Kolekcije kao što su:

- Lista pitanja jednog korisnika
- Svi komentari ispod odgovora
- Sve poruke između dva korisnika
- Lista prijatelja

...često se prikazuju kroz pregledne liste, bez direktnog pristupa čitavom skupu u kontroleru.

Ideja: Omogućava lagano listanje kolekcija bez otkrivanja interne strukture.

Implementacija:

- Napraviti custom iterator za, recimo, ListaPrijatelja, KomentariNaOdgovor, PorukeIzmedjuKorisnika.

- U ViewModelima koristimo IEnumerable ili svoj iterator, a u Viewu radiš jednostavan foreach.
- Može se proširiti za sortiranje, filtriranje, bez promjene glavnog kontrolera.

Iterator omogućava:

- Lakše listanje sadržaja po pravilima
- Jedinstven API za različite kolekcije

- Memento Pattern

Memento patern omogućava čuvanje prethodnih stanja objekta i njihovo vraćanje po potrebi – bez narušavanja enkapsulacije. Korisnici mogu uređivati sadržaj (npr. pitanja, odgovore). Zamislimo da student piše pitanje, ali:

- želi sačuvati verziju prije izmjene
- vrati se korak nazad (undo)
- uporedi verzije

Ideja: Čuvanje verzija objekata radi vraćanja na staro stanje (undo).

Implementacija:

- U modelima Pitanje i Odgovor napraviti vezu prema listi memento objekata (PitanjeMemento, OdgovorMemento).
- Prije svake izmjene, sačuvati trenutnu verziju u memento.
- Dodamo opciju na View: "Prikaži historiju" i "Vrati na staru verziju".
- U kontroleru PitanjeController metoda VratiNaVerziju(int mementoId) vrati stanje.

Pitanje ili Odgovor objekat može imati memento objekat koji čuva ranije verzije, a korisnik ih vidi kroz historiju uređivanja.

Idealno i za buduće funkcije "Obnovi staru verziju".

- Mediator Pattern

Mediator patern centralizuje komunikaciju između objekata, tako da objekti ne komuniciraju direktno, već preko posrednika. U nasoj aplikaciji postoji razmjena poruka između korisnika:

- Student piše profesoru
- Asistent odgovara studentu
- Admin nadgleda komunikaciju

Ideja: Centralizuje komunikaciju između objekata (npr. chat).

Implementacija:

- Umjesto da Korisnik direktno šalje poruku drugom korisniku, koristiš npr. ChatMediator klasu (PorukaService).
- Sve poruke idu kroz tog posrednika:
    - On može filtrirati, moderirati, blokirati ili arhivirati poruke.
    - Dodavanje pravila za grupne poruke ili anonimnost radiš u Mediatoru, ne u svim modelima.

Posebno korisno za buduće proširenje na grupne razgovore, anonimne poruke ili notifikacije unutar poruka.