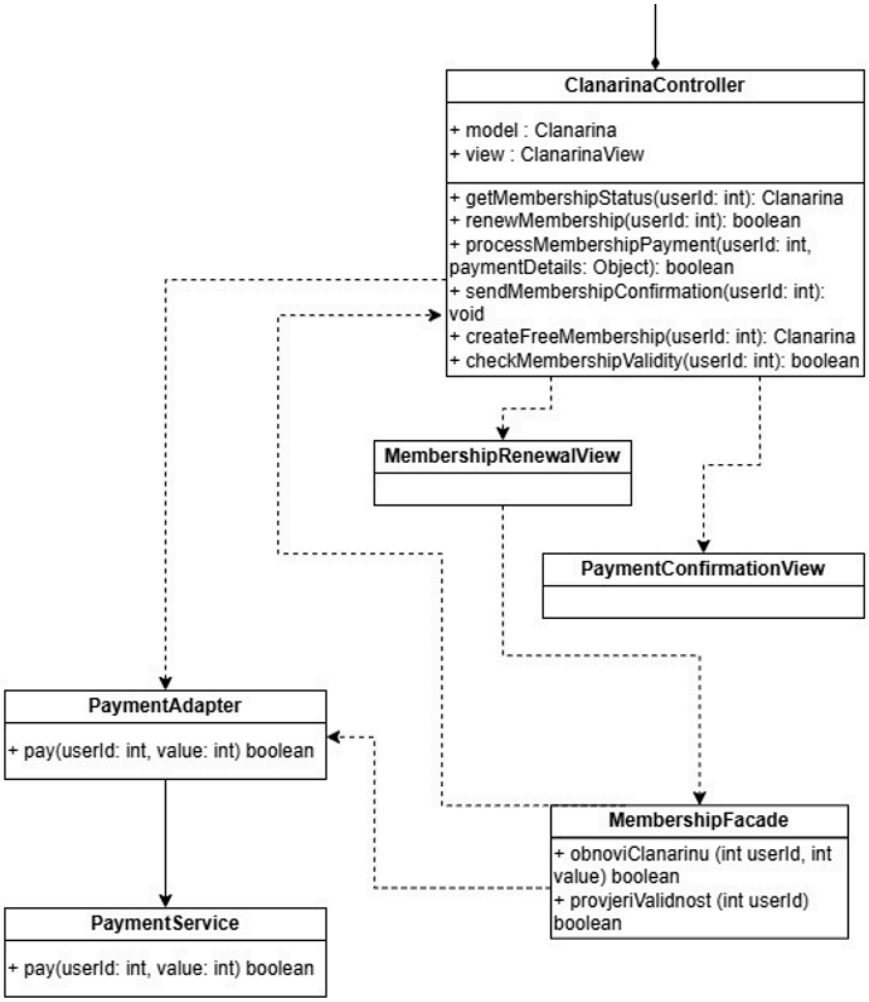
Strukturalni paterni

1.Adapter patern

Adapter pattern služi da se postojeći objekat prilagodi za korištenje na neki drugi način, bez mijenjanja same definicije objekta. Na taj način obezbjeđuje se da će se objekti i dalje moći upotrebljavati na način kako su se dosad upotrebljavali, a u isto vrijeme će se omogućiti njihovo prilagođavanje novim uslovima.
U našem projektu, Adapter pattern služi da omogući ClanarinaController-u da koristi eksterni servis za plaćanje (ServisPlacanja), iako taj servis ima drugačiji interfejs od onoga što kontroler očekuje. Korištenjem klase PlacanjeAdapter, omogućena je komunikacija između kontrolera i eksternog sistema bez potrebe za izmjenom postojećeg koda ni u kontroleru ni u servisu. Na taj način se ostvaruje prilagođavanje interfejsa uz očuvanje postojećih veza u MVC strukturi.

2. Facade pattern

Fasadni pattern služi kako bi se klijentima pojednostavilo korištenje kompleksnih sistema. Klijenti vide samo fasadu, odnosno krajnji izgled objekta, dok je njegova unutrašnja struktura skrivena. Na ovaj način smanjuje se mogućnost pojavljivanja grešaka jer klijenti ne moraju dobro poznavati sistem kako bi ga mogli koristiti.
U sistem se može implementirati Fasadni pattern kroz klasu MembershipFacade, koja pojednostavljuje korištenje funkcionalnosti vezanih za članarinu. Umjesto da korisnički interfejs direktno komunicira sa više komponenti kao što su ClanarinaController i PaymentAdapter, sve operacije poput obnove članarine i provjere validnosti sada su dostupne kroz jednu centralnu tačku. Na taj način, smanjena je složenost korištenja sistema i olakšano održavanje, jer klijenti više ne moraju poznavati unutrašnju strukturu i redoslijed poziva.



Dodani Adapter pattern i Facade pattern na postojeći MVC

3. Decorator pattern

Decorator pattern služi za omogućavanja različitih nadogradnji objektima koji svi u osnovi predstavljaju jednu vrstu objekta (odnosno, koji imaju istu osnovu). Umjesto da se definiše veliki broj izvedenih klasa, dovoljno je omogućiti različito dekoriranje objekata (tj. dodavanje različitih detalja), te se na taj način pojednostavljuje i rukovanje objektima klijentima, i samo implementiranje modela objekata.
Za implementaciju Decorator patterna u ovom sistemu može se dodati fleksibilan mehanizam za filtriranje konja prema različitim kriterijima (spol, starost, boja itd.). Umjesto da se za svaki filter piše posebna metoda, koristi se lančano dekorisanje osnovne liste konja pomoću objekata koji svaki primjenjuju jedan filter. Na taj način se omogućava dinamičko kombinovanje više filtera, bez mijenjanja postojećeg KonjController koda i bez stvaranja velikog broja statičkih metoda.

4. Bridge pattern

Bridge pattern služi kako bi se apstrakcija nekog objekta odvojila od njegove implementacije. Ovaj pattern veoma je važan jer omogućava ispunjavanje Open-Closed SOLID principa, odnosno uz poštivanje ovog patterna omogućava se nadogradnja klasnog modela u budućnosti te osigurava da se neće morati vršiti određene promjene u postojećim klasama.
U sistem je moguće dodati Bridge pattern za generisanje izvještaja o treninzima. Apstrakcija izvještaja (TreningIzvjestaj) odvaja se od konkretnih formata (PDF, tekstualni, prikaz u aplikaciji), čime se omogućava lako proširivanje sistema bez izmjene postojećih klasa.

5. Composite pattern

Composite pattern služi za kreiranje hijerarhije objekata. Koristi se kada svi objekti imaju različite implementacije nekih metoda, no potrebno im je svima pristupati na isti način, te se na taj način pojednostavljuje njihova implementacija.
U sistem se može dodati Composite pattern za upravljanje trening vježbama. Trening može da se sastoji od više jednostavnih vježbi (npr. zagrijavanje, jahanje u krug, preskakanje prepona), ali i složenih vježbi koje se sastoje od više podvježbi. Svaka vježba implementira zajednički interfejs, bez obzira da li je jednostavna ili složena (sastavljena od drugih vježbi). Na taj način se omogućava fleksibilno kreiranje treninga i pozivanje akcija nad cjelinom bez brige o unutrašnjoj strukturi.

6. Proxy pattern

Proxy pattern služi za dodatno osiguravanje objekata od pogrešne ili zlonamjerne upotrebe. Primjenom ovog patterna omogućava se kontrola pristupa objektima, te se onemogućava manipulacija objektima ukoliko neki uslov nije ispunjen, odnosno ukoliko korisnik nema prava pristupa traženom objektu.
U sistem se može dodati Proxy pattern za kontrolisani pristup korisničkim podacima. Umjesto da se svi dijelovi sistema direktno povezuju sa User objektom, koristi se UserProxy koji provjerava prava pristupa na osnovu uloge korisnika. Na ovaj način se štite osjetljive informacije i sprječava neautorizovan pristup podacima.