LISTA UŻYTYCH WZORCÓW

WYDARZENIA MUZYCZNE

	Nazwa wzorca	Klasy	Opis/Uzasadnienie
1.	Single Table Inheritance	Account, AdminAccount, UserAccount, OrganizerAccount + tabela BD: Users	Odwzorowanie hierarchii dziedziczenia w bazie danych. W zaprojektowanym systemie nie przewiduje się, że tabela „urośnie" do ogromnych rozmiarów – każdy typ użytkownika dodaje maksymalnie po jednym polu do klasy bazowej. W przyszłości Organizator mógłby być połączony z konkretną organizacją, ale nie łączyłoby się to z większą ilością kolumn w tabeli, a zastąpieniem kolumny *organization_name* odnośnikiem do konkretnej organizacji (tabeli). Struktura dziedziczenia raczej prosta, więc architektura nie wymaga skomplikowanego rozwiązania.
2.	Identity Field	np. Account, Order, Ticket, Event (praktycznie wszystkie encje)	Zapewnienie utrzymania tożsamości obiektu aplikacji i wiersza bazy danych (koniecznie przy transferze obiekt → BD). Ponadto jest to wymagane przez zastosowaną technologię – użyte wbudowane mechanizmy Spring Boota wymagają odwzorowania wszystkich kolumn tabeli (np. do zapisu danych).
3.	Foreign Table Mapping	np. (Ticket, Order, Event) lub (Order, UserAccount) + odpowiadające im tabele BD	Odwzorowanie zależności między obiektami poprzez klucz obcy w tabeli.
4.	Association Table Mapping	Account, Role + tabele: Users, Roles User_roles,	Odwzorowanie asocjacji *wiele-do-wielu* w bazie danych. W systemie każda z ról nadaje użytkownikowi pewne uprawnienia, ich liczba nie jest ograniczona do 1 per użytkownik. Aktualnie istnieją 3 role, myląco odpowiadające typom użytkowników (jednakże już teraz odnoszące się do innych przywilejów, np. organizator powinien posiadać 2 role: użytkownika i organizatora), ale w miarę rozrastania się sytemu ich liczba może rosnąć i odpowiadać za niezależne od typu konta funkcjonalności/dostępy.
4.	Repository	np. EventRepository, OrderRepository	Wprowadzenie warstwy abstrakcji między dziedziną a warstwą odpowiedzialną za generowanie zapytań do BD. W projekcie wszelkie interfejsy typu Repository rozszerzają wbudowane interfejsy JpaRepository/CrudRepository z zaimplementowanymi już metodami dot. komunikacji z bazą danych.

5.	Service Layer	np. TicketService, TicketServiceImp, OfferService, OfferServiceImpl	Warstwa, która określa zbiór dostępnych operacji (i tym samym je hermetyzuje), koordynuje odpowiedzi aplikacji dla każdej z nich, w tym wykonuje niezbędne operacje związane z przetwarzaniem danych odebranych z warstwy kontrolerów (i analogicznie w drugą stronę).
6.	Transaction Script	TicketService z metodami: -calculateOrderSummary() -bookTickets() -confirmPayment()	Prosta logika systemu - uporządkowanie jej według wykonywanych w systemie transakcji.
7.	Data Transfer Object	np. OfferDTO – łączy Offer z OrganizerAccount	Dostosowanie przesyłanych informacji do potrzeb aplikacji frontendowej za pomocą jednego obiektu zamiast dwóch i tym samym zmniejszenie przesyłanych danych do tylko tych niezbędnych.
8.	Mapper	np. OfferMapper	Mapuje obiekt klasy Offer na OfferDTO izolując tym logikę mapowania jednego obiektu w drugi.
9.	Page Controller	np. EventController	Obiekty obsługujące żądania skierowane do każdej logicznej strony, pośredniczące między GUI a Service Layer.
10.	Model View Controller (MVP, MVVM)		Przykładem wzorca MVC może być komunikacja Aplikacji fontendowej (View) z klasami typu Controller aplikacji backendowej (Presenterem), modelem zaś pozostała cześć backendu odpowiedzialna za logikę aplikacji. Ponadto, już w samym frontendzie (zaimplementowanym w Angularze) odnajdziemy wzorzec MVVM – mechanizmy data bindingu (widokiem jest wyświetlana strona, kontrolerem klasa typu Component odpowiadająca za ewentualną zmianę widoku i jednocześnie komunikująca się z modelem – w tym przypadku klasy typu service).
11.	Template View	ForRolesDirective	Na podstawie znacznika html *forRoles i przekazanej mu listy roli, podejmowana jest decyzja czy danemu użytkownikowi (z określonym zestawem roli w systemie) mają zostać wyświetlone oznaczone obiekty.