Proszę opisać w poniższej tabeli 5 najistotniejszych wzorców zastosowanych w projekcie.

Link do repozytorium: https://github.com/jan-osch/localmessages.server

|  |  |  |
| --- | --- | --- |
| nazwa wzorca (pol. i ang.) | lokalizacja (nazwa pliku, klasy, metody, numer linii itp.);  wyszczególnienie elementów wzorca (co jest czym) | Motywacja i konsekwencje zastosowania wzorca, tzn. co daje, jakie są elastyczności, a jakie wady, ograniczenia i zagrożenia. Proszę pisać konkretnie o danej sytuacji, a nie o wzorcu ogólnie. Jeśli nie ma żadnych widocznych wad w danej sytuacji, to nie należy na siłę ich wymyślać. [30-60 słów] |
| Singleton | **Klasa:**  PostgresConnectionManager  **Klienci:**  DAO i Komendy na bazie danych | Klasa która odpowiada za dostarczanie i przechowywanie połączenia do bazy danych. Ponieważ otwieranie połączeń do bazy danych jest kosztowne, należy ograniczyć ich ilość. Zastosowanie singletona pozwala na użycie pojedynczego połączenia w wielu miejscach.  W przyszłości singleton może przechowywać więcej połączeń i używać algorytmu karuzelowego przy przekazywaniu ich do klientów. |
| Strategy (Strategia) | **Interfejs:** GetMessagesByLocationStrategy  **Implementacje:**  GetMessagesCompositeStrategy  GetPrivateMessagesByDistance  GetPublicMessagesByDistance  **Klienci:**  Kontrolery | Algorytm pobierania wiadomości dla zadanej lokalizacji. Zastosowanie strategii pozwala na zmianę parametrów (np. dystans) algorytmu bez konieczności rekompilacji całego kodu.  Wady: niestety duplikacja kodu (wielość parametrów) |
| Composite  (Kompozyt) | **Klasa:** GetMessagesCompositeStrategy  **Implementuje:**  GetMessagesByLocationStrategy  **Klienci:**  Kontrolery | Dzięki zastosowaniu kompozytu połączonego ze strategią można w sposób przeźroczysty dla klientów połączyć ze sobą kilka strategii – w tym przypadku można połączyć znajdowanie wiadomości publicznych i prywatnych(dla różnych dystansów) w jeden algorytm. |
| Abstract Factory  (Fabryka abstrakcyjna) | **Interfejs:**  DAOFactory  **Implementacja:**  PostgresDAOFactory  **Klienci:**  Kontrolery | Fabryka abstrakcyjna przekazywana jest to kontrolerów z funkcji main. W ten sposób klasy klienckie nie znają szczegółów implementacyjnych jakich obiektów DAO używają. W przyszłości pozwala na np. na zastosowanie różnych baz danych równocześnie.  Wady: ciężko połączyć z singletonem, uniemożliwia efektywne cachowanie na poziomie instancji DAO. |
| Data Access Object (DAO) | **Interfejsy:**  MessageDAO  UserDAO  **Implementacje:**  PostgresMessagesDAO  PostgresUserDAO  **Klienci (pośrednio):**  Kontrolery | Zastosowanie wzorca pozwala na ukrycie szczegółów implementacji dostępu do danych, zapewnia jednolity interfejs dla klientów, zapewnia możliwość ponownego wykorzystania istniejących metod. Daje możliwość zmiany szczegółów bez rekompilacji klientów. |
| Command  (Komenda) | **Interfejs:**  Command  **Implementacje:**  CreateMessagesTableCommand  CreateUsersTableCommand  **Klienci:**  Main | W komendach zostały zapisane procedury tworzenia tabel w bazie danych. Umożliwia to wywołanie danej procedury z poziomu startującej aplikacji albo wywołanie np. z wiersza poleceń. |