**Sprawozdanie z Analizy Projektu Systemu "Marketplace"**

**1. Wstęp**

Niniejsze sprawozdanie przedstawia analizę projektu diagramu klas UML systemu "Marketplace". Celem analizy jest identyfikacja kluczowych komponentów, relacji, zastosowanych wzorców projektowych oraz ocena projektu pod kątem wybranych zasad dobrego projektowania oprogramowania i potencjalnych antywzorców.

**2. Opis Diagramu UML**

Przedstawiony diagram UML opisuje architekturę systemu typu marketplace. Główne zidentyfikowane moduły funkcjonalne obejmują zarządzanie użytkownikami (w tym różne typy użytkowników jak AdminUser, SellerUser, BuyerUser oraz ich tworzenie za pomocą fabryk), zarządzanie przedmiotami (Item), obsługę transakcji (Transaction) oraz spory (Dispute). Centralnym punktem interakcji z systemem wydaje się być klasa MarketplaceFacade, upraszczająca dostęp do jego funkcjonalności.

W projekcie zidentyfikowano zastosowanie następujących wzorców projektowych:

* **Fasada (Facade):** Klasa MarketplaceFacade.
* **Obserwator (Observer):** Relacja między User (jako Subject) a MarketplaceFacade (jako Observer).
* **Polecenie (Command):** Interfejs Command i klasa MakeOfferCommand.
* **Fabryka (Factory):** Interfejs UserFactory i jego implementacje (AdminFactory, SellerFactory, BuyerFactory).
* **Budowniczy (Builder):** Interfejs DisputeBuilder i klasa ConcreteDisputeBuilder.
* **Pełnomocnik (Proxy):** Klasa ItemProxy dla ItemRepository.

**3. Analiza Projektu**

Poniżej przedstawiono pięciopunktową analizę wybranych aspektów projektu:

* **3.1. Zasada Pojedynczej Odpowiedzialności (SRP) a klasa MarketplaceFacade**
  + *Zasada:* Klasa powinna mieć tylko jedną odpowiedzialność (jeden powód do zmiany).
  + *Obserwacja:* MarketplaceFacade agreguje funkcje zarządzania użytkownikami, przedmiotami, transakcjami oraz działa jako Obserwator. Może to prowadzić do naruszenia SRP, czyniąc klasę trudniejszą w utrzymaniu.
  + *Sugestia:* Rozważenie delegacji poszczególnych odpowiedzialności do mniejszych, wyspecjalizowanych serwisów, którymi MarketplaceFacade by zarządzała.
* **3.2. Zasada Otwarte/Zamknięte (OCP) a hierarchia UserFactory**
  + *Zasada:* Komponenty powinny być otwarte na rozszerzenia, ale zamknięte na modyfikacje.
  + *Obserwacja:* Użycie interfejsu UserFactory i jego konkretnych implementacji (AdminFactory, SellerFactory, BuyerFactory) dobrze wspiera OCP. Dodanie nowego typu użytkownika i odpowiadającej mu fabryki nie powinno wymagać modyfikacji istniejącego kodu fabryk.
  + *Pozytywne:* Elastyczne podejście ułatwiające rozbudowę systemu.
* **3.3. Pryncypium Kreator (Creator - GRASP) a Fabryki Użytkowników**
  + *Pryncypium:* Odpowiedzialność za tworzenie obiektu powinna być przypisana klasie, która m.in. ściśle używa tworzonego obiektu lub posiada dane inicjalizujące.
  + *Obserwacja:* Klasy AdminFactory, SellerFactory, BuyerFactory są odpowiedzialne za tworzenie odpowiednich typów użytkowników, co jest zgodne z pryncypium Kreatora, hermetyzując logikę tworzenia.
  + *Pozytywne:* Zwiększa spójność i zmniejsza sprzężenie.
* **3.4. Pryncypium Wysokiej Spójności (High Cohesion - GRASP) a DisputeBuilder**
  + *Pryncypium:* Klasa powinna mieć wąsko zdefiniowany zestaw silnie powiązanych odpowiedzialności.
  + *Obserwacja:* Interfejs DisputeBuilder i jego implementacja ConcreteDisputeBuilder koncentrują się wyłącznie na konstrukcji obiektów Dispute.
  + *Pozytywne:* Wysoka spójność czyni te klasy łatwiejszymi do zrozumienia, utrzymania i ponownego użycia.
* **3.5. Antywzorzec: Przepływ Lawy (Lava Flow) a atrybuty w klasie Item**
  + *Antywzorzec:* Obecność w systemie "zastygłego" kodu, który jest martwy lub nieużywany, ale nieusuwany.
  + *Obserwacja:* Klasa Item zawiera atrybuty w języku angielskim oraz ich polskie odpowiedniki. Jeśli polskie wersje są nieużywanymi pozostałościami, mogą stanowić "przepływ lawy".
  + *Sugestia:* Weryfikacja konieczności utrzymywania zduplikowanych atrybutów i usunięcie nieużywanych w celu uproszczenia modelu i uniknięcia niespójności.

**4. Podsumowanie**

Analizowany projekt systemu "Marketplace" wykorzystuje szereg dobrych praktyk i wzorców projektowych, które wspierają jego elastyczność i modularność (np. Fabryka, Budowniczy, Obserwator). Zidentyfikowano jednak obszary do potencjalnej poprawy, takie jak możliwe naruszenie zasady SRP w klasie MarketplaceFacade oraz potrzeba weryfikacji zduplikowanych atrybutów w klasie Item. Dalsze prace powinny uwzględnić te obserwacje w celu zwiększenia jakości i łatwości utrzymania systemu.