Testowanie Mikroserwisów
Consumer-driven contract testing

Wprowadzenie

Na potrzeby działalności firmy, wprowadzona będzie sprzedaż online produktów. Aby w pełni zautomatyzować proces, operacja zamówienia powinna również obsługiwać nadanie przesyłki (paczki) do klienta (obsługa firmy kurierskiej). Ponieważ inne działy (jak np. dział serwisu i reklamacji) w przyszłości również chciałby mieć zautomatyzowany proces nadawania przesyłki, zdecydowano się na implementację dedykowanego serwisu do obsługi firm kurierskich. W celu skrócenia czasu potrzebnego na uruchomienie kompletnego rozwiązania powołano dwa niezależne zespoły:

Zespół A – Będzie pracować nad głównym serwisem przyjmującym zamówienia (Order Service). Zadaniem usługi będzie koordynacja procesu zamówienia: odbiór danych od aplikacji klienckiej, kontrola stanów magazynowych, archiwizacja zamówień (na potrzeby ćwiczenia ta część implementacji została pominięta) i oczywiście zamówienie nadania paczki poprzez drugi serwis.

Zespół B – Skupiać się będzie nad implementacją usługi (Shipment Service) koordynującej proces wysyłania paczek do klientów, czyli zamawianiem przesyłek kurierskich (łączenie się z odpowiednimi dostawcami, itd.).

Zespół pierwszy nie mógł czekać aż drugi zespół w pełni zakończy implementację Shipment Service. Dlatego wstępnie uzgodniono, jak ma wyglądać komunikacja pomiędzy serwisami. Oba zespoły rozpoczęły pracę…

Testowanie konsumenta serwisu (service consumer tests)

Pobierz źródła: https://github.com/TomWeps/NIP3-Orders
Otwórz solucję NIP3.Orders. Zawiera ona implementację serwisu do składania zamówień wraz z testami

1. Uruchom projekt NIP.Orders w trybie debug (F5), następnie w oknie przeglądarki otwórz url: http://localhost:1994/swagger (UI Swagger).
   W klasie `OrderController` ustaw 'breakpoint' na początku metody `CreateOrder`.
   W oknie przeglądarki wykonaj operację zamówienia produktu (operacja POST).
   Prześledź działanie programu oraz zapoznaj się z implementacją.
   Uwaga: serwer zgłosi błąd, jest to poprawne na tym etapie zadania.

2. Klasa `ShipmentProxy` jest odpowiedzialna za całą komunikację (wraz z odpowiednią obsługą błędów) z usługą Shipment Service.
   Dokończ implementację metody `GetShipmentResponseData`.
   - Zweryfikuj HttpStatus odpowiedzi (response) przysłane przez Shipment Service
   - Jeśli operacja zakończyła się sukcesem, odczytaj przesłane dane (json) i zwróć je przy pomocy klasy `ShipmentResponse` (response.Content.ReadAsAsync<ShipmentResponse>).
   - W przypadku zasygnalizowania błędu przez ShipmentService (np. gdy dane adresowe są niepoprawne), metoda powinna wyrzucić wyjątek `ShipmentException`.
   Pomocne linki:
   Opis http statusów - https://restfulapi.net/http-status-codes/
   Przykład użycia klasy HttpClient (zapytania REST) - https://docs.microsoft.com/en-us/aspnet/web-api/overview/advanced/calling-a-web-api-from-a-net-client

3. Implementacja usługi OrderService jest gotowa. Programu nie można jeszcze uruchomić, ponieważ nie można się połączyć z niezaimplementowanym serwisem do obsługi przesyłek (ShipmentService). W celu weryfikacji poprawności działania serwisu zostały napisane testy, które używają tzw. zaślepek (mocks) symulujących zewnętrzne usługi.
   Zapoznaj się z implementacją NIP3.Orders.Test.
   Zwróć szczególną uwagę na plik Pacts\ordersservice-shipmentservice.json (powinien być pusty na początku) oraz klasę OrderControllerTest.

4. W każdym z testów z klasy OrderControllerTest dokończ implementację zaślepek usługi nadawania przesyłek (shipmentMockProviderService).
   Po zakończeniu zadania, wszystkie testy powinny zakończyć się wynikiem prawidłowym (pass).

   Sprawdź zawartość pliku Pacts\ordersservice-shipmentservice.json, co on zawiera?