Zadanie zaliczeniowe z przedmiotu

Rozproszone przetwarzanie danych w języku Scala

Historia wersji

Wersja	Data	Zmiany
1.0.0	20.05.2021	1. Przekazanie studentom
1.0.1	24.05.2021	1. Zebranie odpowiedzi na pytania, z 21.05.2021, odnośnie oceniania

Słowo wstępne

Poniższy dokument zawiera informacje odnoszące się do wytycznych do realizacji zadania zaliczeniowego z przedmiotu “Rozproszone przetwarzanie danych w języku Scala”, które powinno być zadaniem grupowym, przygotowanym w grupach 2 i 3 osobowych. Zadanie powinno być wykonane w języku programowania Scala oraz dostarczone w określonym czasie i repozytorium Git.

Cel zadania

Celem pierwotnym zadania jest sprawdzenie umiejętności studentów nie tylko pod względem programowania, ale także jeżeli chodzi o umiejętności miękkie (umiejętność współpracy w grupie, efektywność komunikacji), a także umiejętności analityczne oraz trening przelania myśli w produkty (kod źródłowy, dokumentacja, plik Readme, diagramy/schematy), co jest podstawą do przyszłej pracy z systemami rozproszonymi.

Ze względu na ograniczony czas oraz potencjalną złożoność rozwiązań w przypadku typowo rozproszonych zagadnień postawiono na wykonanie prostej aplikacji typu REST API osadzonej w bardzo uproszczonej domenie sklepu internetowego. Aplikacja powinna posiadać odpowiednio ustrukturyzowany kod, implementować znane zespołowi dobre wzorce programowania (np. odporność na podstawowe ataki SQL Injection) oraz dostarczyć niezbędnej dokumentacji do uruchomienia oraz pracy ze stworzonym systemem. Poniżej znajdują się lepiej zdefiniowane wymagania.

Wymagania funkcjonalne

1. Możliwość dodania, usunięcia i edycji produktów (funkcjonalność administratora, uproszczona)
2. Możliwość zarządzania kontami klientów (funkcjonalność administratora, uproszczona)
3. Możliwość dodania, usunięcia i edycji produktów z koszyka (funkcjonalność klienta, uproszczona)
4. Finalizowanie zakupu (uproszczone)
5. Endpointy zabezpieczone tokenem Bearer (wystarczające jest użycie zahardcodowanych tokenów, nie trzeba implementować endpointu authorize z OAuth)
6. Aplikacja powinna zachowywać stan, poprawnie się inicjalizować i wygaszać
7. Aplikacja powinna odpowiednio logować zdarzenia w systemie

Wymagania niefunkcjonalne

1. Aplikacja powinna być podzielona zgodnie z poznanymi praktykami oraz umożliwiać łatwe podmianę implementacji poszczególnych elementów
2. Aplikacja powinna być napisana zgodnie z podstawowymi praktykami języka Scala, praktykami znanymi zespołowi wykonującemu oraz tzw. clean code. Podstawowymi technologiami aplikacji powinny być Akka HTTP, Future, Slick, Postgresql, Scalatest. W przypadku potrzeby powinno się skorzystać z ZIO jako systemu efektów
3. Aplikacja powinna działać poprawnie
4. Aplikacja powinna mieć dokumentację pozwalającą na podstawową interakcję, diagram encji(przygotowany w formie cyfrowej za pomocą dostępnych narzędzi), jak również dokumentację kodu, m.in wszystkie interfejsy.
5. Aplikacja powinna dostarczyć dokumentację API (użyć OpenApi, wystarczy wcommitować sam plik definicji)
6. Aplikacja powinna być wersjonowana w repozytorium GIT, w sposób określony na pierwszych zajęciach
7. Aplikacja powinna być budowana w prosty sposób, wraz z zapewnieniem jakości
8. Aplikacja powinna być zdockeryzowana (wraz z zależnościami), tak aby w prosty sposób można było sprawdzić jej działanie.
9. Podstawowa konfiguracja powinna mieć możliwość zmiany w prosty sposób
10. Funkcjonalność powinna być przetestowana testami e2e oraz unit/integracja

Wymagania dotyczące sposobu wykonania

1. Projekt jest projektem grupowym, w którym oceny poszczególnych członków zespołu projektowego mogą się od siebie różnić, toteż:
    a. Każda z osób powinna mieć osobny wkład w realizację poprzez podział na zadania niżej wymienione, co nie wyklucza, a wręcz wymaga współpracy na poziomie koncepcyjnym. W każdym z w/w zakresów powinien być jeden główny kontrybutor w celu ułatwienia procesu oceny.
        i. Inicjalizacja projektu (tutaj pracę można podzielić proporcjonalnie na cały zespół)
            1. sbt, scalafmt dependencies, startup serwera
            2. podstawowe testy e2e
            3. dockeryzacja
        ii. Baza danych
            1. Repozytoria
            2. Przygotowanie obiektów bazodanowych
            3. Odpowiedź z bazy powinna być opracowana w ten sposób, aby nie uzewnętrzniać typów ze Slicka (np. Either zamiast DBAction)
            4. Diagram ERD wykonany w odpowiednim narzędziu
        iii. API
            1. Implementacja Routes
            2. Przygotowanie exception handlera, tak aby nie uzewnętrzniać potencjalnych exceptionów
            3. Przygotowanie autoryzacji (admin/user, mogą posiadać stałe tokeny)
            4. Przygotowanie ser/deserializatorów oraz DTO
            5. Schemat OpenAPI
        iv. Serwisy aplikacyjne
            1. Implementacja serwisów aplikacyjnych (warstwa logiki między API a bazą danych) zarządzająca procesami biznesowymi
            2. Powinna przyjmować obiekty domenowe, wykonywać operacje biznesowe i zapisywać/odczytywać dane z bazy
        v. Ponadto jako zespół należy :
            1. Dokonać analizy domeny, zaprojektować API, rozdzielić strukturę tak, aby móc pracować równolegle
            2. Dostarczyć oprócz w/w dokumentacji informację w pliku Readme o członkach zespołu (numery indeksów, imię, nazwisko, nickname użyty w repozytorium w celu możliwości oceny)

Odpowiedzi na pytania studentów:

1. Jak będzie wyglądało ocenianie?
    a. Ocenianie będzie proporcjonalne, wyglądające podobnie do typowego odbioru zadania w projekcie IT, rozłożone na 2 mniejsze cząstki.
    W obu częściach możliwe jest uzyskanie oceny progowej, mimo braku idealnego rozwiązania (w tym przypadku decyzja będzie leżeć w gestii sprawdzających)
    b. Na uzyskanie 0,5 liczyć się będzie (minimum projektowe):
        i. poprawność działania kodu (przede wszystkim stabilność i poprawność działania, widoczna obsługa sytuacji wyjątkowych, brak nulli czy rzucania wyjątkami - brak użycia throw, brak zapaści serwera)
        ii. Podstawowy prawidłowy podział na klasy i pakiety, zbudowanie elementów subdomen
        iii. przyzwoita czytelność kodu (np. poprawne nazwanie zmiennych zamiast “aaa”, formatowanie kodu itp.)
        iv. przynajmniej częściowe testy unitowe/integracyjne
        v. Dockeryzacja, setup
    c. Pozostałe 0,5 oceny można uzyskać za pozostałe aspekty zadania m.in za:
        i. poprawne praktyki programowania m.in. elementy DDD, bardzo dobry podział pakietów i klas
        ii. pokrycie kodu poprawnymi testami
        iii. dokumentacja (Readme, diagramy itp.)
        iv. dające się odczuć przemyślenie tematu i napisanie projektu z głową i dbałością
        v. Najważniejsze testy e2e (punkt opcjonalny, ale wpływa pozytywnie w przypadku braków)