Akademia Górniczo-Hutnicza

Wydział Informatyki, Elektroniki i Telekomunikacji

Księgarnia internetowa

Dokumentacja techniczna

Igor Dzierwa
Konrad Makuch
Adrian Nędza

1. Analiza zadania

Celem projektu jest stworzenie systemu backendowego realizującego zadania obsługi księgarni internetowej.

2. Aktorzy

1. Administrator
2. Użytkownik niezarejestrowany
3. Użytkownik zarejestrowany
4. Pracownik

3. Wymagania funkcjonalne

Funkcje systemu:

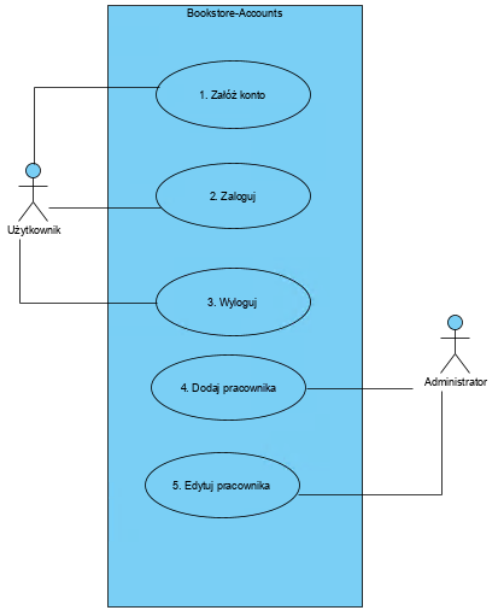
1. Rejestracja użytkowników – metoda restapi umożliwiająca rejestrację użytkownika
2. Dodawanie zasobów (książek do katalogu) – metoda restapi umożliwiająca dodanie nowej książki
3. Dodawanie autorów książek – metoda restapi umozliwiająca dodanie autora książki
4. Przeglądanie zasobów – metoda restapi zwracająca listę książek w bazie danych
5. Funkcja koszyka – tworzenie oraz pamiętanie koszyka przypisanego do użytkownika, obliczanie wartości koszyka
6. Składanie zamówień – tworzenie nowego zamówienia w systemie z informacją o jego wartości, produktach, użytkowniky który je złożył.
7. Wyświetlanie zamówień

4. Wymagania niefunkcjonalne

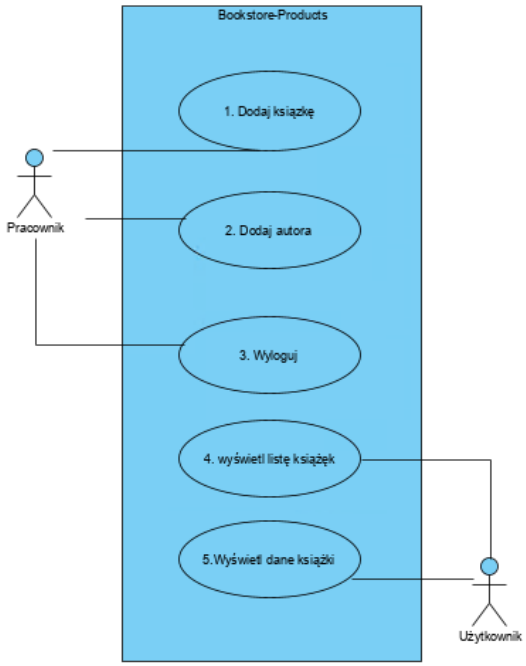
1. System oparty o architekturę mikroserwisową
2. Łatwa skalowalność systemu
3. Konteneryzacja
4. Szyfrowanie danych wrażliwych (np. hasła)
5. Obsługa cen z dokładnością do dwóch miejsc po przecinku

5. Diagramy przypadków użycia

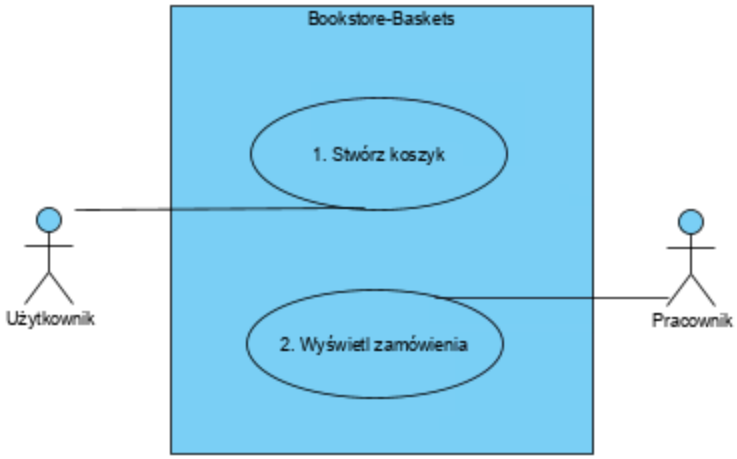
Prezentację diagramów przypadków użycia powiązaliśmy z mikrosrwisem, który jest bezpośrednio związany z daną funckją.



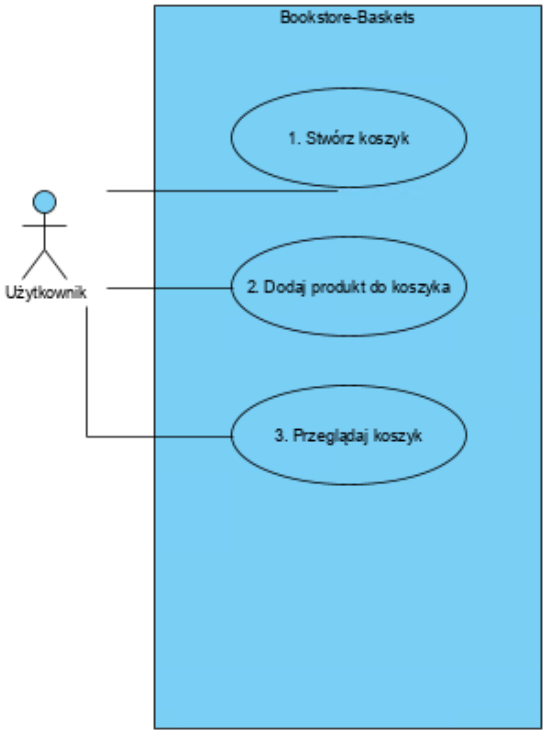
Rysunek 1 - Diagram przypadków użycia w obszasze mikroserwisu Accounts



Rysunek 2 - Diagram przypadków użycia w obszasze mikroserwisu Products



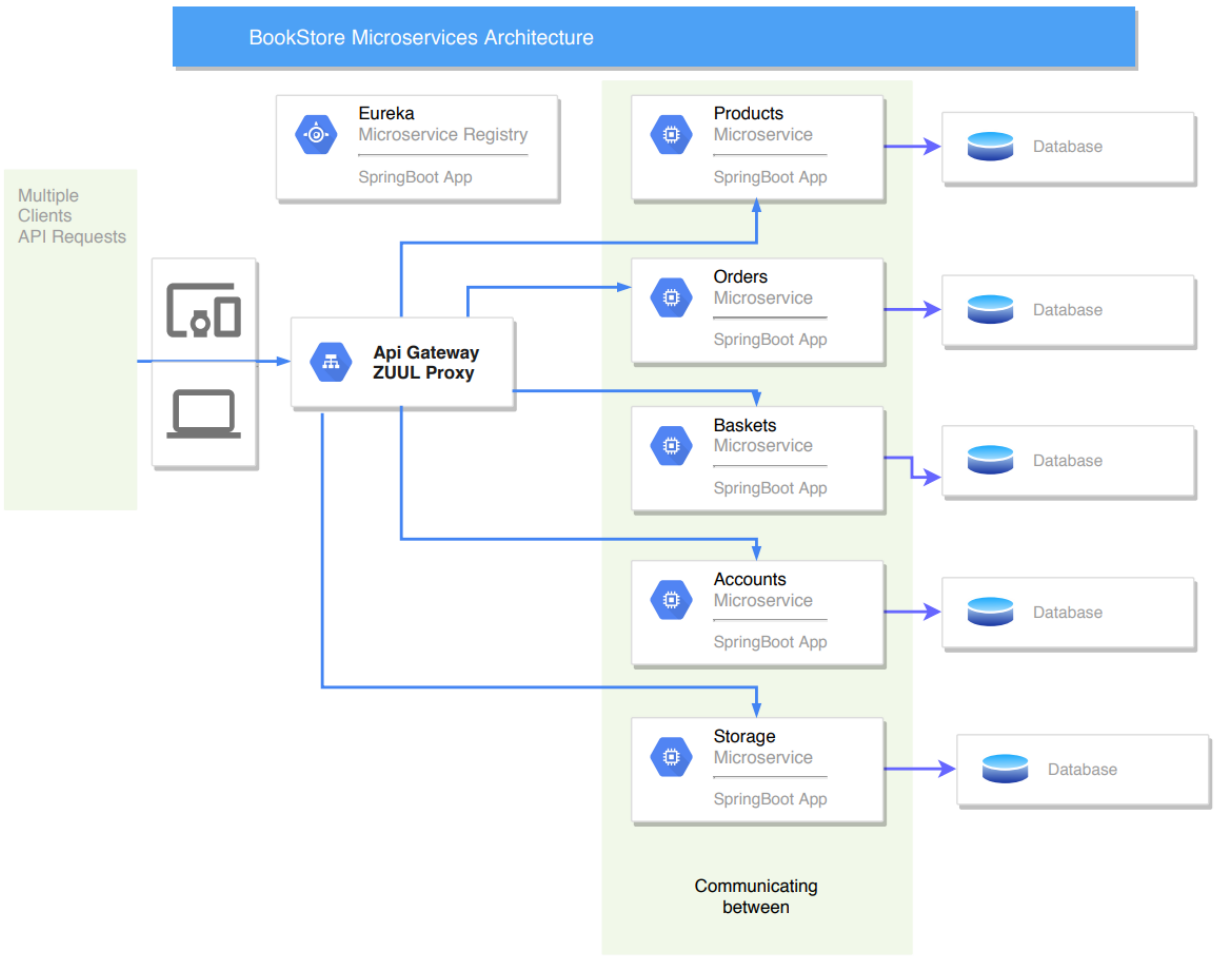
Rysunek 3 - Diagram przypadków użycia w obszasze mikroserwisu Orders



Rysunek 4 - Diagram przypadków użycia w obszasze mikroserwisu Baskets

6. Architektura systemu

Architektura systemu została oparta o paradygmat mikroserwisowy. Mikroserwisy komunikują się wewnątrz pomiędzy innymi mikroserwisami z wykorzystaniem FeignClient. Wyodrębnione zostały następujące mikroserwisy:



1. Bookstore-Gateway

Odpowiadający za przekazywanie zapytań do odpowiednich mikroserwisów (Zuul Proxy) oraz filtrowanie ich na podstawie uprawnień (np. Użytkownik nie może mieć dostępu do metody dodającej książki). W tym mikroserwisie została również zaimplementowana automatyczna dokumentacja wszystkich endpointów dostępnych w systemie (Swagger UI).

2. Bookstore-Products

Mikroserwis realizujący logikę związaną z zasobami dostępnymi w systemie – dodawanie książek, dodawanie autorów, prezentacja dostępnych zasobów.

3. Bookstore-Orders

Obsługa składania zamówień. Rejestr zamówień.

4. Bookstore-Baskets

Obsługuję logikę związaną z koszykiem – zapamiętywanie zawartości koszyka, obliczanie jego wartości załkowitej, przypisywanie koszyka do użytkownika na podstawie Cookie.

5. Bookstore-Accounts

Obsługuję rejestrację użytkowników oraz mechanizm logowania do systemu (JWT Token)

6. Bookstore-Storage

Mikroserwis który obsługuję zapis i prezentacje plików (np. Zdjęć autorów lub okładek książek)

7. Eureka-Service

Mikroserwis pełniący funkcję rejestru dostępnych mikroserwisów.

7. Testy

Zaimplementowane testy oparte są o javascriptowy framework Cypress.
Jest to narzędzie przeznaczone do pisania testów end to end (E2E) zarówno po stronie warstwy frontowej, jak również samego API.

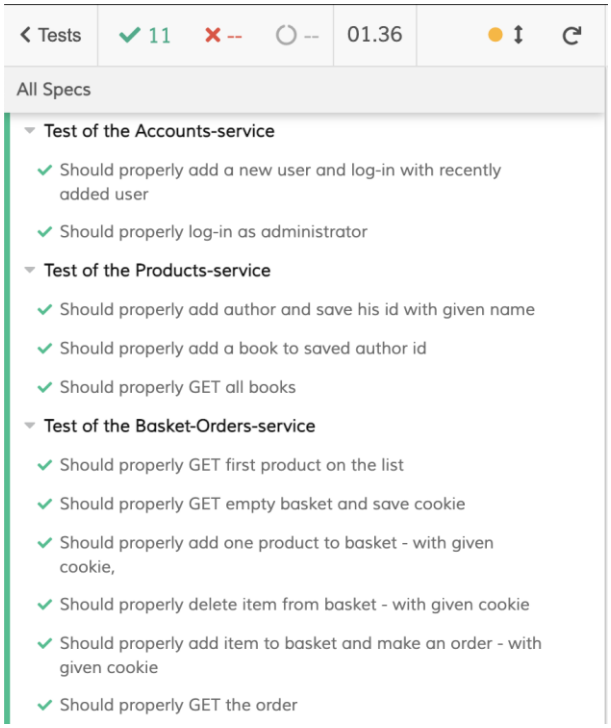
Wybraliśmy takie rozwiązanie, ponieważ w przypadku rozproszonej aplikacji jest mało elementów nadających się bezpośrednio do jednostkowego testowania w Javie.
Dodatkowo, Cypress pozwala na wygodne przejrzenie wyników testów po stronie przeglądarki, co jest o wiele wygodniejsze od bazowania na zwracanych wynikach w konsoli.

Testy oparte są na 3 plikach, które skupiają się wokół głównych serwisów aplikacji:

- **Accounts-service/Accounts.spec.js -** testy serwisu accounts, które obejmują: rejestrację użytkownika, jego zalogowanie oraz sprawdzenie konta administratora.

- **Products-service/Products.spec.js -** testy serwisu products, które obejmują: dodanie autora, dodanie książki oraz prezentację wszystkich zasobów.

- **Baskets-Orders-service/Baskets-Orders.spec.js -** kompleksowe testy serwisów baskets oraz orders, które obejmują: pobranie książki, utworzenie koszyka oraz dodanie jej do niego, usunięcie książki z koszyka i ponowne jej dodanie, by w ostateczności złożyć i pobrać zamówienie.



W celu lokalnego uruchomienia testów należy:

- pobrać potrzebne paczki - **npm install**
- uruchomić cypress - **npm run open**

8. Instrukcja „przejścia” po wszystkich kluczowych metodac/h RestAPI systemu

Po uruchomieniu wszystkich serwisów (kontenetów) system jest gotowy do wykonania m.in. takich zapytań RestAPI:

1. Dodawanie autora

POST /products-service/admin/author FormData

- name
- file
- description
- year

2. Dodawanie książki

POST /products-service/admin/book FormData

- name
- author.id
- file
- length
- description
- stock (ilosc w magazynie)
- price

3. Stworzenie koszyka (hipotetycznie jeśli system by działał to przy pierwszym otwarciu aplikacji wykonywalibysmy takie zapytanie o koszyk) - tworzy się w tym momenecie koszyk i dopisuje się Cookie z Id koszyka, które automatycznie jest dodawane do kolejnych requestów. Jeśli mamy coś w koszyku to ta metoda zwraca aktualny stan koszyka - produkty + sumę.

GET /baskets-service/cart

4. Dodawanie do koszyka (książki)

POST /baskets-service/cart/{ID PRODUKTU}

5. Odczytanie koszyka albo usunięcie produktu - **DELETE /baskets-service/cart/{id produktu do usunięcia}** tutaj ważne - nie podajemy ID produktu z products-service tylko z baskets-service - Itemy w koszyku mają swoje ID i niekoniecznie jest takie samo jak ID produktu.

6. Złożenie zamówienia **POST /orders-service/order/** i po wykonaniu tej metody z zapisanym cookie:

- sprawdzają się aktualne stany magazynowe – warunek do złożenia zamówienia
- zapisuje się zamówienie
- jeśli wykonamy tą metodę z dodanym Bearerem - to doda się paramter Mode:"User Zalogowany" do Order i z accounts-service pobierze się ID zalogowanego użytkownika i też doda się do zamówienia.
- odejmują się stany magazynowe w products-service
- dodają sie orderProducts powiązane z tym zamówieniem - duplikaty obiektów w aktualnym stanie z products-service

9. Wykorzystane technologie

System został oparty o architekturę mikroserwisową, gdzie każdy mikroserwis to odrębna aplikacja SpringBoot napisana w języku JAVA 11. Oprócz tego został wykorzystany FeignClient służący do komunikacji pomiędzy mikroserwisami, ZUUL Proxy jako API Gateway, Eureka Server jako rejestr mikroserwisów. Oprócz tego, zaimplementowany został Swagger jako automatyczna dokumentacja metod RestAPI. Do monitoringu aplikacji wykorzystaliśmy stack Elastic, Logstash, Kibana oraz Zipkin. Całość aplikacji jest skonteneryzowana oraz przygotowana do deploy produkcyjnego (docker swarm).