Buy it

Spis treści

Projekt realizowany w ramach przedmiotu Projekt Zespołowy prowdzony przez Ośrodek Kształcenia Na Odległość Politechniki Warszawskiej. W ramach projektu zdecydowano się zrealizować uproszczoną implementację sklepu internetowego, sprzedającego obuwie.

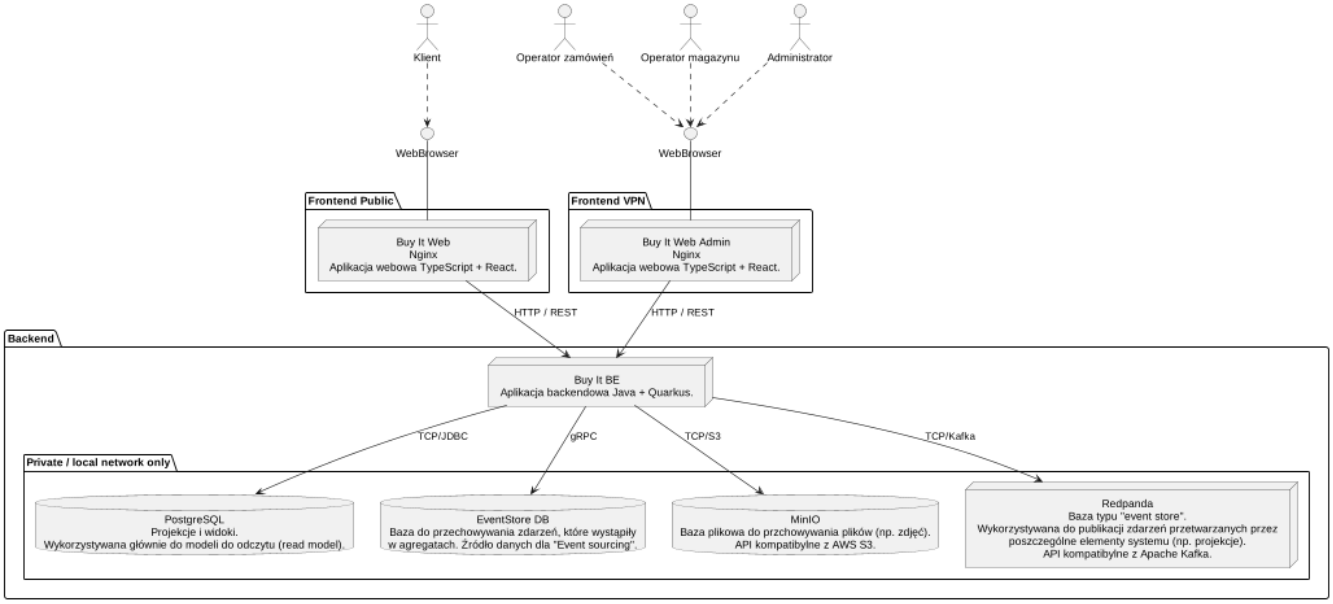
1. Architektura projektu

Projekt został podzielony na 2. części - 2. aplikacje backendowe oraz 1. frontendowa.

Aplikacja "Buy It Web" to ogólnodostępna, publiczna aplikacja sklepu internetowego. Klienci mogą za jej pomocą przeglądać i wyszukiwać produkty czy składać zamówienia.

Aplikacja "Buy It Web Admin" dostępna jest tylko dla zweryfikowanych pracowników sklepu, a połączenie sieciowe do aplikacji możliwe powinno być jedynie w ramach wydzielonej sieci VPN.

Na rysunku poniżej przedstawiono diagram komponentów wykorzystywanych przez aplikację.



Rysunek 1. Komponenty aplikacji Buy It - Level 2 modelu C4

Powyższy diagram przygotowany został z wykorzystniem C4 model - level 2.

1.1. Komponenty

1.1.1. Frontend

Obie aplikacje frontendowe to aplikacje Webowe serwowana jako pliki statyczne za pośrednictwem serwera Nginx.

Aplikacje Webowe zostały napisana z wykorzystaniem:

1. Języka TypeScript (kompilowanego do JavaScript).
2. Języka CSS / Saas SCSS.
3. Języka HTML.
4. Frameworka React.

Komunikacja między aplikacjami frontendowymi a aplikacją backendową została zrealizowana z wykorzystaniem HTTP z zastosowaniem REST (Richardson Maturity Level 2]).

1.1.1.1. Buy It Web

Publicznie dostępna aplikacja internetowego sklepu z obuwiem, której odbiorcami są klienci sklepu.

Aplikacja udostępnia takie funkcje jak:

1. Utworzenie i logowanie do konta klienta.
2. Wyszukiwanie produktów wraz z informacją o ich dostępności i wariantów (np. różnych rozmiarów czy kolorów).
3. Zarządzanie koszykiem zakupowym - dodawanie / usuwanie produktów.

4. Złożenie o opłacenia zamówienia na podstawie koszyka zakupowego.
5. Przegląd historii zamówień.

1.1.1.2. Buy It Web Admin

Aplikacja typu "backoffice" dla pracowników sklepu. Z założenia, niedostępna publicznie w sieci Internet.

Z aplikacji korzystać mogą pracownicy, posiadający różne role w aplikacji:

1. Administrator
2. Operator magazynu
3. Operator zamówień

1.1.1.2.1. Administrator

1. Wyszukiwanie i modyfikacja kont pracowniczyh. [Administrator]
2. Modyfikacja ról użytkowników.

1.1.1.2.2. Operator magazynu

1. Przeglądanie / wyszukiwanie produktów. [Operator magazynu]
2. Modyfikację produktów (dodawanie / usuwanie, edycja atrybutów - np. rozmiaru czy koloru). [Operator magazynu]

1.1.1.2.3. Operator zamówień

1. Przeglądanie / wyszukiwanie zamówień. [Operator zamówień]
2. Zmiana statusów zamówień (Złożone, Opłacone, W trakcie realizacji, Wysłane, Dostarczone). [Operator zamówień]

1.1.2. Backend

Aplikacja backendowa to aplikacja serwerowa, która zarządza stanem magazynów, ofertą, zamówieniami itp. Ta aplikacja jest głównym źródłem danych dla całego systemu i dba o zachowanie ich spójności.

W ramach aplikacji backendowej możemy wyróżnić takie komponenty jak:

1. Buy It BE - aplikacja serwerowa napisana w języku Java, która implementuje logikę biznesową systemu po stronie backendu. Udostępnia 2. rodzaje API HTTP dla klientów aplikacji webowych - osobne dla aplikacji dla klientów sklepu Buy It Web oraz osobne dla pracowników Buy It Web Admin.
2. EventStore DB - baza danych noSQL, przechowująca strumień zdarzeń domenowych, wykonanych / zaaplikowanych przez agregaty domenowe w architekturze Event Sourcing.
3. PostgreSQL - relacyjna baza danych, przechowująca dane do podglądu. Wygenerowane z użyciem projekcji na podstawie zdarzeń domenowych.

4. MinIO - baza obiektowa, przechowująca zdjęcia produktów czy logo marek. Posiada API kompatybilne z AWS S3.
5. Redpanda - broker zdarzeń, baza strumieniowa, służąca jako medium wymiany zdarzeń pomiędzy komponentami logiki biznesowej (np. agregatami domenowymi i projekcjami). Posiada API kompatybilne z Apache Kafka.

1.1.3. Buy It BE

Jest to główny komponent backendowy systemu. W tym komponencie zaimplementowana została logika biznesowa części backendowej sklepu.

Aplikacja została napisana z wykorzystaniem architektury Event Sourcing oraz elementami architektur CQRS i Heksagonalnej.

Wykorzystane języki i frameworki do utworzenia aplikacji to:

1. Java 17.
2. SQL.
3. Quarkus.
4. SmallRye Mutiny.

2. Aktualny stan projektu

2.1. Buy It BE

W ramach części backendowej projektu, aktualnie zrealizowano:

1. Architekturę aplikacji backendowej.
    - Przygotowanie abstrakcji Event Sourcing oraz CQRS - dla agregatów i zdarzeń domenowych, publikacji zdarzeń, projekcji, obsługi komend i zapytań.
    - Przygotowanie abstrakcji dla zapytań z wykorzystaniem Keyset Pagination
    - Podział aplikacji Quarkus na moduły Maven, starając się odwzorować architekturę Heksagonalną.
    - Konfiguracja testów integracyjnych Quarkus z wykorzystaniem Testcontainers.
2. Integrację z bazą EventStore DB.
3. Integrację z bazą PostgreSQL wraz z wykorzystaniem narzędzia migracji bazy danych [Liquibase].
4. Integrację z brokerem komunikatów / zdarzeń Redpanda.
5. Integrację z bazą MinIO.
6. Konfigurację środowiska lokalnego z wykorzystaniem Docker i docker compose.
7. Implementacja pierwszych usług do zarządzania produktem.
    - Proste wyszukiwanie produktu po podstawowych atrybutach (id produktu, kod produktu, id

marki)

- Tworzenie nowego produktu wraz z jego wariacjami / zestawami atrybutów
- Aktualizacja produktu wraz z jego wariacjami / zestawami atrybutów
- Aktualizacja liczby wariacji produktów w magazynie
- Tworzenie i modyfikacja marek.

8. Przygotowanie dokumentacji OpenAPI dla wystawionych usług HTTP wraz z wykorzystaniem Quarkus Swagger UI.

2.1.1. Uruchomienie lokalnie

Część backendową można uruchomić na lokalnym środowisku. Wymagane jest posiadanie na maszynie Docker wraz z docker compose oraz Java 17.

W celu ułatwienia uruchomienia, przygotowano plik Makefile z przygotowanymi "celami":

1. `local-infra-up` - uruchamia infrastrukturę potrzebną do działania aplikacji (EventStore DB, PostgreSQL, MinIO, Redpanda) z wykorzystaniem docker compose.
2. `run-local` - uruchamia aplikację Quarkus na lokalnej maszynie.

```
$: make local-infra-up
docker compose -f local/docker-compose.yml up -d
[+] Building 0.0s (0/0)
[+] Running 9/9
 ✔ Container buy-it-eventstoredb-certs  Started  0.4s
 ✔ Container buy-it-postgres-adminer    Running  0.0s
 ✔ Container buy-it-postgres            Running  0.0s
 ✔ Container buy-it-minio               Running  0.0s
 ✔ Container buy-it-redpanda            Running  0.0s
 ✔ Container buy-it.eventstore.node3    Running  0.0s
 ✔ Container buy-it.eventstore.node2    Running  0.0s
 ✔ Container buy-it-redpanda-console    Running  0.0s
 ✔ Container buy-it.eventstore.node1    Running  0.0s

$: make run-local
./mvnw compile quarkus:dev -pl application -Dquarkus.profile=local --also-make
# ...
# logi z uruchomienia aplikacji Quarkus
# ...
2023-07-01 15:08:21,097 INFO  [io.quarkus] (Quarkus Main Thread) application 0.0.0-
227e5a29 on JVM (powered by Quarkus 3.1.0.Final) started in 7.773s. Listening on:
http://localhost:4000
2023-07-01 15:08:21,098 INFO  [io.quarkus] (Quarkus Main Thread) Profile local
activated. Live Coding activated.
2023-07-01 15:08:21,099 INFO  [io.quarkus] (Quarkus Main Thread) Installed features:
[agroal, cdi, config-yaml, hibernate-validator, jdbc-postgresql, kafka-client,
liquibase, narayana-jta, reactive-pg-client, resteasy-reactive, resteasy-reactive-
jackson, smallrye-context-propagation, smallrye-graphql, smallrye-openapi, smallrye-
```

```
reactive-messaging, smallrye-reactive-messaging-kafka, swagger-ui, vertx]

--
Tests paused
Press [r] to resume testing, [o] Toggle test output, [:] for the terminal, [h] for
more options>
```

Po uruchomieniu aplikacji w ten sposób, dostępny jest panel developerski Quarkus pod adresem http://localhost:4000/q/dev-ui/extensions.