[PIK/21I] Dokumentacja końcowa

Iwo Sokal, Szymon Kisiel, Olgierd Sobieraj, Illia Yatskevich

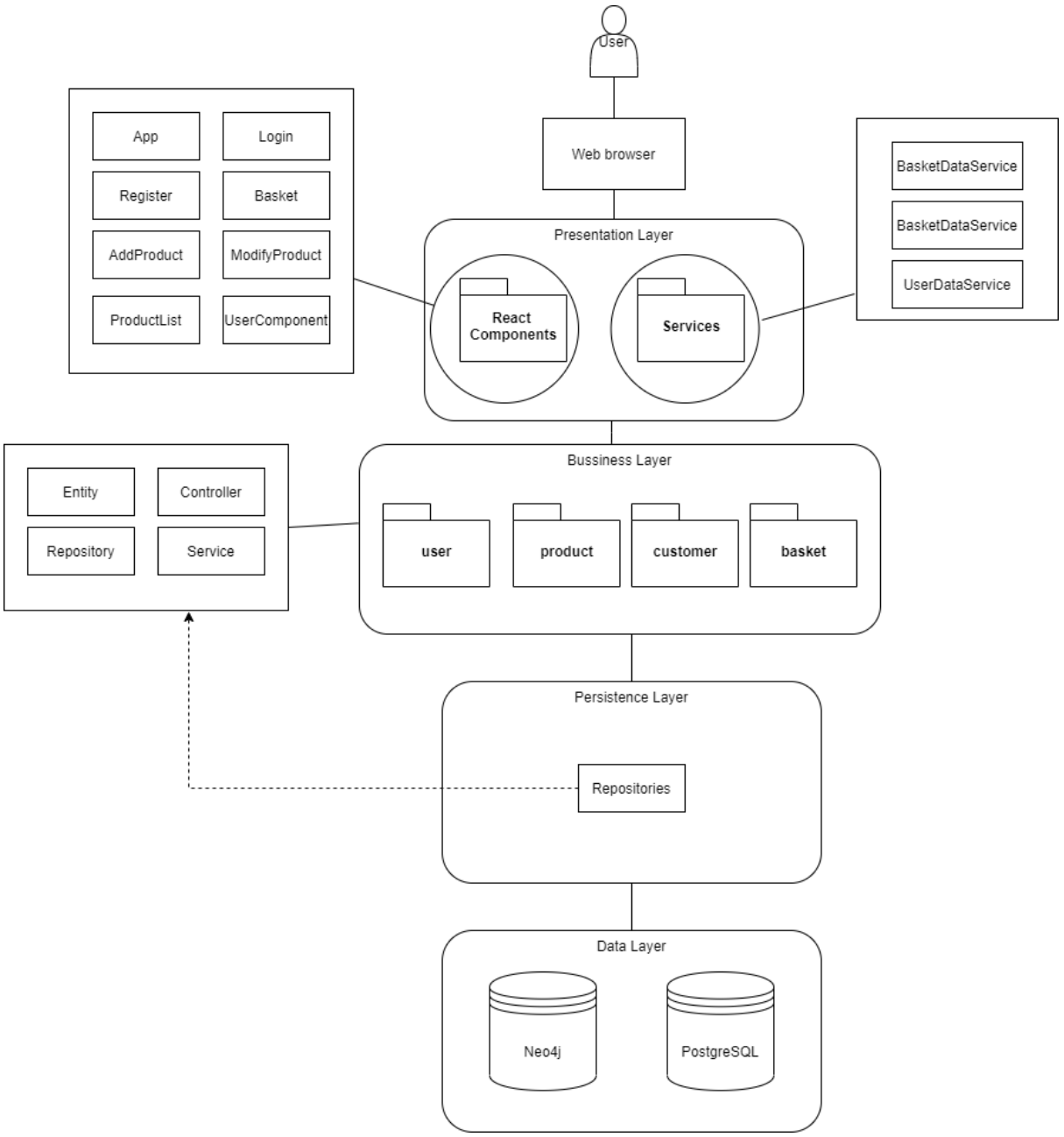
Wstęp

Celem projektu jest stworzenie aplikacji webowej obsługującej prosty sklep internetowy. W sklepie można dodawać, usuwać i modyfikować produkty w sklepie, logować się i rejestrować, obsługiwać koszyk i dokonywać zakupu, wyszukiwać produkty po kategoriach oraz otrzymywać rekomendowane produkty.

Narzędzia

Repozytorium kodu Github
Zarządzanie zadaniami Github Issues
Mierzenie czasu zadań Clockify
Budowanie projektu Maven
Serwer CI Jenkins
Serwer aplikacji Tomcat
Repozytorium artefaktów Nexus
Pokrycie kodu JaCoCo
Testy JUnit5
Frontend React
Backend Spring Boot
Bazy danych Neo4j i PostgreSQL
Komunikacja z użytkownikiem REST API

Diagram aplikacji



Warstwy:

- Presentation Layer - w React, JavaScript, Html i CSS. Zawiera ekrany aplikacji oraz kontrolery odpowiadające za poruszanie się po GUI i sprawdzanie wykonanych akcji. Użytkownik komunikuję się z tą warstwą przez przeglądarkę.
- Business Layer - zawiera:
    - model danych, encje Produkt i Użytkownik
    - kontrolery odpowiadające m.in za podawanie danych za pomocą Rest API, które następnie są pobierane przez frontend
    - funkcjonalności:
        - obsługa koszyka
        - obsługa konta

        - zarządzanie produktami
        - zarządzanie użytkownikami
        - moduł rekomendacji
- Persistence Layer - zawiera repozytoria, to znaczy obiekty porozumiewające się z bazami danych będącymi również DAO dla encji
- Data Layer - bazy danych PostgreSQl i Neo4j.

Implementacja

- Frontend
    - Używamy modułu axios do komunikacji z backendem, używanego w plikach .js z nazwą Service w nazwie
    - Aby React poprawnie działał ze Spring Bootem używamy pluginu frontend-maven-plugin
    - Pozostałe pliki .js odpowiadają za komponenty widoczne w przeglądarce
    - Pliki css są używane do stylowania, określają wygląd stron
    - Do poruszania się po aplikacji używamy biblioteki react-router
- Backend
    - Używamy dwóch baz danych: PostgreSQL oraz Neo4j. PostgreSQL przechowuje dane użytkowników, w tym hasła, obsługuje logowanie i rejestrowanie. Pozostałymi rzeczami zajmuje się baza grafowa Neo4j - przechowuje szczegółowe dane klientów, produkty oraz zakupione lub aktualne koszyki
    - Kontrolery obsługują żądania użytkownika z frontendu, wywołując odpowiednie metody Repository, działających na bazach danych
    - Kontrolery umożliwiają wykonanie metod HTTP: GET, POST, DELETE
    - Repository bazują na klasach encyjnych, User, Product, Basket i Customer
    - Do obsługi PostgreSQL używamy Datasource'ów, a do obsługi Neo4j Spring Data Neo4j
    - Rekomendacje polegają na znalezieniu podobnych koszyków, które mają jak największą liczbą wspólnych produktów z produktami w naszym koszyku (liczba ta jest siłą rekomendacji) i rekomendowaniu produktów, które są w tych podobnych koszykach. Zapytanie w języku Cypher zwracające rekomendacje:

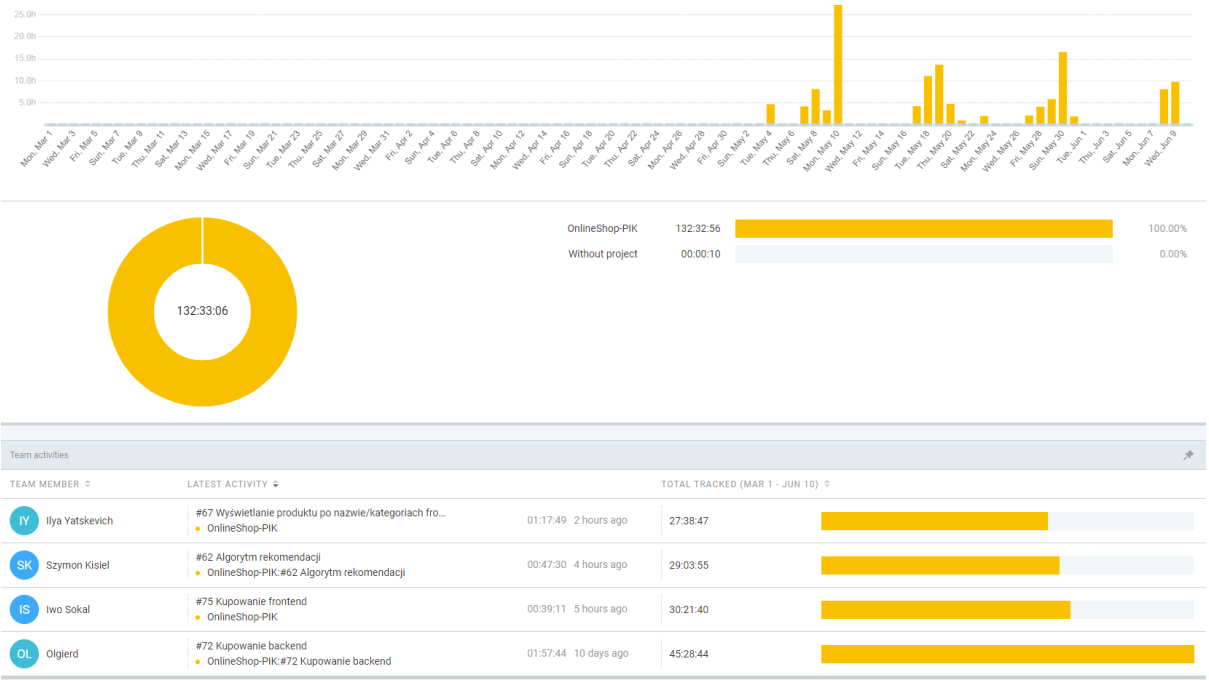
```
MATCH (customer:Customer {login: $customerLogin})
      -[:CURRENT]->(basket:Basket)<--(commonProduct:Product)
      -[r]->(commonBasket:Basket)<-[:BOUGHT]-(:Customer)
MATCH (commonBasket)<--(recommendedProduct:Product)
WHERE NOT (recommendedProduct)-->()<--(customer)
WITH  recommendedProduct,
      commonBasket,
      count(r) AS recommendationStrength
WITH  DISTINCT recommendedProduct,
      MAX(recommendationStrength) AS recommendationStrength
RETURN recommendedProduct, recommendationStrength
ORDER BY recommendationStrength DESC
```

Instrukcja zbudowania projektu

1) mvn clean install w katalogu głównym projektu
2) zdeployowanie pliku online-shop.war w podkatalogu target na tomcata

Raport

Spędzony czas



Iwo Sokal - 54h
Illia Yatskevich - 48h
Olgierd Sobieraj - 66h
Szymon Kisiel - 49h

Zadania

Konfiguracja

Iwo Sokal

- ☑ ~~instalacja Nexusa~~
- ☑ ~~konfiguracja JaCoCo~~
- ☑ ~~konfiguracja Tomcata~~
- ☑ ~~konfiguracja aplikacji na warze~~
- ☑ ~~połączenie Reacta ze Spring Bootem~~

Szymon Kisiel

- ☑ ~~wypełnienie bazy Neo4j~~
- ☑ ~~łączenie z bazą danych Neo4j~~

- [x] ~~koszyk w bazie~~

Olgierd Sobieraj
- [x] ~~wypełnienie bazy PostgreSQL~~
- [x] ~~łączenie z bazą PostgreSQL~~

Wspólnie
- [x] ~~instalacja i konfiguracja Neo4j~~
- [x] ~~instalacja i konfiguracja PostgreSQL~~
- [x] ~~konfiguracja PostgreSQL z Tomcatem (Iwo Sokal, Olgierd Sobieraj)~~
- [x] ~~instalacja Jenkinsa (Szymon Kisiel, Olgierd Sobieraj)~~
- [x] ~~konfiguracja Jenkinsa~~
- [x] ~~konfiguracja Nexusa~~
- [x] ~~xml mavenowy~~

Frontend

Iwo Sokal
- [x] ~~szata graficzna aplikacji~~
- [x] ~~logowanie~~
- [x] ~~rejestrowanie~~
- [x] ~~dodawanie produktów~~
- [x] ~~wyświetlanie produktów~~
- [x] ~~kupowanie~~
- [ ] panel administracyjny (wygląd okna)
- [ ] wyświetlanie historii zakupów
- [ ] wyrejestrowanie użytkownika
- [ ] modyfikowanie swoich danych
- [ ] wyświetlanie swoich danych
- [ ] panel użytkownika

Illia Yatskevich
- [x] ~~wyświetlanie produktów~~
- [x] ~~usuwanie produktów~~
- [x] ~~modyfikacja produktów~~
- [x] ~~dodanie produktów do koszyka~~
- [x] ~~usuwanie produktów z koszyka~~
- [x] ~~filtrowanie produktów po nazwie/kategoriach~~

Backend

Szymon Kisiel
- [x] ~~tworzenie klientów w bazie Neo4j~~
- [x] ~~dodanie produktu do sklepu~~
- [x] ~~usuwanie produktu ze sklepu~~
- [x] ~~dodanie produktu do koszyka~~
- [x] ~~usunięcie produktu z koszyka~~

- [x] ~~kupowanie~~
- [x] ~~rekomendacje~~
- [x] ~~mapowanie encji z Neo4j na klasy~~
- [x] ~~wyświetlanie produktów po kategorii~~
- [ ] wyświetlanie produktów po nazwie
- [ ] modyfikacja produktów

Olgierd Sobieraj

- [x] ~~mapowanie encji z PostGreSQL na klasy~~
- [x] ~~hashowanie haseł~~
- [x] ~~rejestrowanie użytkownika~~
- [x] ~~wyświetlanie koszyka~~
- [x] ~~kupowanie~~
- [x] ~~zapisywanie danych użytkownika w sesji~~
- [ ] wyświetlanie własnych danych
- [ ] wyświetlanie historii zakupów
- [ ] podział na role klient i admin

Wspólnie

- [x] ~~modyfikowanie kontrolera produktów~~
- [x] ~~modyfikacje kontrolera użytkownika~~
- [x] ~~zapisywanie danych użytkownika w sesji (Szymon Kisiel, Olgierd Sobieraj)~~
- [x] ~~kupowanie backend (Szymon Kisiel, Olgierd Sobieraj)~~

Testy

Uważamy, że efektywniej było testować manualnie, ponieważ większość z funkcjonalności opierała się na operacjach na bazach danych, do których dostęp był jedynie z sieci wewnętrznej. Dodatkowo, dane w bazach ciągle się zmieniały, a operacje na nich wykonywane były na tyle proste, że uznaliśmy że napisanie testów w naszym konkretnym przypadku było trudniejsze i bardziej błędogenne niż sama implementacja.

Statystyki

	Count	Size	Lines	Lines of code
.css	8	4kB	252	219
.java	19	25kB	808	587
.js	15	22kB	639	545
Total	42	51kB	1699	1351

Harmonogram

Total: 131:15:07