Mini Projekt - Bazy Danych II

Aplikacja Aukcyjna - Backend i Frontend

Autorzy:

- **Bartosz Ludwin**
- **Filip Malejki**
- **Mateusz Pawliczek**

Link do repozytorium GitHub: https://github.com/FalcO0O/Bazy-danych-II (https://github.com/FalcO0O/Bazy-danych-II)

Wprowadzenie

Projekt stanowi implementację **pełnej aplikacji aukcyjnej** składającej się z backendu oraz frontendu. Umożliwia on tworzenie aukcji oraz licytowanie ich przez użytkowników. Informacje są przechowywane w systemie bazodanowym, w którym znajdują się dane o użytkownikach, aukcjach oraz logi transakcji.

Technologie

Projekt został zrealizowany przy użyciu następujących technologii:

Backend

- **Python + FastAPI**
- **Baza danych**: MongoDB
- **Komunikacja z bazą**: Motor (asynchroniczny klient MongoDB) + PyMongo

Frontend

- **Interfejs użytkownika** dostępny pod adresem **localhost:3000**

Środowisko uruchomieniowe

- **Docker** z Docker Compose

Dodatkowo wykorzystano biblioteki do:

- obsługi zapytań HTTP,
- szyfrowania haseł,

- tworzenia i walidacji tokenów JWT.

Uruchomienie aplikacji

Aplikacja składa się z backendu i frontendu, które są uruchamiane jednocześnie za pomocą Docker Compose:

```
docker compose up
```

Po uruchomieniu:

- **Frontend** będzie dostępny pod adresem: **http://localhost:3000**
- **Backend API** będzie dostępny zgodnie z konfiguracją Docker Compose

---

Funkcjonalności

API aplikacji aukcyjnej umożliwia:

- Rejestrację użytkownika i logowanie do systemu.
- Obsługę tokenów JWT i mechanizmu odświeżania tokenów (refresh token).
- Tworzenie nowych aukcji lub ich zamknięcie przez zalogowanych użytkowników.
- Edycję i usuwanie aukcji przez administratora.
- Licytowanie aukcji przez użytkowników z weryfikacją poprawności oferty.
- Ręczne zakończenie aukcji przez administratora.
- Generowanie raportów zawierających:
    - zestawienie aktywnych i zakończonych aukcji,
    - statystyki użytkowników,
    - informacje o przepływach finansowych.
- Logowanie działań użytkowników (logowanie, rejestracja konta)
- Logowanie zmian aukcji (utworzenie, zamknięcie, licytowanie)

Backend zapewnia poprawne wykonanie powyższych operacji poprzez wykorzystanie **systemu transakcji**. Jest to szczególnie istotne w przypadku konkurencyjnych operacji, takich jak licytacje, gdzie wiele ofert może być składanych jednocześnie.

Frontend zapewnia intuicyjny interfejs użytkownika umożliwiający korzystanie ze wszystkich funkcjonalności aplikacji aukcyjnej bez konieczności bezpośredniego korzystania z API.

Możliwe udoskonalenia

Aktualnie aukcje są kończone ręcznie przez właściciela aukcji lub administratora. Możliwym usprawnieniem byłoby wprowadzenie mechanizmu automatycznego zamykania aukcji na podstawie ustalonej daty zakończenia.

Dokumentacja

W ramach projektu przygotowano osobny plik `dokumentacja.pdf`, który zawiera szczegółowy opis każdego endpointu umożliwiającego komunikację z serwerem i modyfikację danych w bazie.

Dokumentacja zawiera:

- Opis działania każdego endpointu.
- Schemat danych wejściowych i wyjściowych.
- Informację o typie odpowiedzi HTTP *(kody statusu)*.
- Przykładowe zapytania i odpowiedzi w formacie JSON.
- Informację o wymaganiach walidacyjnych *(np. minimalna długość hasła, wymagane pola)*.

Testowanie Race Condition

W celu weryfikacji działania systemu transakcji użytego w endpointach licytacji, przygotowano dedykowany program testujący symulujący warunki wyścigu (race condition).

Testy są uruchamiane na osobnej, testowej bazie danych, co zapewnia bezpieczeństwo i izolację środowiska.

Wymagania

Do uruchomienia testu potrzebny jest interpreter **Python**. Można go pobrać ze strony: https://www.python.org/downloads/ (https://www.python.org/downloads/)

Opis testu

Test obejmuje następujące kroki:

- Rejestrację lub logowanie trzech kont testowych.
- Utworzenie aukcji przez użytkownika A.
- Trzykrotne synchroniczne licytowanie tej aukcji przez użytkowników B i C (równocześnie).
- Zamknięcie aukcji przez użytkownika A.

Uruchomienie testu

Aby uruchomić test:

1. Przejdź do folderu `testing`:

```
cd testing
```

2. Pobierz wymagane dependencje jeśli jeszcze tego nie zrobiłeś:

```
pip install httpx asyncio
```

3. Uruchom skrypt:

```
py racetest.py
```

Test zostanie wykonany, a wyniki pojawią się w konsoli.

Przykładowe wyniki:

```
[!] test-user-xkawofngyh131 already exists, trying to login...
[!] test-user-jjkoelfmah125 already exists, trying to login...
[!] test-user-kdrihnekog232 already exists, trying to login...
 Auction created: 6845fca72bda3f4afc7ac07c
[BID] Status: 200, Amount: 150, Response: {"id":"6845fca72bda3f4afc7ac07e", ...
[BID] Status: 400, Amount: 150, Response: {"detail":"Kwota oferty musi być wyższa ...
[BID] Status: 200, Amount: 200, Response: {"id":"6845fca92bda3f4afc7ac080", ...
[BID] Status: 400, Amount: 200, Response: {"detail":"Kwota oferty musi być wyższa ...
[BID] Status: 200, Amount: 250, Response: {"id":"6845fca92bda3f4afc7ac082", ...
[BID] Status: 400, Amount: 250, Response: {"detail":"Kwota oferty musi być wyższa ...
 Auction 6845fca72bda3f4afc7ac07c closed.
```

Jak widać, system transakcji zapobiega konfliktom — tylko jedna oferta o danej kwocie jest zaakceptowana, a pozostałe są odrzucane. W przypadku błędu transakcyjnego mogłoby dojść do zaakceptowania tej samej kwoty wielokrotnie.