ZADANIE:
KANTOR

OPIS ZADANIA:

W ramach tego zadania należy stworzyć aplikację symulującą e-kantor. W kantorze tym zarejestrowani użytkownicy mogą kupować lub sprzedawać waluty po aktualnych cenach.

WYMAGANIA BIZNESOWE:

- Użytkownicy:
  - Aplikacja umożliwia zarejestrowanie się nowego użytkownika oraz zdefiniowanie przez niego swojego portfela (zasoby finansowe w różnych walutach). Nowy użytkownik nie musi posiadać wszystkich oferowanych przez kantor walut.
  - Do aplikacji jednocześnie może być zalogowanych wielu użytkowników.
  - Każdy użytkownik może na bieżąco sprzedawać waluty ze swojego portfela oraz kupować waluty z kantoru. Nie ma możliwości bezpośredniego handlu pomiędzy różnymi zalogowanymi w tym samym czasie użytkownikami.
  - Przed zakupem/sprzedażą użytkownik powinien potwierdzić daną czynność.
- Kantor:
  - Kantor nie może sprzedać więcej jednostek danej waluty niż tyle, ile obecnie posiada. Każda waluta może być sprzedawana tylko jako wielokrotność bazowej jednostki (np. x * 1 USD, y * 100 CZK).
  - Aplikacja informuje o dokładnym czasie, z którego pochodzą aktualne kursy walut.
- Dodatkowe: (w przypadku, gdy podstawowe funkcjonalności zostaną zaimplementowane)
  - Kantor na żądanie użytkownika generuje wykres średnich kursów walut z ostatnich 20 kursów.
  - Użytkownik może edytować swoje dane oraz portfel.

ZADANIE:
KANTOR

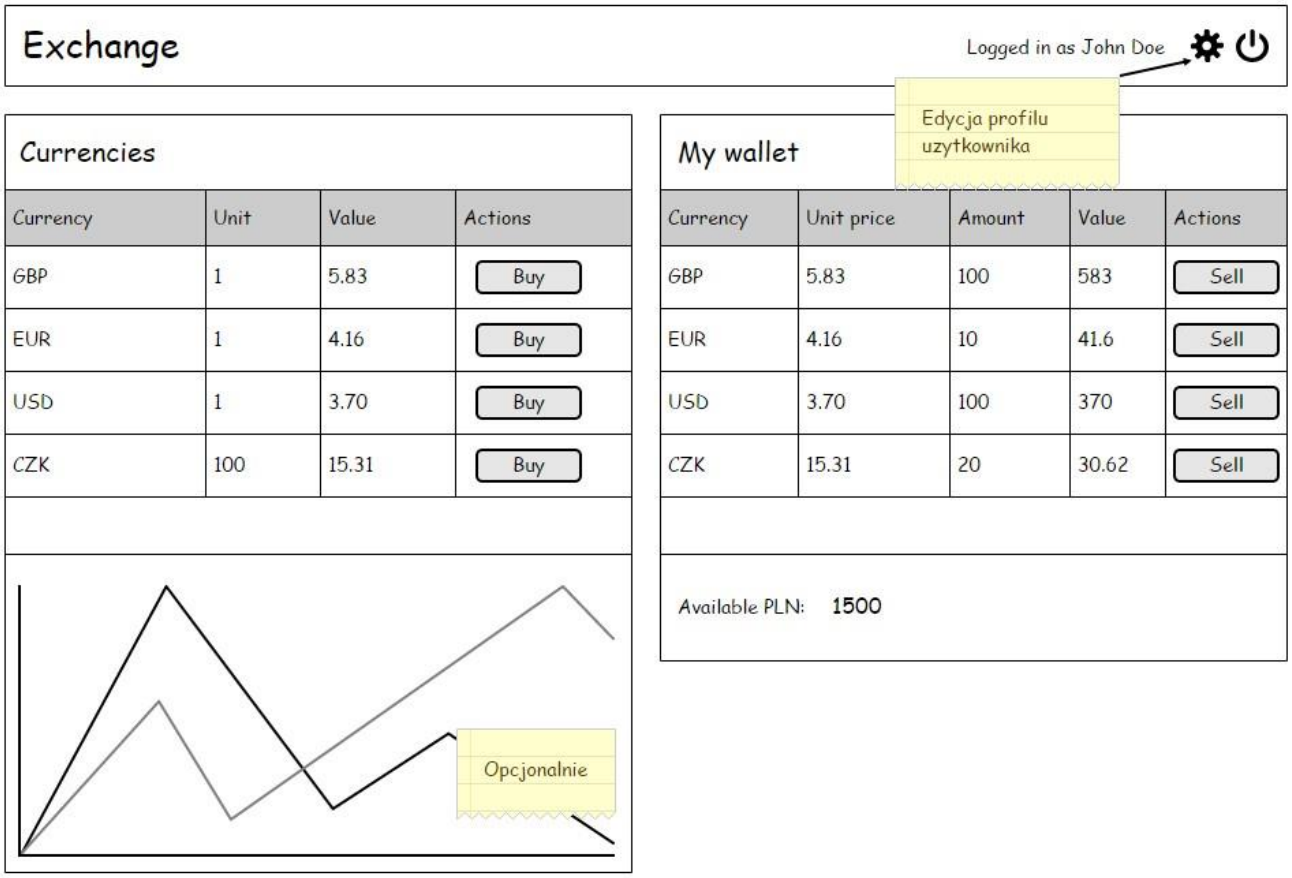
WYMAGANIA TECHNICZNE:

- Kursy walut należy pobierać z udostępnionego przez Future Processing serwera. Udostępnia on dane za przy użyciu protokołu HTTP pod adresem **http://webtask.future-processing.com:8068/currencies** oraz poprzez protokół WebSockets pod adresem **ws://webtask.future-processing.com:8068/ws/currencies**
- Domyślnym formatem zwracanych danych jest JSON. Aby uzyskać dane w innej postaci należy:
  - W przypadku użycia WebSocketów dodać do URL parametr *format* (*format=xml* lub *format=json*).
  - W przypadku połączeń za pomocą protokołu HTTP należy dodać do żądania nagłówek Accept (o wartości application/xml lub application/json).
- Kursy walut mogą zmieniać się co 20-30 sekund. Kantor zawsze musi wyświetlać aktualne kursy walut. W przypadku awarii serwera z kursami lub braku możliwości nawiązania połączenia nie powinno być możliwości dokonania operacji finansowych.
- Rozwiązanie powinno wykorzystywać technologie oparte o AJAX i/lub WebSockets.
- Dla uproszczenia zarówno kantor jak i użytkownicy operują tymi samymi walutami: dolar amerykański (USD), euro (EUR), frank szwajcarski (CHF), rubel (RUB), korona czeska (CZK), funt brytyjski (GBP). Walutą rozliczeniową jest polski złoty (PLN) i w tej walucie należy podawać kursy w kantorze.
- Początkowa wartość portfela kantoru jest definiowana w bazie danych podczas tworzenia aplikacji.
- Aplikacja powinna być zabezpieczona przed atakami z zewnątrz (np. SQL Injection, XSS itp.).
- W ramach zadania powinna zostać stworzona baza danych.
- Całość rozwiązania (tzn. zarówno kod jak i interfejs użytkownika) powinna być w języku angielskim.
- Powinno to być rozwiązanie backendowe oparte o języki uruchamiane na JVM.
- W kwestii uruchamiania aplikacji należy spełnić jeden z trzech warunków:
  - Wszystkie niezbędne do uruchomienia aplikacji pliki konfiguracyjne znajdują się w jej kodzie. Jest załączona dokumentacja opisująca manualne kroki wymagane do jej uruchomienia w środowisku developerskim.
  - Uruchamianie i konfigurowanie aplikacji jest zautomatyzowane za pomocą odpowiednich narzędzi, na przykład Vagrant lub Docker. Procedura uruchomienia aplikacji jest udokumentowana.
  - Aplikacja jest zdeployowana w chmurze publicznej (na przykład AWS, Azure, Heroku, Openshift) lub na dowolnym hostingu pod podanym adresem URL.

ZADANIE:
KANTOR

INTERFEJS UŻYTKOWNIKA:

Interfejs użytkownika powinien zostać utworzony zgodnie z poniższym mock-upem (ew. Inne strony, okna dialogowe, itp. Należy zaprojektować samodzielnie):



Program powinien być najwyższej jakości i charakteryzować się zamkniętą funkcjonalnością, czyli wszystkie zaimplementowane funkcjonalności muszą działać od początku do końca.

Program powinien demonstrować rozsądne użycie możliwości języka w zakresie abstrakcji, interfejsów, dziedziczenia, dobrą strukturę klas oraz pokazywać dobre praktyki programistyczne. **Zwracamy uwagę na jakość kodu i projektu.** Nieprzekraczalny czas na wykonanie zadania: **4 dni.**