Dokumentacja projektu MiniBanku

Mateusz Górczany

2021

1 Opis problemu i opis funkcjonalności udostępnianej przez API

Celem projektu było wykonanie prostego systemu bankowego. Zadanie zakładało wykonanie rozproszonego systemu, składającego się z kilku baz danych i obsługującego je serwera.

Udostępnione API umożliwia dodawanie/pobieranie kont bankowych oraz klientów, a także wykonywanie przelewów pomiędzy kontami. Konta klientów są w jednej - głównej bazie danych - Main, natomiast dane klientów i ich transakcje znajdują się w bazach oddziałów: Rzeszów, Kraków. Utworzony server obsługuje wymianę danych pomiędzy nimi oraz udostępnia API.

Aby uruchomić projekt należy:
{ wymagane technologie: Docker, docker-compose }

- wykonać komendę "docker-compose up"w folderze projektu
- odczekać do pobrania obrazów i uruchomienia projektu - powinna pojawić się informacja o zainicjalizowaniu baz danych, jeśli się nie powiedzie, spróbować jeszcze raz, jak nie zadziała to należy zmienić czas oczekiwania w skrypcie DB/init_db.sh na dłuższy
- wejść przez przeglądarkę na adres: http://localhost:8080/swagger/index.html, gdzie można przetestować API

Aby zatrzymać projekt należy:

- nacisnąć Ćtrl+C"w terminalu, gdzie uruchomiony został system
- wykonać komendę "docker-compose down"

2 Opis typów danych oraz metod (funkcji) udostępnionych w ramach API

2.1 Typy danych

Departments

- **id**,
- **name:** nazwa placówki,

- Typ przechowujący departamenty, które reprezentowane są przez oddzielne bazy danych.

Accounts

- **id**,
- **department_id:** identyfikator departamentu,
- **customer_id:** identyfikator właściciela konta,
- **balance:** ilość środków na koncie,

- Reprezentuje konto użytkownika w bazie danych

Customers

- **id**,
- **fname:** imię,
- **lname:** nazwisko,
- **pesel:** pesel,
- **dateOfBirth:** data urodzenia,
- **phone:** telefon,
- **email:** adres email

- Reprezentuje klienta banku

Transactions

- **id**,

- **fromAccount:** identyfikator konta zlecającego przelew,
- **fromDepartment:** nazwa departamentu,
- **toAccount:** identyfikator konta otrzymującego przelew,
- **toDepartment:** nazwa departamentu,
- **value:** wartość przelewu,
- **dateOfTransaction:** data transakcji

- Reprezentuje przelew pomiędzy dwoma kontami.

2.2 Metody API

- **POST** /api/accounts/transfer
  obsługa transferu danych pomiędzy kontami (użycie kilku baz)
- **GET** /api/accounts/
  pobieranie kont wszystkich klientów
- **POST** /api/accounts/
  dodawanie konta nowego użytkownika
- **GET** /api/accounts/detailed
  pobieranie wszystkich kont wraz z danymi właścicieli (użycie kilku baz)
- **GET** /api/departments
  pobieranie wszystkich departamentów
- **GET** /api/rzeszow/customers
  pobieranie wszystkich rzeszowskich klientów
- **POST** /api/rzeszow/customers
  dodawanie klienta do rzeszowskiej bazy
- **GET** /api/rzeszow/transactions
  transakcje rzeszowskich klientów
- **GET** /api/krakow/customers
  pobieranie wszystkich krakowskich klientów
- **POST** /api/krakow/customers
  dodawanie klienta do krakowskiej bazy
- **GET** /api/krakow/transaction
  transakcje krakowskich klientów

3 Opis implementacji

Do wykonania zadania użyto następujących technologii:

- **C#**
  - .NET Core 5.0
  - Entity Framework
  - LINQ

  technologi użyto do napisania servera implementującego wymagane funkcjonalności,
  skorzystano także z mapowania obiektowego oraz technologii LINQ
- **SQL Server Express**
  utworzono trzy instancje, które reprezentowały główną bazę danych i dwie od niej zależne
- **Docker**
  użyto go do skonteneryzowania baz danych oraz servera
- **docker-compose**
  użyto do zorganizowania kontenerów w jeden spójny system

  Po uruchomieniu projektu bazy danych zostają zainicjalizowane przy użyciu skryptów języka Bash, które czekają 60 sekund na wykonanie zadania.

4 Podsumowanie, wnioski

Utworzony system z powodzeniem implementuje założone funkcjonalności. Użytkownik może pobierać, dodawać dane oraz wykonywać przelewy.

Obsługa kilku baz danych przez jeden server oraz wykonywanie transferu danych pomiędzy nimi działa poprawnie, co można przetestować na stronie dokumentacji swaggera.

Konteneryzacja pozwoliła na łatwe uruchomienie projektu w innym środowisku komputerowym bez obaw o niezgodność zależności, czy inne błędy nieznanego pochodzenia.

Niestety jedyną techniczną funkcjonalnością, która nie została użyta, choć powinna, są "distributed transactions". .NET Core w każdej wersji jej nie wspiera - można to sprawdzić odkomentowując kod transakcji w metodzie "TransferMoney"w klasie kontrolera w pliku Ćontroller.cs". Obsługa "Distributed Transactions"jest dostępna tylko w środowisku .NET na systemy Windows. Brak użycia

tej funkcji nie powoduje jednak błędu systemu - funkcjonalnośc przelewów w dalszym stopniu działa poprawnie - przy błędzie jednej z operacji zmiany w innych nie są aplikowane.

5 Literatura

- https://docs.microsoft.com/en-us/dotnet/
- https://www.youtube.com/channel/UCIMRGVXufHT69s1uaHHYJIA
- https://upel2.cel.agh.edu.pl/wfiis/course/view.php?id=412