Kolegium Nauk Przyrodniczych

Uniwersytet Rzeszowski

Przedmiot:

Bazy danych

System Bankowy

Wykonali:

Jakub Siłka 131509

Paweł Powęska 131496

Prowadzący: Piotr Grochowalski

Rzeszów 2025

Spis Treści

Wstęp

Baza danych banku została zaprojektowana, aby zarządzać operacjami finansowymi, danymi klientów oraz transakcjami w sposób efektywny, bezpieczny i zorganizowany. Głównym celem tego systemu jest zarządzanie środkami finansowymi klientów, obsłudze płatności oraz realizacji operacji bankowych takich jak przelewy, obsługa kart płatniczych czy zarządzanie odbiorcami. System ten pełni kluczową rolę w cyfrowym funkcjonowaniu banku, oferując kompleksowe wsparcie w zakresie bankowości elektronicznej.

Cele i zadania bazy danych

- Gromadzenie i przechowywanie wszystkich kluczowych danych dotyczących klientów, ich kont, kart oraz transakcji w jednym miejscu.
- Zarządzanie danymi klientów, takimi jak PESEL, imię, nazwisko czy numer telefonu, co umożliwia bankowi identyfikację i obsługę użytkowników.
- Powiązanie klienta z jego kontami.
- Łatwe zakładania nowych kont, monitorowania salda, statusu oraz zarządzania danymi logowania i hasłami.
- Umożliwienie integracji kont z wieloma kartami płatniczymi i odbiorcami przelewów.
- Monitorowanie statusu karty (aktywna/nieaktywna) oraz zarządzanie limitami dziennymi i pojedynczymi.
- Zapewnienie rejestracji operacji finansowych takich jak przelewy.
- Umożliwienie kontroli przepływu środków między kontami oraz rejestracji kluczowych danych transakcji (data, kwota, typ).

- Przechowywanie danych o odbiorcach powiązanych z kontami użytkowników, co pozwala na szybkie i wygodne wykonywanie przelewów. Umożliwienie dodawania opisów odbiorców w celu łatwiejszego rozpoznawania zapisanych kontaktów.

Baza danych banku jest kluczowym elementem systemu, który pozwala na efektywne zarządzanie operacjami finansowymi, zapewniając jednocześnie bezpieczeństwo i wygodę użytkowników. Dzięki swojej strukturze rozwiązuje problemy związane z chaosem danych, bezpieczeństwem transakcji oraz wydajnością obsługi klientów, wspierając zarówno codzienne działania banku, jak i jego długoterminowe cele strategiczne.

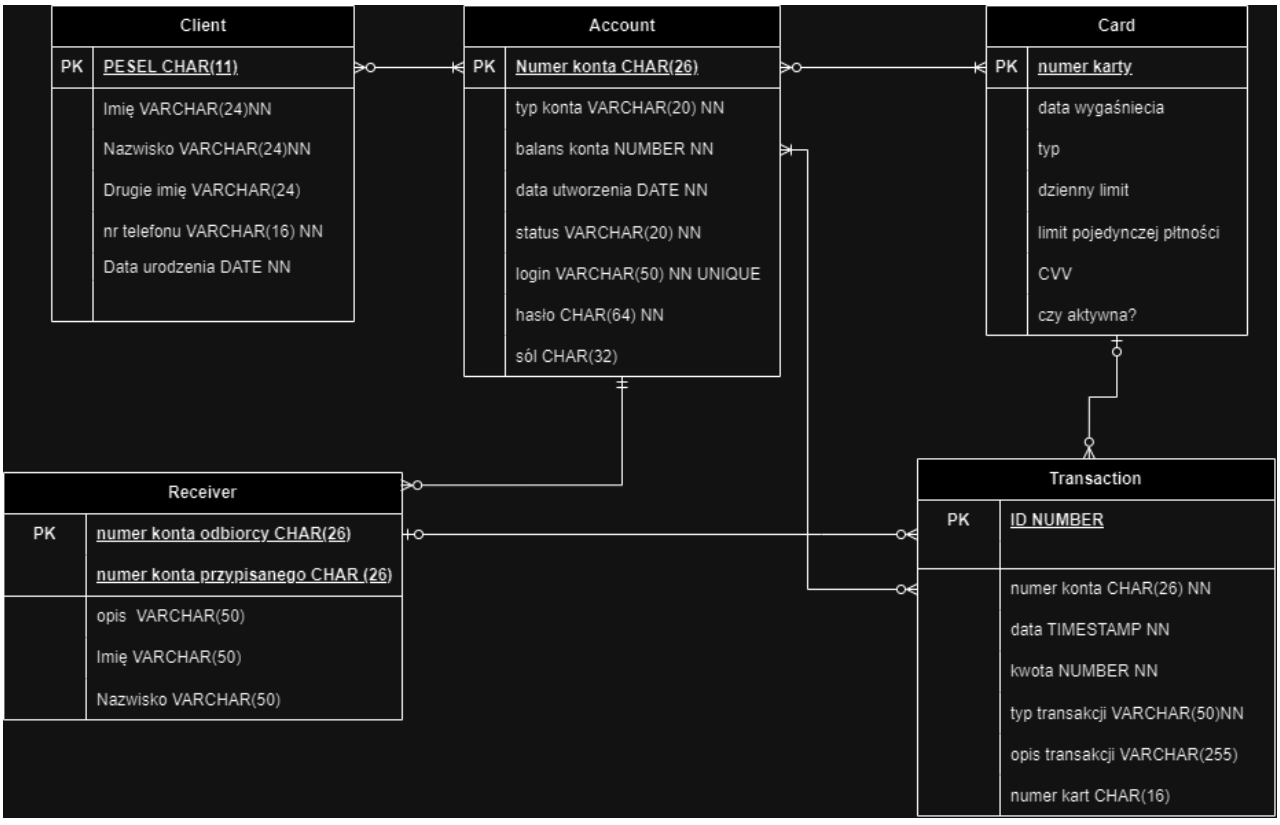
Baza danych

W projekcie zastosowano relacyjną bazę danych, zaprojektowaną z wykorzystaniem języków SQL oraz PL/SQL w środowisku Oracle SQL Developer Free. Baza danych pełni kluczową rolę w systemie bankowym, zapewniając niezawodne przechowywanie i efektywne zarządzanie informacjami, które są podstawą funkcjonowania systemu.

Architektura i Schemat bazy danych

System bazuje na relacyjnej strukturze danych, w której informacje są przechowywane w tabelach, a zależności między nimi definiowane za pomocą kluczy obcych i głównych. Projekt bazy danych został stworzony w środowisku Oracle z wykorzystaniem PL/SQL, co umożliwia implementację zaawansowanych procedur i funkcji. Każdy rodzaj przechowywanych danych posiada dedykowaną tabelę, np. dane osobowe w tabeli client, a informacje o kontach w tabeli account.

Diagram schematu więzów encji:



Opis Struktury bazy danych

Podstawowym elementem bazy danych są tabele:

- **Tabela CLIENT** - Tabela przechowuje dane osobowe klientów banku. Zawiera następujące kolumny:
  - PESEL (CHAR(11)): unikalny numer identyfikacyjny klienta, pełni rolę klucza głównego.
  - first_name, last_name, middle_name (VARCHAR(24)): imię, nazwisko i opcjonalne drugie imię klienta.
  - phone_number (VARCHAR(16)): numer telefonu klienta.
  - date_of_birth (DATE): data urodzenia klienta.
- **Tabela CLIENT_ACCOUNT** - łączy klientów z ich kontami bankowymi. Klucz główny składa się z dwóch kolumn:
  - pesel (CHAR(11)): klucz obcy wskazujący na PESEL klienta.
  - account_number (CHAR(26)): klucz obcy wskazujący na account_number w tabeli ACCOUNT.
- **Tabela ACCOUNT** - przechowuje informacje o kontach bankowych:
  - account_number (CHAR(26)): unikalny numer konta, pełni rolę klucza głównego.
  - type_of_account (VARCHAR(20)): typ konta (np. oszczędnościowe, bieżące).
  - balance (NUMBER): saldo konta.
  - date_of_creation (DATE): data utworzenia konta.
  - status (VARCHAR(20)): status konta (np. aktywne, zablokowane).
  - login (VARCHAR(50)): login do konta (unikalny).
  - salt, password (CHAR): zabezpieczenia konta (hasło oraz salt do szyfrowania).
- **Tabela RECEIVER** - przechowuje informacje o odbiorcach przelewów:
  - account_number_RECEIVER i account_number_TIED (CHAR(26)): numery kont, jedno powiązane z odbiorcą, drugie z kontem klienta.
  - description (VARCHAR(255)): opis odbiorcy.
  - first_name, last_name (VARCHAR(50)): imię i nazwisko odbiorcy.
- **Tabela CARD** - zawiera informacje o kartach płatniczych przypisanych do kont:
  - card_number (CHAR(16)): unikalny numer karty, pełni rolę klucza głównego.
  - date_of_expiration (DATE): data ważności karty.
  - daily_limit, single_payment_limit (NUMBER): limity transakcji.
  - CVV (CHAR(3)): kod CVV karty.
  - is_active (NUMBER): informacja, czy karta jest aktywna.
  - account_number (CHAR(26)): powiązanie karty z kontem bankowym.

- PIN (CHAR(4)): numer PIN karty.

❖ **Tabela TRANSACTION** - zawiera dane o transakcjach wykonywanych na kontach:

- ID (NUMBER): unikalny identyfikator transakcji.
- account_number, account_number_RECEIVER (CHAR(26)): numery kont klienta i odbiorcy.
- date_of_transaction (TIMESTAMP): data i czas transakcji.
- amount (NUMBER): kwota transakcji.
- type_of_transaction (VARCHAR(50)): typ transakcji (np. przelew, wpłata).
- description_of_transaction (VARCHAR(255)): opis transakcji.
- card_number (CHAR(16)): numer karty, jeśli transakcja jest powiązana z kartą.

Referencje i Zasady Usuwania:

Tabela CLIENT:

- Referencje: Tabela CLIENT jest powiązana z tabelą CLIENT_ACCOUNT przez klucz obcy pesel w tabeli CLIENT_ACCOUNT, który odnosi się do kolumny PESEL w tabeli CLIENT. Powiązanie to umożliwia określenie, który klient jest właścicielem konkretnego konta.
- Usuwanie: Usunięcie rekordu z tej tabeli prowadzi do usunięcia relacyjnych rekordów w tabeli CLIENT_ACCOUNT.

Tabela ACCOUNT:

- Referencje: Tabela ACCOUNT jest powiązana z tabelą CLIENT_ACCOUNT poprzez klucz obcy account_number, który odnosi się do kolumny account_number w tabeli ACCOUNT. Tabela CLIENT_ACCOUNT łączy dane konta z klientem, wskazując, który klient posiada dane konto.
- Usuwanie: Usunięcie rekordu z tej tabeli prowadzi do usunięcia relacyjnych rekordów w tabeli CLEINT_ACCOUNT.

Tabela CLIENT_ACCOUNT:

- Referencja: Kolumna pesel jest kluczem obcym odnoszącym się do PESEL w tabeli CLIENT, a kolumna account_number jest kluczem obcym odnoszącym się do account_number w tabeli ACCOUNT.

- Usuwanie: Zasada ON DELETE CASCADE została zastosowana, usunięcie klienta lub konta w tabelach źródłowych sprawi, że rekordy w tabeli łączącej zostaną usunięte. Jeżeli ilość tych relacyjnych rekordów spadnie do 0 (dla danego konta) nastąpi do usunięcia rekordu w tabeli klient.

Tabela RECEIVER:

- Referencja: Kolumna account_number_TIED jest kluczem obcym odnoszącym się do account_number w tabeli ACCOUNT. Oznacza to, że każdemu odbiorcy przypisane jest konto w tabeli ACCOUNT.
- Usuwanie: W przypadku usunięcia rekordu w tabeli ACCOUNT, który jest powiązany z tabelą RECEIVER, nastąpi usunięcie rekordów powiązanych w tej tabeli, dzięki regule ON DELETE CASCADE.

Tabela CARD:

- Referencja: Kolumna account_number jest kluczem obcym odnoszącym się do account_number w tabeli ACCOUNT. Każda karta jest przypisana do konkretnego konta.
- Usuwanie: Reguła ON DELETE CASCADE została zastosowana w przypadku tej tabeli. Oznacza to, że jeśli rekord w tabeli ACCOUNT zostanie usunięty, to wszystkie rekordy w tabeli CARD związane z tym kontem również zostaną automatycznie usunięte. Pomaga to utrzymać spójność danych i usuwa karty przypisane do usuniętych kont.

Tabela TRANSACTION:

- Referencja: Kolumna account_number w tabeli TRANSACTION jest kluczem obcym odnoszącym się do account_number w tabeli ACCOUNT, a kolumna account_number_RECEIVER jest kluczem obcym odnoszącym się do account_number w tabeli ACCOUNT. Ponadto, kolumna card_number w tabeli TRANSACTION jest kluczem obcym odnoszącym się do card_number w tabeli CARD.
- Usuwanie: Tabela TRANSACTION nie zawiera określonej reguł usuwania, co oznacza, że jeśli konto lub karta, powiązane z transakcją zostaną usunięte. To

transakcje nie będą usuwane i pozostaną w bazie. Rekordy te w aktualnym systemie są tutaj zapisywane permanentnie.

Szczegółowy opis metod CRUD dla tabeli CLIENT

Procedury oraz funkcje CRUD dla tabel znajdują się w odpowiadających im pakietom. Na przykład CRUD dla tabeli CLIENT znajduje się w pliku client_pkg.sql.

Operacja CREATE

CREATE_CLIENT

```
PROCEDURE CREATE_CLIENT (
    p_pesel IN CHAR,
    p_first_name IN VARCHAR2,
    p_last_name IN VARCHAR2,
    p_middle_name IN VARCHAR2,
    p_phone_number IN VARCHAR2,
    p_date_of_birth IN DATE,
    p_result OUT VARCHAR2
) IS
    valid_pesel BOOLEAN := FALSE;
    v_pesel CHAR;
BEGIN

    IF LENGTH(v_pesel) = 11 AND REGEXP_LIKE(v_pesel, '^\d{11}$') THEN
        valid_pesel := TRUE;
    ELSE
        p_result := 'Zły pesel';
        RETURN;
    END IF;

    IF valid_pesel THEN
        BEGIN
            INSERT INTO CLIENT (
                PESEL, first_name, last_name, middle_name, phone_number, date_of_birth
            ) VALUES (
                v_pesel, p_first_name, p_last_name, p_middle_name, p_phone_number, p_date_of_birth
            );
            COMMIT;
            p_result := 'Client created successfully.';
        EXCEPTION
            WHEN OTHERS THEN
                ROLLBACK;
                IF SQLCODE = -1400 THEN
                    p_result := 'Error: One or more required fields are missing or NULL.';
                ELSE
                    p_result := 'Error creating client: ' || SQLERRM;
                END IF;
        END;
    END IF;
END CREATE_CLIENT;
```

Procedura ta służy do tworzenia nowego rekordu w tabeli CLIENT. Pobiera dane

wejściowe, takie jak PESEL, imię, nazwisko, drugie imię, numer telefonu oraz datę urodzenia, a następnie sprawdza poprawność numeru PESEL. Walidacja numeru PESEL polega na sprawdzeniu, czy jego długość wynosi 11 znaków i czy zawiera wyłącznie cyfry. W przypadku niepoprawnego numeru procedura zwraca komunikat błędu i przerywa działanie. Po pomyślnym zweryfikowaniu numeru PESEL następuje próba wstawienia danych do tabeli. Procedura obsługuje potencjalne błędy, takie jak naruszenie ograniczeń NOT NULL, dzięki zastosowaniu mechanizmu obsługi wyjątków. Jeśli operacja się powiedzie, dane zostają zatwierdzone COMMITem, a wyjściowy parametr p_result przekaże informację o sukcesie. W przeciwnym razie wykonywany jest ROLLBACK, a p_result przekaże odpowiedni komunikat o błędzie.

Operacje READ

READ_ALL_CLIENTS

```
FUNCTION READ_ALL_CLIENTS RETURN client_tbl PIPELINED IS
BEGIN
    FOR rec IN (
        SELECT PESEL,
               first_name,
               last_name,
               middle_name,
               phone_number,
               date_of_birth
        FROM CLIENT
    ) LOOP
        PIPE ROW(client_obj(
            rec.PESEL,
            rec.first_name,
            rec.last_name,
            rec.middle_name,
            rec.phone_number,
            rec.date_of_birth
        ));
    END LOOP;
    RETURN;
END READ_ALL_CLIENTS;
```

Funkcja ta zwraca wszystkie rekordy z tabeli CLIENT. Iteruje po każdym rekordzie tabeli, a następnie mapuje dane na obiekty typu client_obj. Funkcja pozwala uzyskać pełną listę klientów przechowywanych w bazie danych, niezależnie od ich liczby. Jest potrzebna do wyświetlenia wszystkich rekordów tabeli, co jest bardzo przydatne do jej analizy.

READ_CLIENT_BY_PESEL

```
FUNCTION READ_CLIENT_BY_PESEL (
    p_pesel IN CHAR
) RETURN client_tbl PIPELINED IS
BEGIN
    FOR rec IN (
        SELECT PESEL,
               first_name,
               last_name,
               middle_name,
               phone_number,
               date_of_birth
        FROM CLIENT
        WHERE PESEL = p_pesel
    ) LOOP
        PIPE ROW(client_obj(
            rec.PESEL,
            rec.first_name,
            rec.last_name,
            rec.middle_name,
            rec.phone_number,
            rec.date_of_birth
        ));
    END LOOP;
    RETURN;
END READ_CLIENT_BY_PESEL;
```

Funkcja służy do wyszukiwania danych pojedynczego klienta na podstawie numeru PESEL. Po podaniu tego numeru jako parametru wejściowego funkcja odnajduje odpowiedni rekord w tabeli CLIENT i zwraca go w postaci obiektu. Dzięki tej funkcji możliwe jest szybkie pozyskanie informacji o konkretnym kliencie. W przypadku, gdy numer PESEL nie istnieje w bazie, wynik funkcji będzie pusty.

READ_CLIENT_BY_ACCOUNT

```
FUNCTION READ_CLIENTS_BY_ACCOUNT (
    p_account_number IN CHAR
) RETURN client_tbl PIPELINED IS
BEGIN
    FOR rec IN (
        SELECT c.PESEL,
               c.first_name,
               c.last_name,
               c.middle_name,
               c.phone_number,
               c.date_of_birth
        FROM CLIENT c
        INNER JOIN CLIENT_ACCOUNT ca ON c.PESEL = ca.pesel
        WHERE ca.account_number = p_account_number
    ) LOOP
        PIPE ROW(client_obj(
            rec.PESEL,
            rec.first_name,
            rec.last_name,
            rec.middle_name,
            rec.phone_number,
            rec.date_of_birth
        ));
    END LOOP;
    RETURN;
END READ_CLIENTS_BY_ACCOUNT;
```

Funkcja ta umożliwia pobranie danych klientów powiązanych z określonym numerem konta. Wykorzystuje relację między tabelami CLIENT i CLIENT_ACCOUNT, aby znaleźć klientów przypisanych do danego konta bankowego. Dane klientów są następnie zwracane jako kolekcja obiektów. Funkcja jest przydatna w sytuacjach, gdy konieczne jest zidentyfikowanie wszystkich osób korzystających z jednego konta, na przykład w przypadku kont wspólnych.

Operacje UPDATE

UPDATE_FIRST_NAME

```
PROCEDURE UPDATE_FIRST_NAME (
    p_pesel IN CHAR,
    p_value IN VARCHAR2,
    p_result OUT VARCHAR2
) IS
BEGIN
    UPDATE CLIENT
    SET first_name = p_value
    WHERE PESEL = p_pesel;

    IF SQL%ROWCOUNT > 0 THEN
        COMMIT;
        p_result := 'First name updated successfully.';
    ELSE
        p_result := 'Error: Client not found.';
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        p_result := 'Error updating first name: ' || SQLERRM;
END UPDATE_FIRST_NAME;
```

Procedura ta pozwala na zaktualizowanie imienia klienta w tabeli CLIENT na podstawie numeru PESEL. Po pomyślnym wykonaniu aktualizacji zatwierdzane są zmiany (COMMIT), a parametr p_result informuje o powodzeniu operacji. Jeśli klient o podanym numerze PESEL nie istnieje, procedura zwraca komunikat o błędzie. Mechanizm obsługi wyjątków zapewnia, że wszelkie nieprzewidziane błędy w trakcie aktualizacji skutkują wycofaniem zmian (ROLLBACK).

UPDATE_MIDDLE/LAST_NAME oraz UPDATE_PHONE_NUMBER

```
PROCEDURE UPDATE_MIDDLE_NAME (
    p_pesel IN CHAR,
    p_value IN VARCHAR2,
    p_result OUT VARCHAR2
) IS
BEGIN
    UPDATE CLIENT
    SET middle_name = p_value
    WHERE PESEL = p_pesel;

    IF SQL%ROWCOUNT > 0 THEN
        COMMIT;
        p_result := 'Middle name updated successfully.';
    ELSE
        p_result := 'Error: Client not found.';
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        p_result := 'Error updating middle name: ' || SQLERRM;
END UPDATE_MIDDLE_NAME;

PROCEDURE UPDATE_PHONE_NUMBER (
    p_pesel IN CHAR,
    p_value IN VARCHAR2,
    p_result OUT VARCHAR2
) IS
BEGIN
    UPDATE CLIENT
    SET phone_number = p_value
    WHERE PESEL = p_pesel;

    IF SQL%ROWCOUNT > 0 THEN
        COMMIT;
        p_result := 'Phone number updated successfully.';
    ELSE
        p_result := 'Error: Client not found.';
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        p_result := 'Error updating phone number: ' || SQLERRM;
END UPDATE_PHONE_NUMBER;
```

Podobnie jak w przypadku procedury aktualizującej imię, procedury aktualizujące drugie imię oraz nazwisko umożliwiają zmianę wartości komórki w tabeli CLIENT. Metoda identyfikuje rekord na podstawie peselu i dokonuje aktualizacji wartości w rekordzie. Metoda do aktualizacje numeru telefonu działa w dokładnie taki sam sposób.

Operacja DELETE

DELETE_CLIENT

```
PROCEDURE DELETE_CLIENT (
    p_pesel IN CHAR,
    p_result OUT VARCHAR2
) IS
    v_client_count NUMBER;
    v_account_result VARCHAR2(255);
BEGIN
    FOR account_rec IN (
        SELECT account_number
        FROM CLIENT_ACCOUNT
        WHERE pesel = p_pesel
    ) LOOP
        DELETE FROM CLIENT_ACCOUNT
        WHERE pesel = p_pesel
        AND account_number = account_rec.account_number;

        SELECT COUNT(*)
        INTO v_client_count
        FROM CLIENT_ACCOUNT
        WHERE account_number = account_rec.account_number;

        IF v_client_count = 0 THEN
            account_pkg.DELETE_ACCOUNT(account_rec.account_number, v_account_result);
            DBMS_OUTPUT.PUT_LINE(v_account_result);
        END IF;
    END LOOP;

    DELETE FROM CLIENT WHERE pesel = p_pesel;

    COMMIT;
    p_result := 'Client and associated accounts handled successfully.';

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        p_result := 'Error: ' || SQLERRM;
END DELETE_CLIENT;
```

Procedura usuwa rekord klienta z tabeli CLIENT na podstawie numeru PESEL. Przed usunięciem sprawdza, czy klient ma powiązane rekordy w tabeli CLIENT_ACCOUNT. W przypadku znalezienia takich rekordów są one usuwane w pierwszej kolejności. Jeśli konto klienta nie jest powiązane z żadnym innym klientem, procedura wywołuje funkcję usuwającą konto z tabeli ACCOUNT. Po usunięciu wszystkich powiązanych rekordów oraz klienta następuje zatwierdzenie transakcji COMMITem. W przypadku błędów transakcja

jest wycofywana, a parametr wyjściowy p_result zwraca szczegółowe informacje o problemie.

Poglądowy opis metod CRUD dla pozostałych tabel

Tabela ACCOUNT

Procedura create_account umożliwia utworzenie nowego konta użytkownika. Generuje unikalny numer konta na podstawie losowej liczby i aktualnej daty. Hasło użytkownika jest haszowane przy użyciu algorytmu SHA-256 z dodatkiem losowej soli, co zwiększa bezpieczeństwo. Procedura wywołuje wewnętrzną procedurę create_client_account, która zapisuje powiązanie między klientem (identyfikowanym przez PESEL) a nowo utworzonym kontem. Kryteria, które muszą zostać spełnione: poprawna długość PESEL (11 znaków). W przypadku błędu (np. brak wymaganych danych) transakcja jest wycofywana.

Funkcja read_all_accounts_func zwraca listę wszystkich kont w systemie, przekazując wyniki w formacie tabelarycznym za pomocą mechanizmu pipelined.

Funkcja read_account_by_number_func umożliwia odczyt szczegółów konkretnego konta na podstawie jego numeru.

Funkcja read_accounts_by_client zwraca listę wszystkich kont powiązanych z danym klientem, identyfikowanym na podstawie numeru PESEL.

Wszystkie funkcje READ wykorzystują kursory do iteracji po danych z tabeli ACCOUNT i w przypadku powiązań klienta, również z tabeli CLIENT_ACCOUNT.

Procedura update_password wymaga weryfikacji starego hasła przed zmianą na nowe. W tym celu porównuje hash hasła wprowadzonego przez użytkownika z zapisanym w tabeli ACCOUNT. Jeśli hasła się zgadzają, nowe hasło jest hashowane i zapisane w bazie.

Procedura update_login sprawdza, czy nowy login nie jest już zajęty, zanim zostanie zaktualizowany.

Wersje administracyjne procedur (admin_update_password i admin_update_login) pozwalają na zmianę hasła lub loginu bez konieczności podawania starego hasła lub wykonywania innych weryfikacji, co jest przydatne w wersji administratorskiej aplikacji.

Procedura DELETE_ACCOUNT usuwa konto użytkownika z tabeli ACCOUNT oraz wszystkie powiązane z nim rekordy z tabel CARD i CLIENT_ACCOUNT. Przed usunięciem rekordów procedura zapewnia spójność danych poprzez kaskadowe czyszczenie zależnych rekordów. Jeżeli konto nie istnieje, procedura zwraca odpowiedni komunikat. W przypadku błędów cała transakcja jest wycofywana.

Tabela CLIENT_ACCOUNT

Procedura ADD_CLIENT_TO_ACCOUNT dodaje nowego klienta do istniejącego konta w tabeli CLIENT_ACCOUNT. Sprawdza, czy podane powiązanie między PESEL a numerem konta już istnieje. Jeśli tak, zwraca komunikat, że konto i klient są już powiązane. W przypadku braku istniejącego powiązania dodaje nowy rekord do tabeli, zapisuje zmiany w bazie za pomocą COMMIT i obsługuje błędy, takie jak brak wymaganych danych. Procedura zwraca odpowiedni komunikat w zależności od wyniku operacji.

Procedura DELETE_CLIENT_FROM_ACCOUNT usuwa powiązanie klienta z kontem w tabeli CLIENT_ACCOUNT. Przed usunięciem sprawdza, czy konto jest powiązane z co najmniej dwoma klientami. Jeśli konto ma tylko jednego klienta, operacja jest blokowana, a procedura zwraca komunikat o braku możliwości usunięcia ostatniego powiązania. W przypadku spełnienia warunków rekord jest usuwany z tabeli, zmiany zapisywane za pomocą COMMIT, a w razie wystąpienia błędu transakcja jest wycofywana. Procedura zwraca komunikat informujący o wyniku operacji.

Tabela CARD

Procedura CREATE_CARD tworzy nową kartę płatniczą. Parametry wejściowe obejmują limity dzienne i pojedynczej płatności, CVV, aktywność karty, numer konta oraz PIN. Procedura generuje unikalny numer karty i ustala datę jej wygaśnięcia na pięć lat od daty bieżącej. Weryfikowana jest długość PIN-u (4 cyfry) i CVV (3 cyfry). W przypadku błędów, takich jak brak wymaganych danych lub inne problemy z bazą danych, wynik operacji jest odpowiednio modyfikowany, a transakcja wycofywana.

Funkcja READ_ALL_CARDS_FUNC zwraca wszystkie karty zapisane w tabeli CARD w formie rekordów typu card_type. Dane są zwracane w sposób strumieniowy (PIPELINED), co umożliwia ich przetwarzanie w miarę ich odczytu.

Funkcja READ_CARD_BY_NUMBER zwraca dane karty na podstawie numeru karty podanego w parametrze wejściowym. Dane są zwracane jako rekordy typu card_type i przetwarzane strumieniowo (PIPELINED).

Funkcja READ_CARD_BY_ACCOUNT_NUMBER zwraca wszystkie karty powiązane z określonym numerem konta. Dane są zwracane w formacie strumieniowym (PIPELINED), co pozwala na ich wydajne przetwarzanie.

Procedura UPDATE_CARD_LIMITS aktualizuje limity dzienne i limity pojedynczych płatności dla wskazanej karty. Jeśli operacja przebiegnie pomyślnie, zmiany są zatwierdzane, a wynik zawiera odpowiednią informację. W przypadku błędu zwracana jest stosowna wiadomość.

Procedura UPDATE_CARD_STATUS umożliwia zmianę statusu aktywności karty na podstawie numeru karty. Przyjmuje parametr p_is_active, który decyduje, czy karta ma być aktywna (1) czy nieaktywna (0). Zmiany są zatwierdzane, a wynik operacji odpowiednio aktualizowany.

Procedura UPDATE_CARD_PIN pozwala na zmianę PIN-u karty na podstawie jej numeru. Nowy PIN jest przekazywany jako parametr wejściowy p_PIN. Po pomyślnej aktualizacji zmiany są zatwierdzane, a wynik zawiera informację o powodzeniu operacji.

Procedura DELETE_CARD usuwa kartę z bazy danych na podstawie numeru karty. Jeśli karta nie zostanie odnaleziona, zwracana jest odpowiednia informacja. W przypadku błędów baza danych przywracana jest do stanu sprzed operacji.

Tabela TRANSACTION

CREATE_TRANSACTION służy do tworzenia nowej transakcji pomiędzy dwoma kontami. Procedura przyjmuje numer konta nadawcy, numer konta odbiorcy, kwotę transakcji, typ transakcji oraz opis transakcji. W przypadku, gdy dane wejściowe są niepoprawne, np. numery kont są nieprawidłowe lub saldo nadawcy jest niewystarczające, procedura zwraca odpowiedni komunikat o błędzie. Jeśli wszystko jest poprawne, procedura aktualizuje salda na kontach nadawcy i odbiorcy, a także zapisuje transakcję w tabeli TRANSACTION. Jeśli wystąpią jakiekolwiek błędy, transakcja jest wycofywana, a użytkownik otrzymuje komunikat o błędzie.

READ_ALL_TRANSACTIONS_FUNC zwraca wszystkie transakcje zapisane w tabeli TRANSACTION. Funkcja ta wykorzystuje podejście strumieniowe (PIPELINED), aby zwrócić dane transakcji jako rekordy transaction_obj. Działa to efektywnie przy dużych zbiorach

danych, umożliwiając użytkownikowi dostęp do wszystkich transakcji zapisanych w bazie danych.

READ_TRANSACTIONS_BY_ACCOUNT_FUNC pozwala na pobranie transakcji związanych z konkretnym kontem. Funkcja ta przyjmuje numer konta i zwraca wszystkie transakcje, w których dane konto występuje jako nadawca lub odbiorca. Podobnie jak poprzednia funkcja, używa podejścia strumieniowego do zwrócenia wyników w postaci rekordów transaction_obj.

Tabela RECEIVER

CREATE_RECEIVER służy do tworzenia nowego odbiorcy. Przyjmuje numer konta odbiorcy, numer konta powiązanego, opis, imię i nazwisko odbiorcy. Metoda sprawdza poprawność danych wejściowych, takich jak długość numeru konta, format cyfr oraz poprawność imion i nazwisk. Jeśli dane są prawidłowe, procedura wstawia nowego odbiorcę do tabeli RECEIVER. W przypadku wystąpienia błędów, np. brakujących wymaganych pól, użytkownik otrzymuje odpowiedni komunikat.

READ_ALL_RECEIVERS_FUNC to funkcja, która zwraca wszystkich odbiorców z tabeli RECEIVER. Działa na zasadzie strumienia (PIPELINED), zwracając dane o każdym odbiorcy w postaci rekordu receiver_obj. Funkcja ta umożliwia pobranie pełnej listy odbiorców z bazy.

READ_RECEIVER_BY_TIED_ACCOUNT_FUNC to funkcja, która umożliwia wyszukiwanie odbiorców na podstawie numeru konta powiązanego. Funkcja ta zwraca listę odbiorców, których numer konta powiązanego odpowiada podanemu w parametrze. Zawiera także walidację danych, umożliwiającą efektywne przetwarzanie wyników.

DELETE_RECEIVER to procedura, która usuwa odbiorcę z tabeli RECEIVER na podstawie numeru konta odbiorcy oraz numeru konta powiązanego. Po wykonaniu operacji sprawdza, czy usunięto jakiś rekord i zwraca odpowiedni komunikat. W przypadku błędów, np. nieistniejącego odbiorcy, procedura zwraca komunikat o niepowodzeniu.

Inne ważne komponenty kodu PL/SQL

Wyzwalacze oraz operacje cykliczne

```
CREATE SEQUENCE transaction_id_seq
START WITH 1
INCREMENT BY 1
NOCACHE
NOCYCLE;
/

CREATE OR REPLACE TRIGGER trg_transaction_id
BEFORE INSERT ON "TRANSACTION"
FOR EACH ROW
BEGIN
    :NEW.ID := transaction_id_seq.NEXTVAL;
END;
/
```

Trigger trg_transaction_id automatycznie przypisuje unikalny identyfikator do kolumny ID w tabeli TRANSACTION przed każdą próbą wstawienia nowego rekordu. Wartość ta jest generowana przy użyciu sekwencji transaction_id_seq, która zaczyna się od 1 i inkrementuje o 1 dla każdego nowego rekordu. Dzięki temu zapewniona jest unikalność identyfikatorów transakcji w tabeli, a proces wstawiania danych jest automatycznie zautomatyzowany.

```
PROCEDURE CREATE_RANSOM_JOB IS
    v_result VARCHAR2(200);
BEGIN
    DBMS_SCHEDULER.create_job (
        job_name        => 'RANSOM_JOB',
        job_type        => 'PLSQL_BLOCK',
        job_action      => 'BEGIN pakiety_pkg.CREATE_RANSOM_TRANSACTION; END;',
        start_date      => SYSTIMESTAMP,
        repeat_interval => 'FREQ=MONTHLY; BYMONTHDAY=1; BYHOUR=0; BYMINUTE=0; BYSECOND=1',
        enabled         => TRUE
    );
END CREATE_RANSOM_JOB;

PROCEDURE CREATE_RANSOM_TRANSACTION IS
    v_account_number CHAR(12);
    v_receiver_account_number CHAR(26) := '00000000000000000000000000';
    v_amount NUMBER := 500;
    v_type_of_transaction VARCHAR2(50) := 'Ransom';
    v_description VARCHAR2(200) := 'Ransom payment';
    v_result VARCHAR2(200);
    CURSOR account_cursor IS
        SELECT account_number FROM ACCOUNT;
BEGIN
    FOR rec IN account_cursor LOOP
        transaction_pkg.CREATE_TRANSACTION(
            p_account_number => rec.account_number,
            p_account_number_receiver => v_receiver_account_number,
            p_amount => v_amount,
            p_type_of_transaction => v_type_of_transaction,
            p_description_of_transaction => v_description,
            p_result => v_result
        );
    END LOOP;
END CREATE_RANSOM_TRANSACTION;
```

Procedura CREATE_RANSOM_TRANSACTION tworzy transakcje typu ransom dla wszystkich kont w tabeli ACCOUNT. Dla każdego konta wykonuje transakcję na stałą kwotę, z określonym numerem konta odbiorcy. Parametry transakcji przekazywane są do procedury CREATE_TRANSACTION w pakiecie transaction_pkg. Celem procedury jest generowanie cyklicznych płatności dla wszystkich użytkowników systemu.

Typy danych

```
TYPE client_obj IS RECORD (
    pesel        CHAR(11),
    first_name   VARCHAR2(50),
    last_name    VARCHAR2(50),
    middle_name  VARCHAR2(50),
    phone_number VARCHAR2(20),
    date_of_birth DATE
);

TYPE client_tbl IS TABLE OF client_obj;
```

Typ client_obj to rekord, który przechowuje dane dotyczące klienta. Zawiera numer PESEL klienta jako ciąg 11 cyfr, imię, nazwisko i drugie imię klienta, które mogą mieć do 50 znaków każde. Dodatkowo przechowuje numer telefonu klienta, który może zawierać do 20 znaków, oraz datę urodzenia klienta, zapisaną w formacie DATE.

Typ client_tbl to tabela typu client_obj, która przechowuje kolekcję rekordów tego typu, umożliwiając przechowywanie wielu klientów w jednej zmiennej tabelarycznej.

Typ client_tbl jest lustrzanym odbiciem tabeli CLIENT. Ten typ jest wykorzystywany do efektywnego przekazywania danych do części frontendowej aplikacji.

Każda tabela posiada swój odpowiednik w postaci takiej tabeli, który jest wykorzystywany do operacji READ.

Definicje typów danych znajdują się w odpowiadających im pakietom. Np client_obj i client_tbl znajdują się w pliku cleint_pkg.sql.

Zarządzanie danymi i bezpieczeństwo

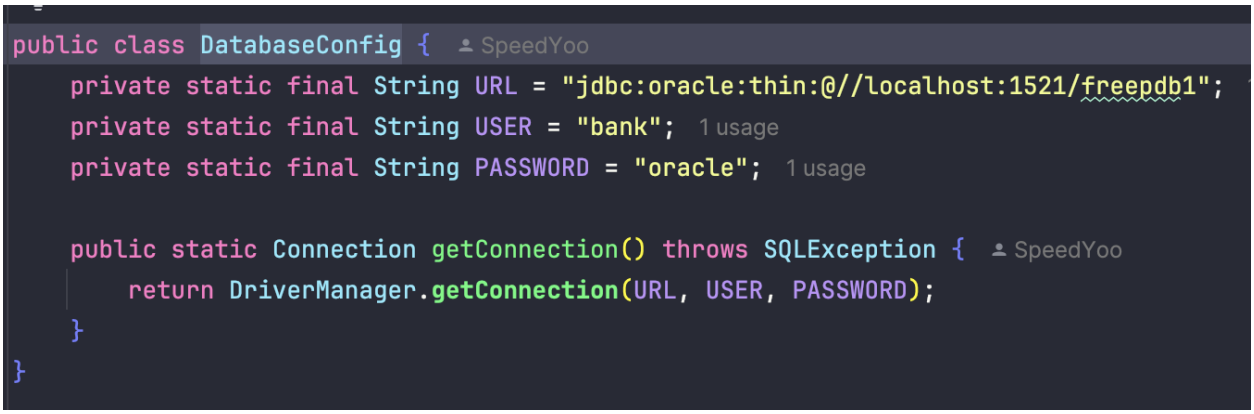
Dane są zarządzane przy pomocy procedur oraz funkcji w języku PL/SQL. Są one uporządkowane w pakiety.

Pakiety client_pkg, account_pkg, card_pkg, client_account_pkg, receiver_pkg, transation_pkg posiadają odpowiadające im zestawy operacji CRUD. Są one przystosowane do chronienia wrażliwych informacji i wychwytują błędy użytkownika.

Hasło do konta jest zabezpieczone przez haszowane z użyciem algorytmu SHA-256 oraz soli.

Aplikacja

Aplikacja w Javie stanowi graficzny interfejs użytkownika do systemu bankowego, który umożliwia klientom oraz administratorom zarządzanie danymi bazy danych, przeprowadzanie transakcji i wykonywanie innych operacji bankowych. Aplikacja korzysta z aplikacyjnej warstwy logiki (Java classes) i obsługuje bazę danych za pomocą odpowiednich repozytoriów, które komunikują się z bazą danych poprzez procedury i funkcje. Aplikacja korzysta z biblioteki JDBC do łączenia się z bazą danych oraz z javaFX do tworzenia interfejsu graficznego



przykładowa konfiguracja połączenia z bazą w aplikacji

Struktura

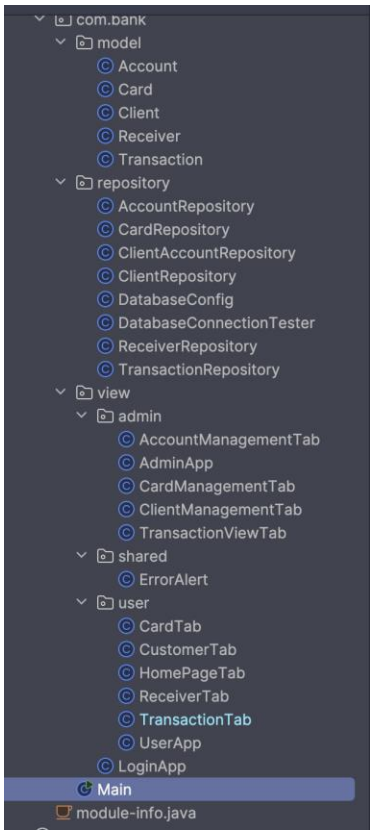
Aplikacja znajduje się w na githubie w folderze bank w środku folderze src został umieszczony kod źrodłowy aplikacji, aplikacja jest złożona z wielu klas podzielonych na 3 główne pakiety: model, repository oraz view, pakiety te znajdują się w com.bank.

Klasy w pakiecie model takie jak Account, Card, Transaction, Client zawierają encje biznesowe które odzwierciedlają rekordy tabeli bazy danych są używane do przechowywania danych, które mogą być wyświetlane lub edytowane przez użytkowników aplikacji.

Klasy z pakietu Repository obsługują połączenie z bazą danych oraz wywołują odpowiednie procedury i funkcje z bazy danych z operacjami CRUD. Każda klasa z pakietu repository jest odpowiedzialna za jedną tabele z bazy danych zapewnia, że wszystkie operacje wykonywane na danych są zgodne z wymaganiami systemu.

Klasy pakietu View odpowiedzialne są za interfejs graficzny aplikacji. Okno logowania z klasy LoginApp tworzy proste okno do zalogowania się do bazy danych które rówież informuje użytkownika o tym czy udało się połączenie z bazą danych. Aplikacja została podzielona na dwie główne częsci na interfejs admina i użytkownika banku.

Odpowiadają za to dwie klasy AdminApp z pakietu view.admin i UserApp z pakietu view.user



Struktura projektu

Opis działania aplikacji

Aplikacja uruchamia się z klasy Main w pakiecie com.bank

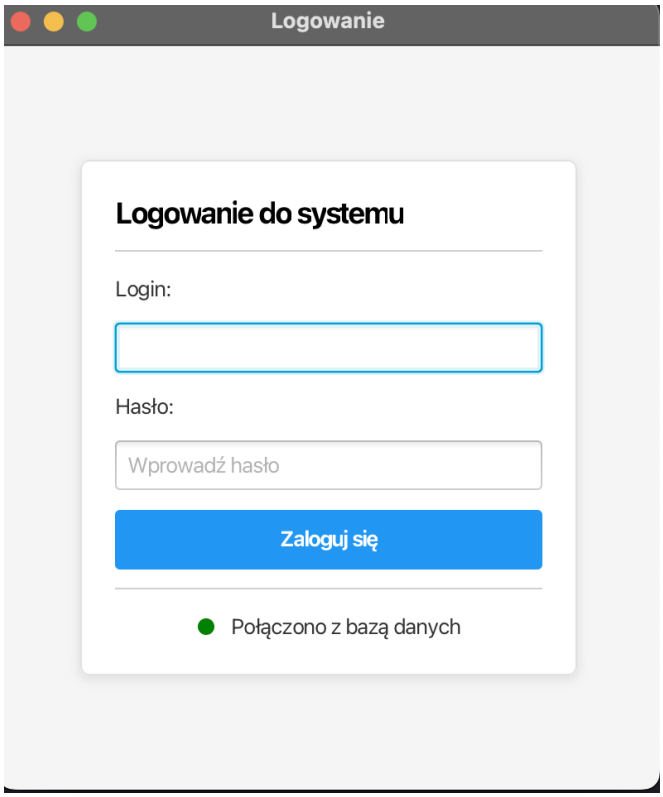
```java
public class Main extends Application{  SpeedYoo
    @Override  SpeedYoo
    public void start(Stage primaryStage) {
        LoginApp loginApp = new LoginApp();
        loginApp.showLoginWindow(primaryStage);
    }

    public static void main(String[] args) {  SpeedYoo
        launch(args);  //LAUNCH GUI !!!
    }
}
```

Uruchomienie metody Main wywołuje utworzenie okna logowania

Klasa LoginApp sprawdza, czy udało się połączyć z bazą danych

```java
if (DatabaseConnectionTester.testConnection()) {
    statusCircle.setFill(Color.GREEN);
    dbStatusLabel.setText("Połączono z bazą danych");
} else {
    statusCircle.setFill(Color.RED);
    dbStatusLabel.setText("Brak połączenia z bazą danych");
    loginButton.setDisable(true);
}
```

Jeśli połączenie się udało można zalogować się do bazy danych wpisując login i hasło



Tak wygląda okno logowania do aplikacji, jeśli połączenie zostało nawiązane

```
    String output = AccountRepository.login(login, password);
    System.out.println(output);
    if (output.contains("Login successful")) {
        Pattern pattern = Pattern.compile( regex: "(\\d+)$");
        Matcher matcher = pattern.matcher(output);
        if (matcher.find()) {
            String accountNumber = matcher.group(1);
            UserApp.setCurrent_login_account_number(accountNumber);
        }
        primaryStage.close();
        UserApp userApp = new UserApp();
        userApp.showMainWindow();
    } else {
        showAlert( title: "Błąd logowania",  content: "Błędne hasło lub login");
    }
});
```

Aplikacja sprawdza czy dane logowania są prawidłowe za pomocą metody z klasy AccountRepository, metoda login z tej klasy wywołuje procedurę do logowania się i jeśli dane są poprawne zwraca numer konta użytkownika. Jeśli proces logowania użytkownika przebiegł pomyślnie ukazuje się okno aplikacji banku dla użytkownika.

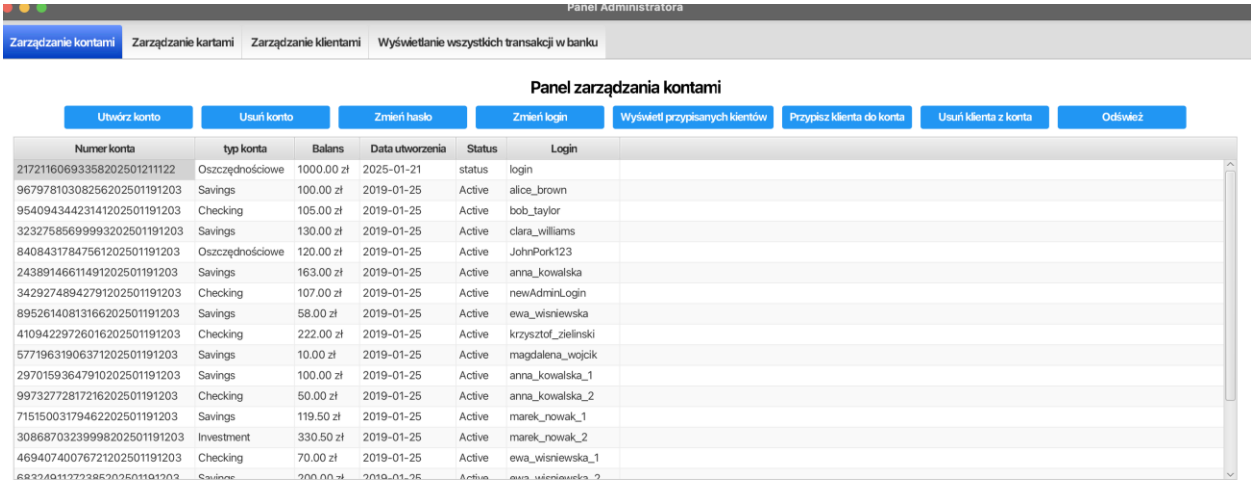
Aplikacja również pozwala zalogować się jako administrator

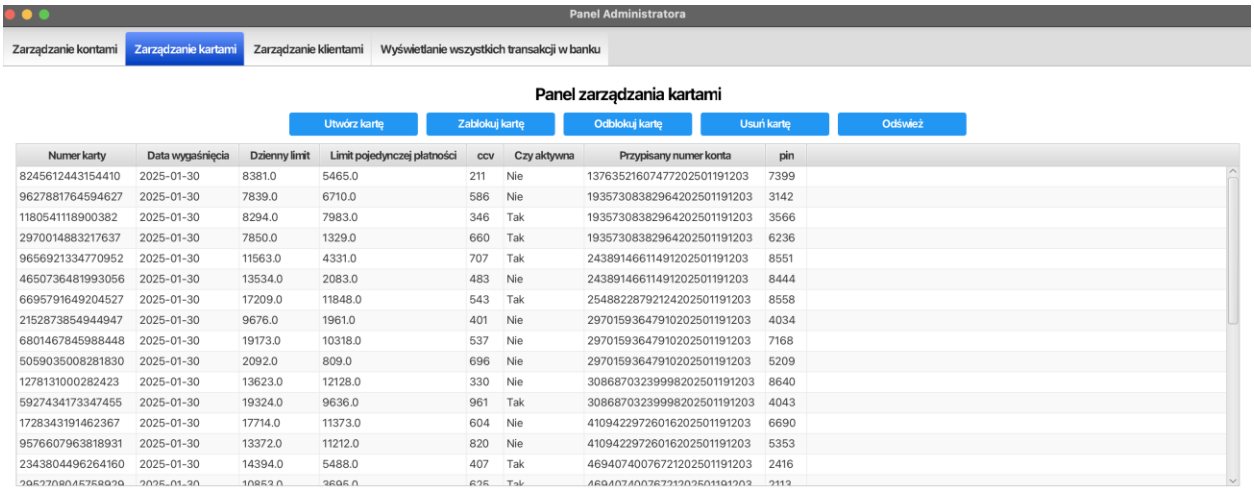
```
if (login.equals("admin") && password.equals("admin")) {
    primaryStage.close();
    AdminApp adminApp = new AdminApp();
    adminApp.showMainWindow();
    return;
}
```

Do zalogowania się jako administrator zostało ustawione tymczasowy login i hasło admin admin.

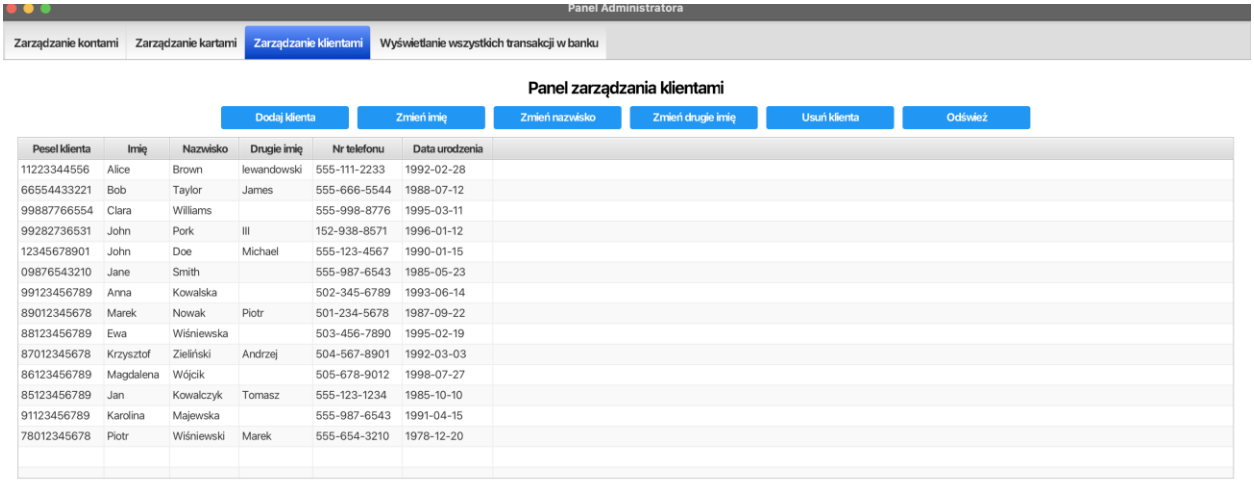
Jeśli login i hasło się zgadzają otwiera się okno do zarządzania bazą danych jako administrator

Okno admina bazy danych

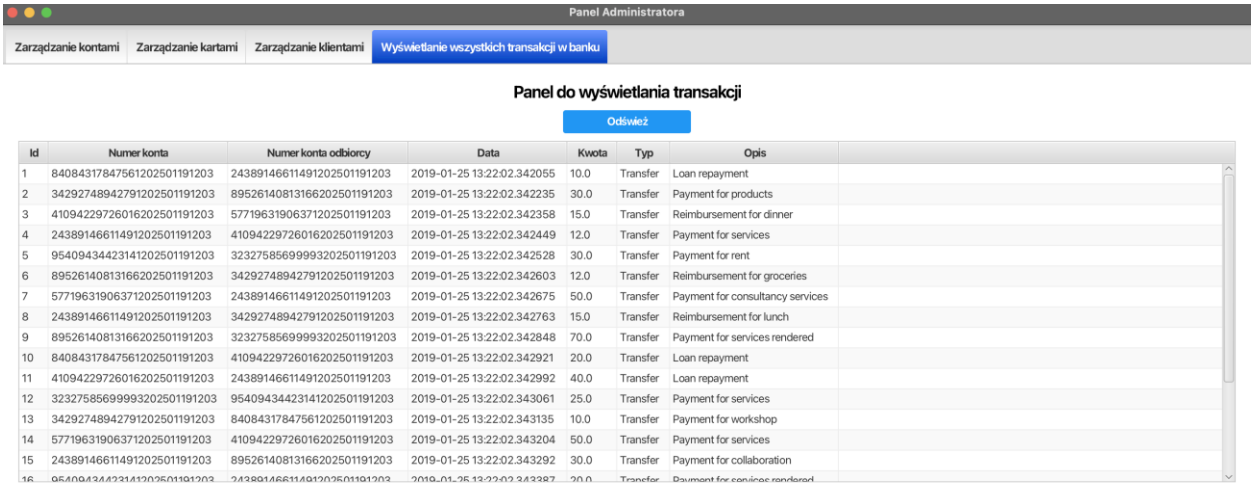




Numer karty	Data wygaśnięcia	Dzienny limit	Limit pojedynczej płatności	ccv	Czy aktywna	Przypisany numer konta	pin
8245612443154410	2025-01-30	8381.0	5465.0	211	Nie	13763521607477202501191203	7399
9627881764594627	2025-01-30	7839.0	6710.0	586	Nie	19357308382964202501191203	3142
1180541118900382	2025-01-30	8294.0	7983.0	346	Tak	19357308382964202501191203	3566
2970014883217637	2025-01-30	7850.0	1329.0	660	Tak	19357308382964202501191203	6236
9656921334770952	2025-01-30	11563.0	4331.0	707	Tak	24389146611491202501191203	8551
4650736481993056	2025-01-30	13534.0	2083.0	483	Nie	24389146611491202501191203	8444
6695791649204527	2025-01-30	17209.0	11848.0	543	Tak	25488228792124202501191203	8558
2152873854944947	2025-01-30	9676.0	1961.0	401	Nie	29701593647910202501191203	4034
6801467845988448	2025-01-30	19173.0	10318.0	537	Nie	29701593647910202501191203	7168
5059035008281830	2025-01-30	2092.0	809.0	696	Nie	29701593647910202501191203	5209
1278131000282423	2025-01-30	13623.0	12128.0	330	Nie	30868703239998202501191203	8640
5927434173347455	2025-01-30	19324.0	9636.0	961	Tak	30868703239998202501191203	4043
1728343191462367	2025-01-30	17714.0	11373.0	604	Nie	41094229726016202501191203	6690
9576607963818931	2025-01-30	13372.0	11212.0	820	Nie	41094229726016202501191203	5353
2343804496264160	2025-01-30	14394.0	5488.0	407	Tak	46940740076721202501191203	2416
2952709045758939	2025-01-30	10853.0	3695.0	625	Tak	46940740076721202501191203	2113



Pesel klienta	Imię	Nazwisko	Drugie imię	Nr telefonu	Data urodzenia
11223344556	Alice	Brown	lewandowski	555-111-2233	1992-02-28
66554433221	Bob	Taylor	James	555-666-5544	1988-07-12
99887766554	Clara	Williams		555-998-8776	1995-03-11
99282736531	John	Pork	III	152-938-8571	1996-01-12
12345678901	John	Doe	Michael	555-123-4567	1990-01-15
09876543210	Jane	Smith		555-987-6543	1985-05-23
99123456789	Anna	Kowalska		502-345-6789	1993-06-14
89012345678	Marek	Nowak	Piotr	501-234-5678	1987-09-22
88123456789	Ewa	Wiśniewska		503-456-7890	1995-02-19
87012345678	Krzysztof	Zieliński	Andrzej	504-567-8901	1992-03-03
86123456789	Magdalena	Wójcik		505-678-9012	1998-07-27
85123456789	Jan	Kowalczyk	Tomasz	555-123-1234	1985-10-10
91123456789	Karolina	Majewska		555-987-6543	1991-04-15
78012345678	Piotr	Wiśniewski	Marek	555-654-3210	1978-12-20



Id	Numer konta	Numer konta odbiorcy	Data	Kwota	Typ	Opis
1	84084317847561202501191203	24389146611491202501191203	2019-01-25 13:22:02.342055	10.0	Transfer	Loan repayment
2	34292748942791202501191203	89526140813166202501191203	2019-01-25 13:22:02.342235	30.0	Transfer	Payment for products
3	41094229726016202501191203	57719631906371202501191203	2019-01-25 13:22:02.342358	15.0	Transfer	Reimbursement for dinner
4	24389146611491202501191203	41094229726016202501191203	2019-01-25 13:22:02.342449	12.0	Transfer	Payment for services
5	95409434423141202501191203	32327585699993202501191203	2019-01-25 13:22:02.342528	30.0	Transfer	Payment for rent
6	89526140813166202501191203	34292748942791202501191203	2019-01-25 13:22:02.342603	12.0	Transfer	Reimbursement for groceries
7	57719631906371202501191203	24389146611491202501191203	2019-01-25 13:22:02.342675	50.0	Transfer	Payment for consultancy services
8	24389146611491202501191203	34292748942791202501191203	2019-01-25 13:22:02.342763	15.0	Transfer	Reimbursement for lunch
9	89526140813166202501191203	32327585699993202501191203	2019-01-25 13:22:02.342848	70.0	Transfer	Payment for services rendered
10	84084317847561202501191203	41094229726016202501191203	2019-01-25 13:22:02.342921	20.0	Transfer	Loan repayment
11	41094229726016202501191203	24389146611491202501191203	2019-01-25 13:22:02.342992	40.0	Transfer	Loan repayment
12	32327585699993202501191203	95409434423141202501191203	2019-01-25 13:22:02.343061	25.0	Transfer	Payment for services
13	34292748942791202501191203	84084317847561202501191203	2019-01-25 13:22:02.343135	10.0	Transfer	Payment for workshop
14	57719631906371202501191203	41094229726016202501191203	2019-01-25 13:22:02.343204	50.0	Transfer	Payment for services
15	24389146611491202501191203	89526140813166202501191203	2019-01-25 13:22:02.343292	30.0	Transfer	Payment for collaboration
16	95409434423141202501191203	24389146611491202501191203	2019-01-25 13:22:02.343387	20.0	Transfer	Payment for services rendered

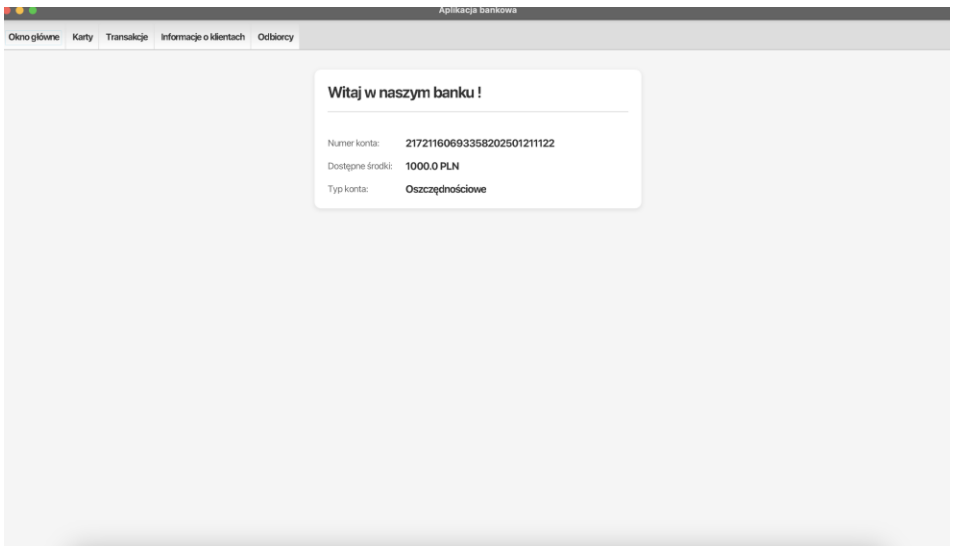
```java
public void showMainWindow() {  1 usage  SpeedYoo
    stage = new Stage();
    TabPane tabPane = new TabPane();
    tabPane.setTabClosingPolicy(TabPane.TabClosingPolicy.UNAVAILABLE);

    tabPane.getTabs().addAll(
            AccountManagementTab.getTab(),
            CardManagementTab.getTab(),
            ClientManagementTab.getTab(),
            TransactionViewTab.getTab()
    );
```

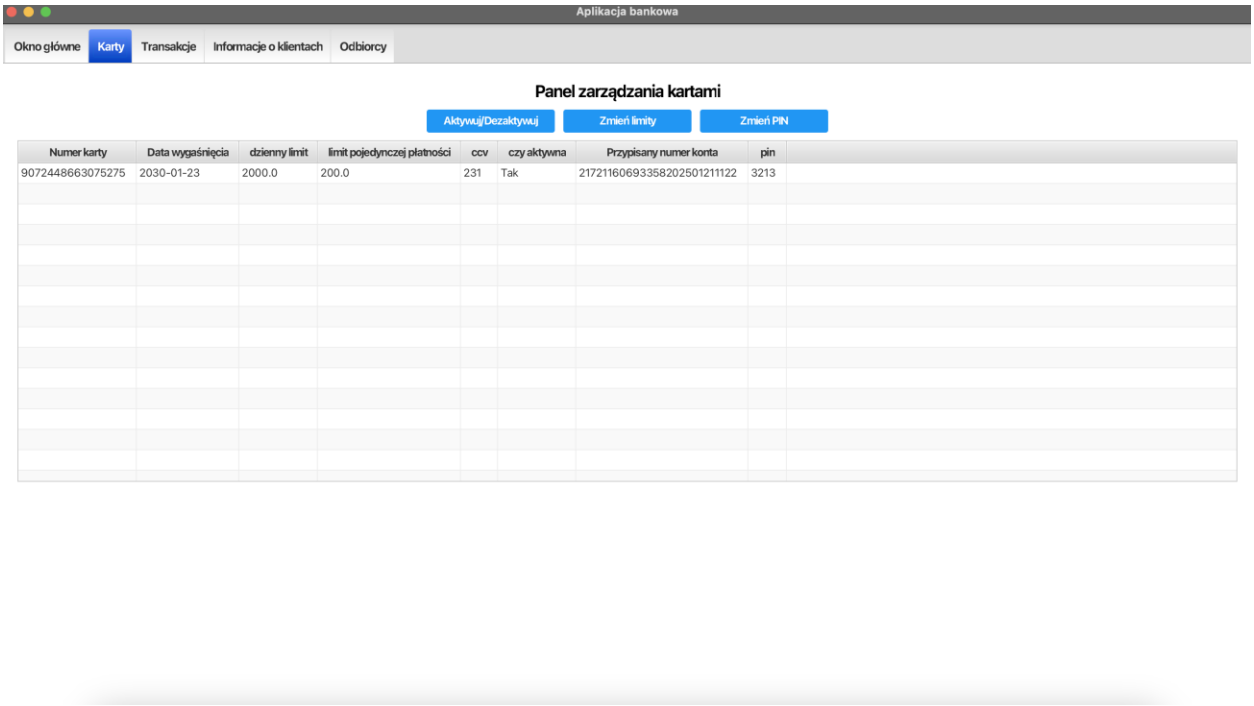
Widok administratora w aplikacji został podzielony na zakładki. Każda zakładka została podzielona na osobną klasę

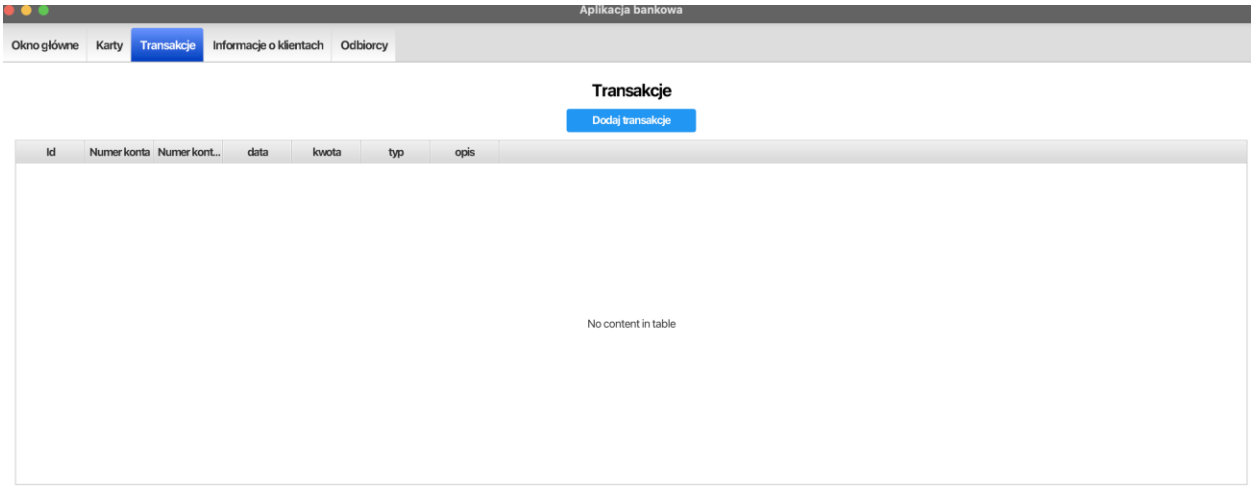
Każda klasa z zakładką wyświetla dane w postaci tabelki oraz przyciski, za pomocą których wywoływane są odpowiednie metody z klas repository, które wywołują procedury lub funkcje w bazie danych

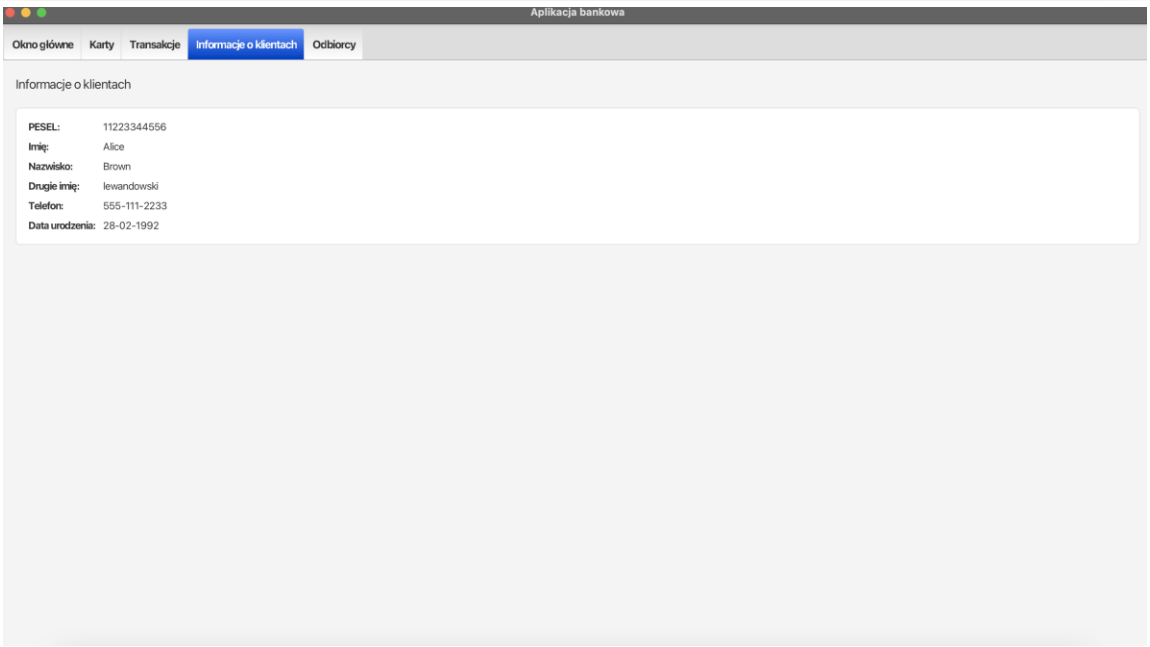
Do wykonania niektórych operacji takich jak usuwanie klienta należy wybrać odpowiedni rekord z tabeli a następnie kliknąć przycisk

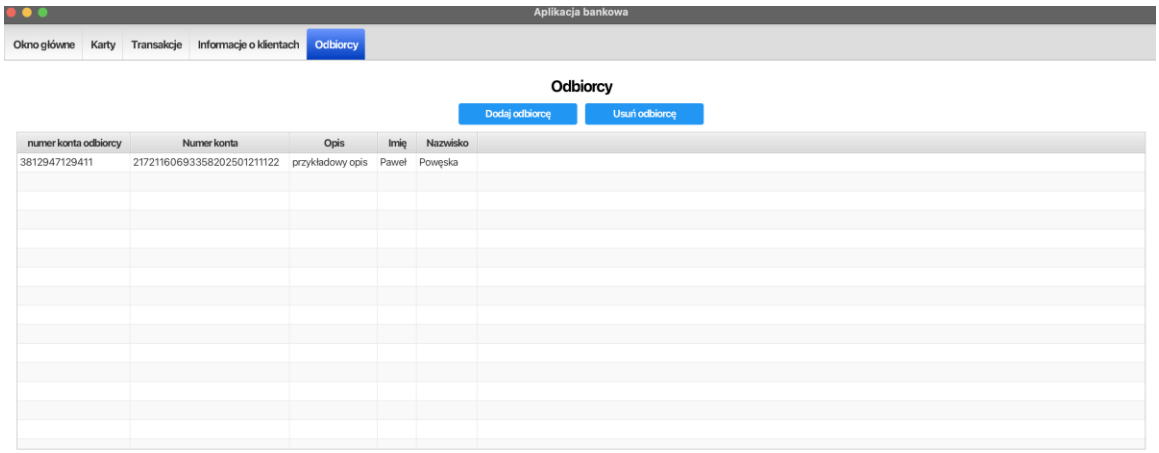
Widok dla klienta




Aplikacja bankowa
Okno główne
Karty
Transakcje
Informacje o klientach
Odbiorcy
Panel zarządzania kartami
Aktywuj/Dezaktywuj
Zmień limity
Zmień PIN
Numer karty
Data wygaśnięcia
dzienny limit
limit pojedynczej płatności
ccv
czy aktywna
Przypisany numer konta
pin
9072448663075275
2030-01-23
2000.0
200.0
231
Tak
21721160693358202501211122
3213


Aplikacja bankowa
Okno główne
Karty
Transakcje
Informacje o klientach
Odbiorcy
Transakcje
Dodaj transakcje
Id
Numer konta
Numer kont...
data
kwota
typ
opis
No content in table


Aplikacja bankowa
Okno główne
Karty
Transakcje
Informacje o klientach
Odbiorcy
Informacje o klientach
PESEL: 11223344556
Imię: Alice
Nazwisko: Brown
Drugie imię: lewandowski
Telefon: 555-111-2233
Data urodzenia: 28-02-1992



Widok dla użytkownika w aplikacji działa na podobnej zasadzie jak widok admina, również składa się z zakładek, każda zakładka jest generowana przez osobną klasę

```
public void showMainWindow() {  1 usage   SpeedYoo
    TabPane tabPane = new TabPane();
    tabPane.setTabClosingPolicy(TabPane.TabClosingPolicy.UNAVAILABLE);
    tabPane.getTabs().addAll(HomePageTab.homePage(),
            CardTab.cardsTab(),
            TransactionTab.transactionsTab(),
            CustomerTab.customerInfoTab(),
            ReceiverTab.receiversTab());
```

Funkcjonalności systemu

System składający się z aplikacji w Javie oraz bazy danych pozwalają użytkownikowi na między innymi:

- Możliwość dokonywania przelewów pomiędzy kontami co automatycznie dokona zmian w balansach obydwu kont
- Logowanie się do aplikacji przy użyciu loginu oraz hasła, które jest haszowane w procedurze

- Dodawania odbiorców i możliwość szybkiego wybierania odbiorców w celu szybkiego dokonania transakcji
- Pobieranie miesięcznej opłaty za posiadanie konta
- Tworzenie, aktualizowanie, usuwanie oraz wyświetlanie rekordów tabel (operacje CRUD)
- Graficzny interfejs użytkownika, który pozwala na automatyczne operowanie na wybranych wierszach tabeli

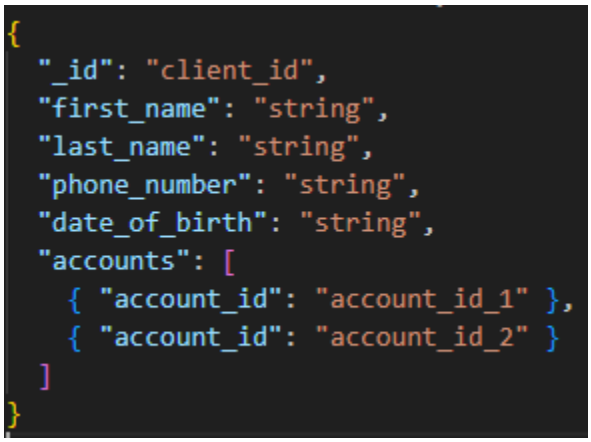
Założenia migracja na nierelacyjną bazę danych

W ramach tego projektu, dokonaliśmy analizy struktury bazy danych oraz zaprezentowaliśmy sposób przechowywania danych w formacie dokumentów JSON/BSON. Konwersja na bazę danych MongoDB ma na celu poprawę skalowalności, elastyczności, a także optymalizację operacji związanych z przechowywaniem danych oraz ich przetwarzaniem.

Po migracji do MongoDB struktura danych, które były przechowywane w tabelach relacyjnych, przybiera formę dokumentów BSON, które mogą zawierać zagnieżdżone dane. W tym przypadku projekt obejmuje trzy główne obiekty: Klientów, Kont oraz Transakcje, a także dodatkowe elementy, takie jak Karty i Odbiorcy, które są powiązane z kontami.

W wyniku migracji tabela client_account występująca w SQL zostaje porzucona. Dzięki elastyczności nierelacyjnych baz danych nie jest ona konieczna do stworzenia relacji wiele do wielu pomiędzy tabelą account oraz client.

- Client:



Klient w MongoDB jest dokumentem, który zawiera pole accounts będące tablicą, w której przechowywane są referencje do powiązanych kont. Dzięki tej strukturze, łatwo jest uzyskać dostęp do wszystkich kont powiązanych z danym klientem.

- Account:

```
{
    "_id": "account_id",
    "type_of_account": "string",
    "balance": "number",
    "date_of_creation": "string",
    "status": "string",
    "cards": [
      {
        "_id": "card_id_1",
        "card_number": "string",
        "date_of_expiration": "string",
        "daily_limit": "number",
        "single_payment_limit": "number",
        "CVV": "string",
        "is_active": "boolean",
        "account_id": "account_id",
        "PIN": "string"
      },
      {
        "_id": "card_id_2",
        "card_number": "string",
        "date_of_expiration": "string",
        "daily_limit": "number",
        "single_payment_limit": "number",
        "CVV": "string",
        "is_active": "boolean",
        "account_id": "account_id",
        "PIN": "string"
      }
    ],
    "receivers": [
      {
        "_id": "receiver_id_1",
        "account_number": "receiver_account_id_1",
        "account_id": "account_id",
        "description": "Receiver description",
        "first_name": "Receiver First Name",
        "last_name": "Receiver Last Name"
      },
      {
        "_id": "receiver_id_2",
        "account_number": "receiver_account_id_2",
        "account_id": "account_id",
        "description": "Another receiver description",
        "first_name": "Receiver First Name 2",
        "last_name": "Receiver Last Name 2"
      }
    ]
}
```

Konta są reprezentowane przez dokumenty, które zawierają informacje o kartach, odbiorcach oraz innych powiązanych danych, takich jak saldo, status i typ konta. Referencja do klientów pozwala łątwy dostęp do nich.

- Transactions:

Transakcje są przechowywane w dokumentach, które zawierają referencje do kont nadawcy i odbiorcy, a także powiązanych klientów. Taki model umożliwia łatwe zarządzanie transakcjami, ich historią oraz śledzenie powiązań między klientami a transakcjami.

Benefity pochodzące z migracji:

Elastyczność: MongoDB umożliwia przechowywanie złożonych, zagnieżdżonych struktur danych, co jest idealne dla systemów wymagających przechowywania złożonych informacji, takich jak karty, odbiorcy czy transakcje.

Skalowalność: MongoDB wspiera poziome skalowanie, co pozwala na łatwe zarządzanie dużymi zbiorami danych, które rosną w miarę rozwoju systemu.

Szerokie wsparcie dla JSON/BSON: Przechowywanie danych w formacie JSON/BSON zapewnia dużą elastyczność w modelowaniu struktury danych oraz wygodne mapowanie danych między aplikacjami a bazą danych.

Instalacja systemu Bankowego

Pełna instrukcja instalacji znajduje się w pliku README.md na GitHubie:

https://github.com/jakub7038/System-bankowy/commits/main/