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 2](#_Toc110182101)

[Cele i zadania bazy danych 2](#_Toc2008060135)

[Baza danych 3](#_Toc2143780246)

[Architektura i Schemat bazy danych 3](#_Toc833221442)

[Opis Struktury bazy danych 4](#_Toc803334048)

[Szczegółowy opis metod CRUD dla tabeli CLIENT 7](#_Toc1023829326)

[Operacja CREATE 8](#_Toc1809637828)

[Operacje READ 9](#_Toc303164950)

[Operacje UPDATE 11](#_Toc307659976)

[Operacja DELETE 13](#_Toc1046165363)

[Poglądowy opis metod CRUD dla pozostałych tabel 15](#_Toc1302293018)

[Tabela ACCOUNT 15](#_Toc1666750064)

[Tabela CLIENT\_ACCOUNT 16](#_Toc961489731)

[Tabela CARD 16](#_Toc907206921)

[Tabela TRANSACTION 17](#_Toc126853735)

[Tabela RECEIVER 18](#_Toc1053420259)

[Inne ważne komponenty kodu PL/SQL 18](#_Toc1643866172)

[Wyzwalacze oraz operacje cykliczne 19](#_Toc714606374)

[Typy danych 20](#_Toc311458954)

[Zarządzanie danymi i bezpieczeństwo 21](#_Toc782954450)

[Aplikacja 21](#_Toc159072007)

[Struktura 22](#_Toc717305176)

[Opis działania aplikacji 23](#_Toc287285735)

[Funkcjonalności systemu 30](#_Toc117830719)

[Założenia migracja na nierelacyjną bazę danych 30](#_Toc1390670756)

[Benefity pochodzące z migracji: 30](#_Toc971584640)

[Instalacja systemu Bankowego 30](#_Toc693303876)

# 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.

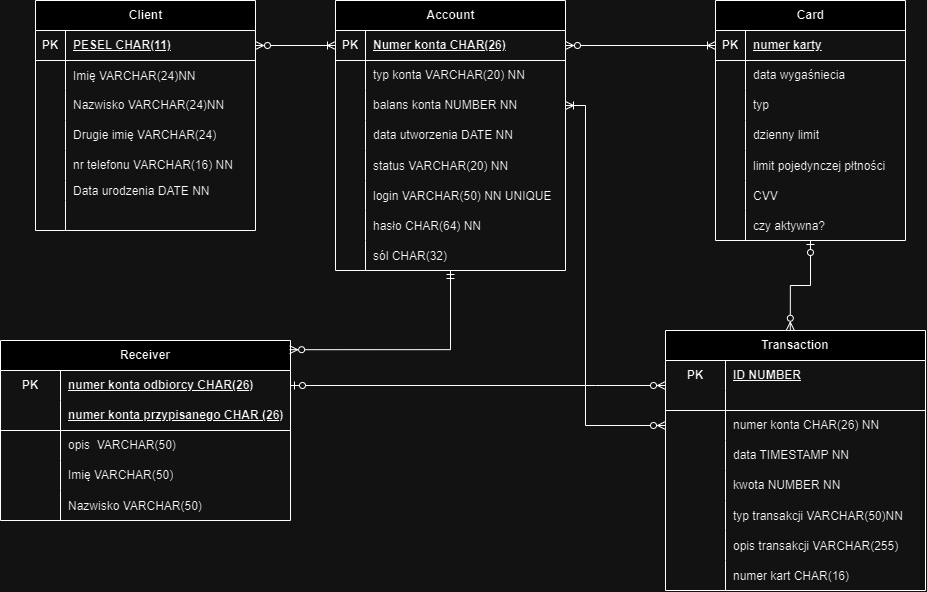
# **Baza danych**

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.

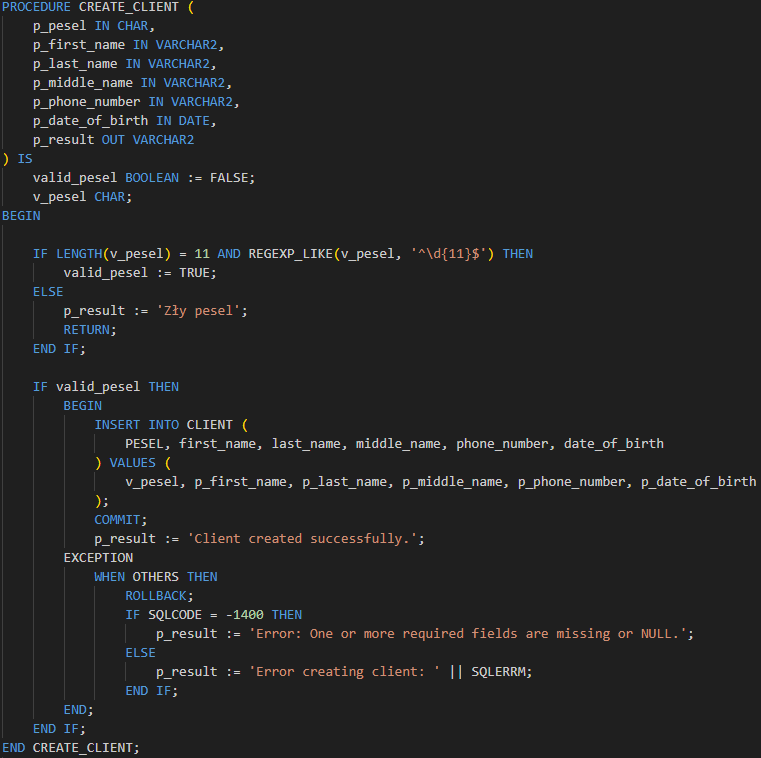
Tabela TRANSACTION:

* 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**

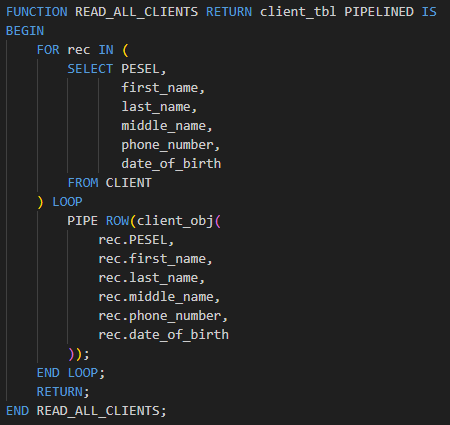
Procedury oraz funkcje CRUD dla tabel znajdują się w odpowiadających im pakietom. Na przykład CRUD dla tabeli CLIENT znajduje się w pliku client\_pkg.sql.

### Operacja CREATE

CREATE\_CLIENT 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.

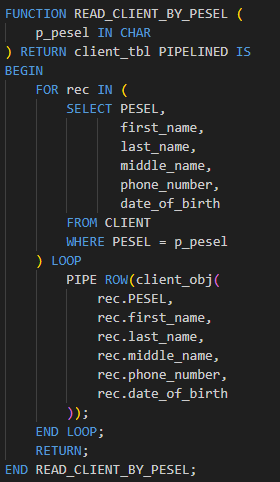
### Operacje READ

#### READ\_ALL\_CLIENTS

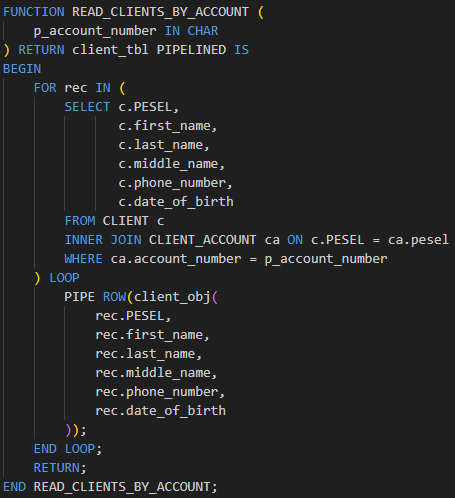


Funkcja ta zwraca wszystkie rekordy z tabeli CLIENT. Iteruje po każdym rekordzie tabeli, a następnie mapuje dane na obiekty typu client\_obj. Funkcja pozwala uzyskać pełną listę klientów przechowywanych w bazie danych, niezależnie od ich liczby. Jest potrzebna do wyświetlenia wszystkich rekordów tabeli,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

READ\_CLIENT\_BY\_PESEL

  
 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.

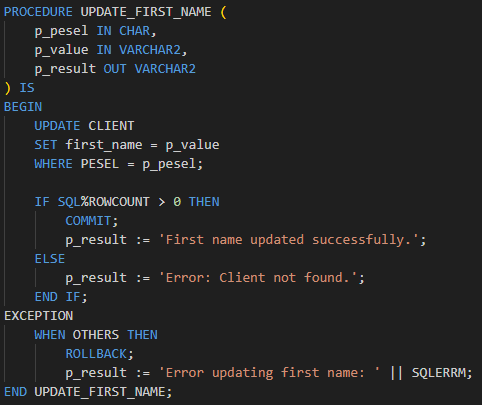
#### READ\_CLIENT\_BY\_ACCOUNT



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.

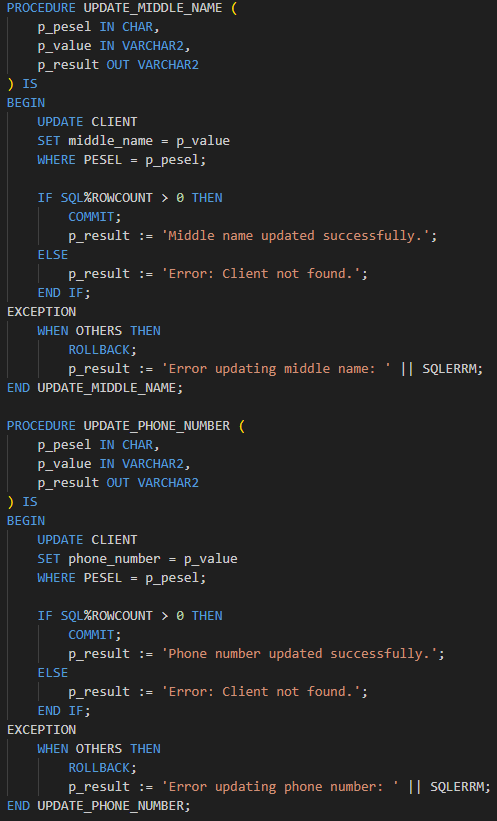
### Operacje UPDATE

#### UPDATE\_FIRST\_NAME



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).

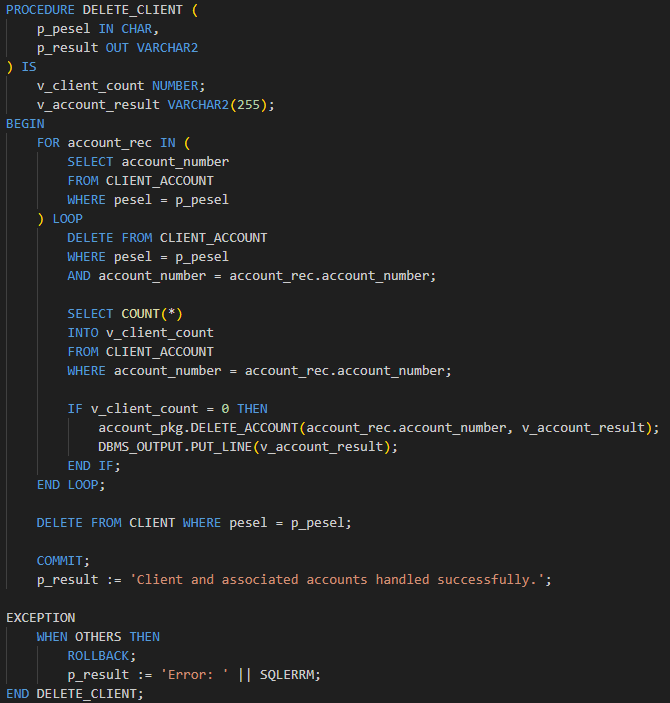
#### UPDATE\_MIDDLE/LAST\_NAME oraz UPDATE\_PHONE\_NUMBER



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



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

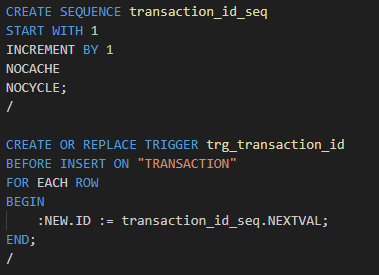
CREATE\_RECEIVER służy do tworzenia nowego odbiorcy. Przyjmuje numer konta odbiorcy, numer konta powiązanego, opis, imię i nazwisko odbiorcy. Metoda sprawdza poprawność danych wejściowych, takich jak długość numeru konta, format cyfr oraz poprawność imion i nazwisk. Jeśli dane są prawidłowe, procedura 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.

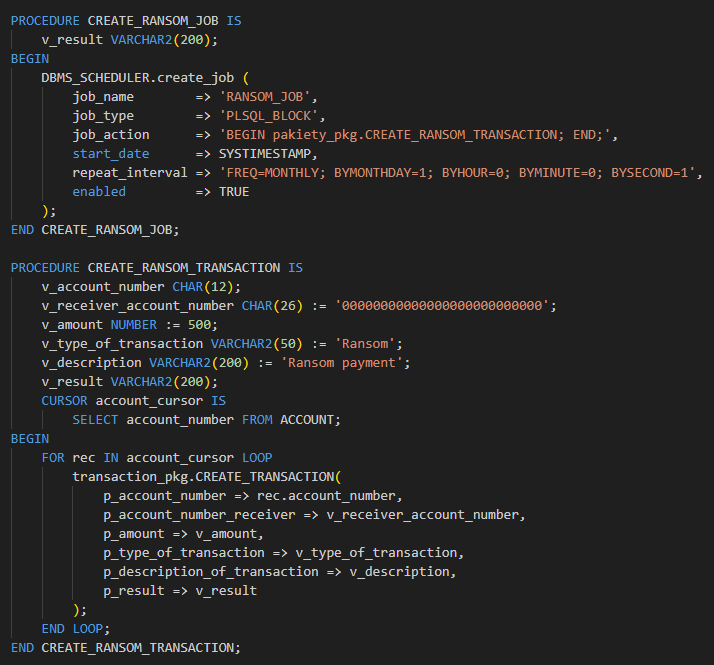
READ\_RECEIVER\_BY\_TIED\_ACCOUNT\_FUNC to funkcja, która umożliwia wyszukiwanie odbiorców na podstawie numeru konta powiązanego. Funkcja ta zwraca listę odbiorców, których numer konta powiązanego odpowiada podanemu w parametrze. Zawiera także walidację danych, umożliwiającą efektywne przetwarzanie wyników.

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.

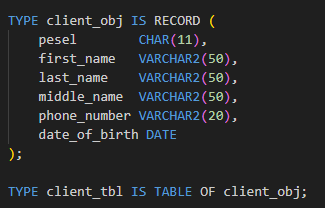
## **Inne ważne komponenty kodu PL/SQL**

Wyzwalacze oraz operacje cykliczne

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.

 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



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**

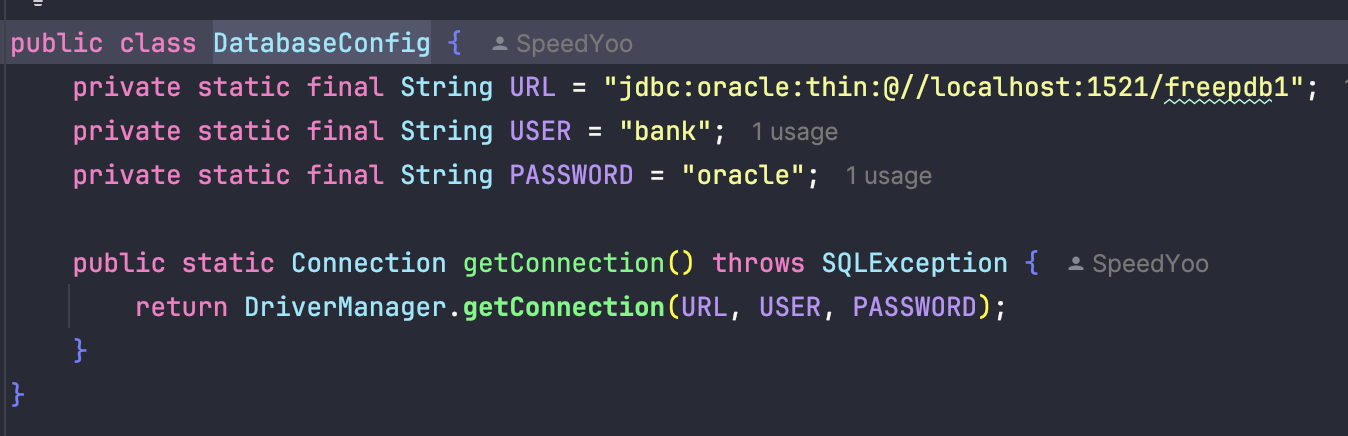
Dane są zarządzane przy pomocy procedur oraz funkcji w języku PL/SQL. Są one uporządkowane w pakiety.

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

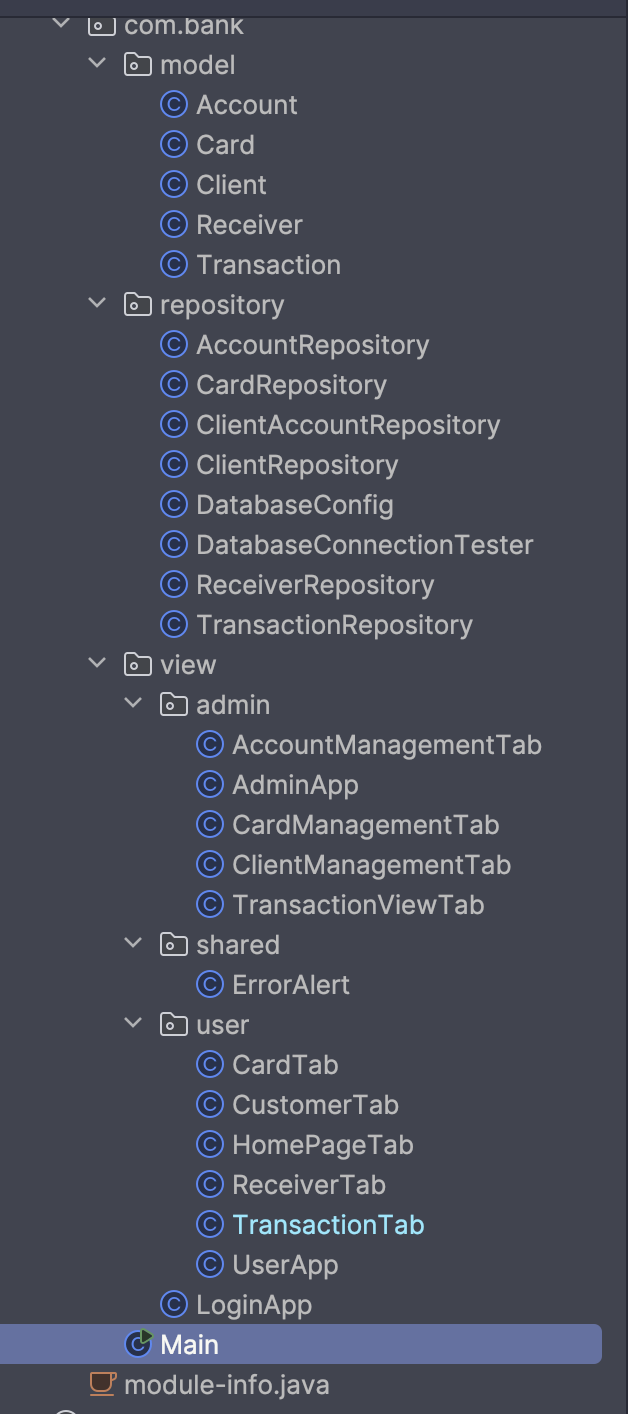
## Struktura

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.

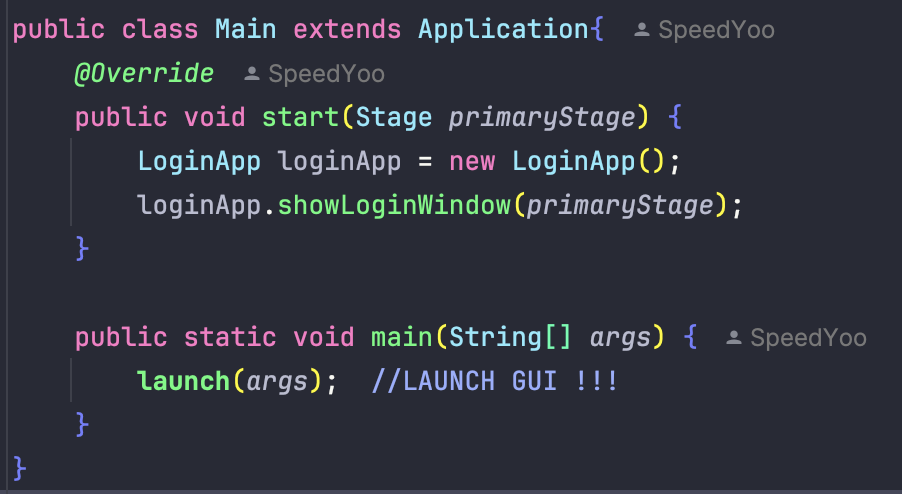
Klasy pakietu View odpowiedzialne są za interfejs graficzny aplikacji. Okno logowania z klasy LoginApp tworzy proste okno do zalogowania się do bazy danych które rówież informuje użytkownika o tym czy udało się połączenie z bazą danych. Aplikacja została podzielona na dwie główne częsci na interfejs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



Struktura projektu

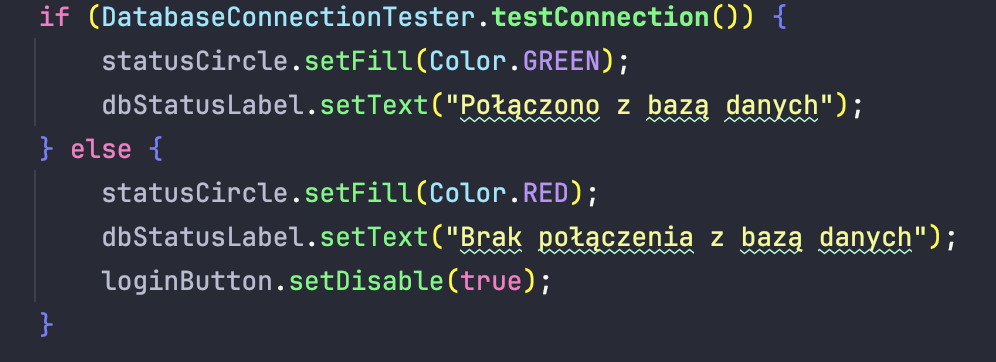
# Opis działania aplikacji

Aplikacja uruchamia się z klasy Main w pakiecie com.bank

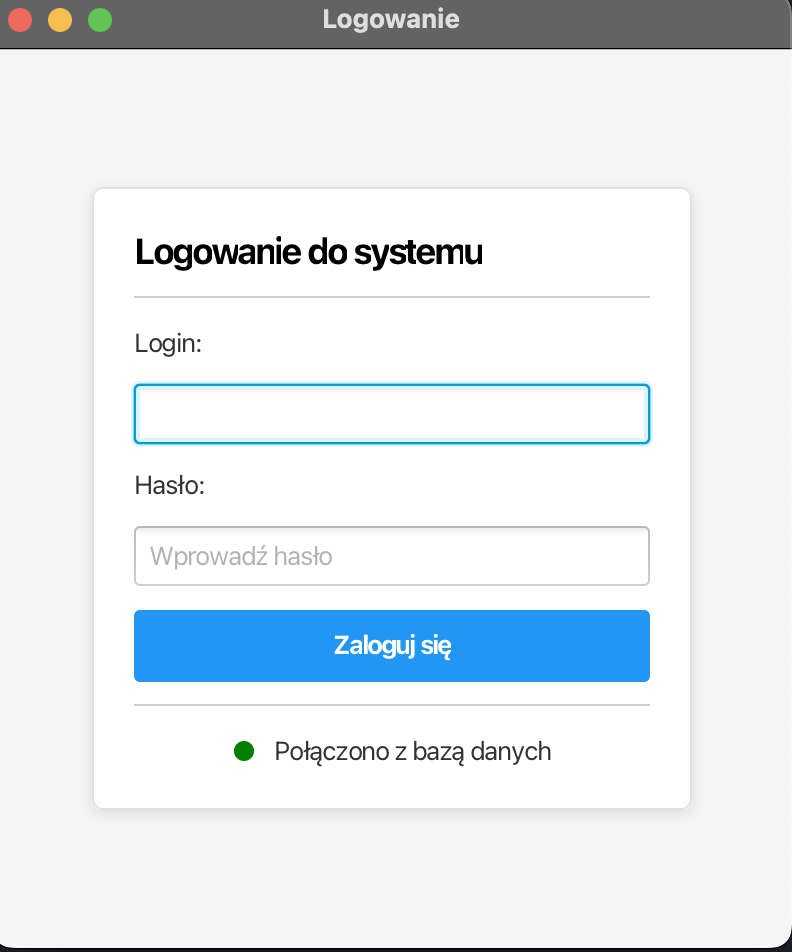


Uruchomienie metody Main wywołuje utworzenie okna logowania

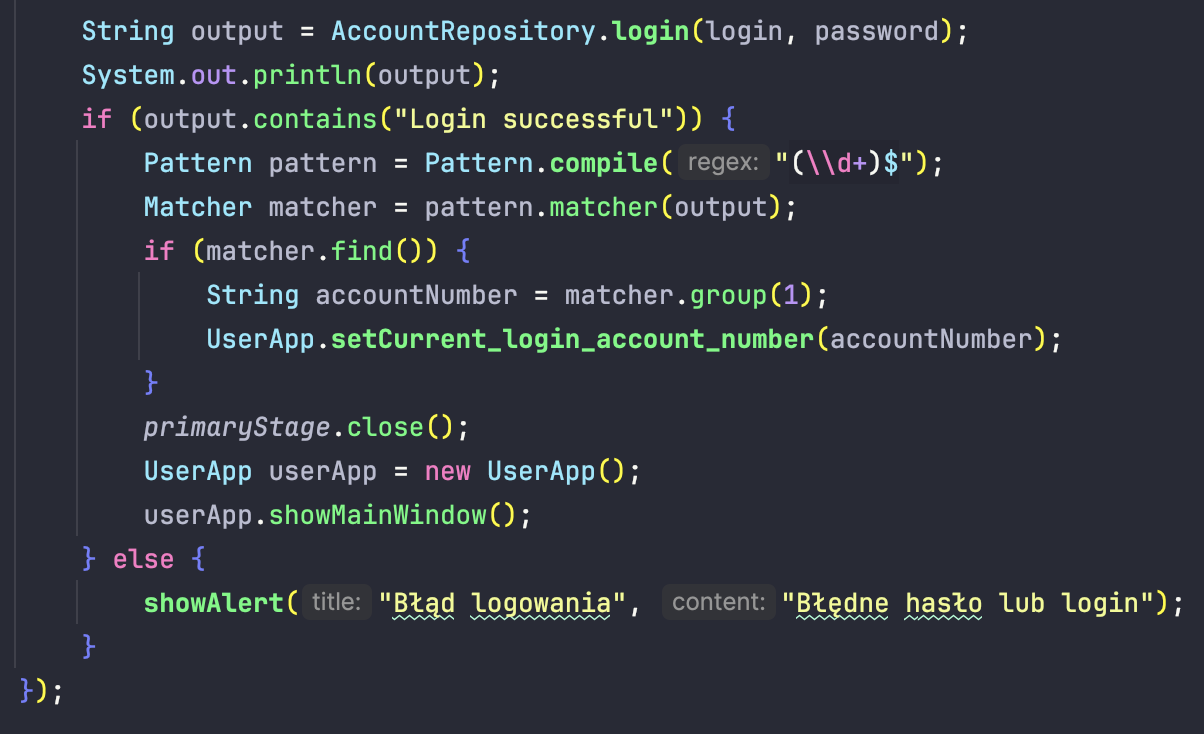
Klasa LoginApp sprawdza, czy udało się połączyć z bazą danych



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

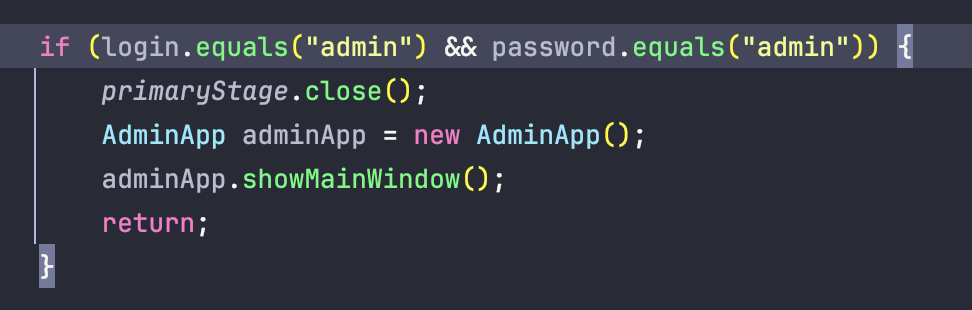


Tak wygląda okno logowania do aplikacji, jeśli połączenie zostało nawiązane



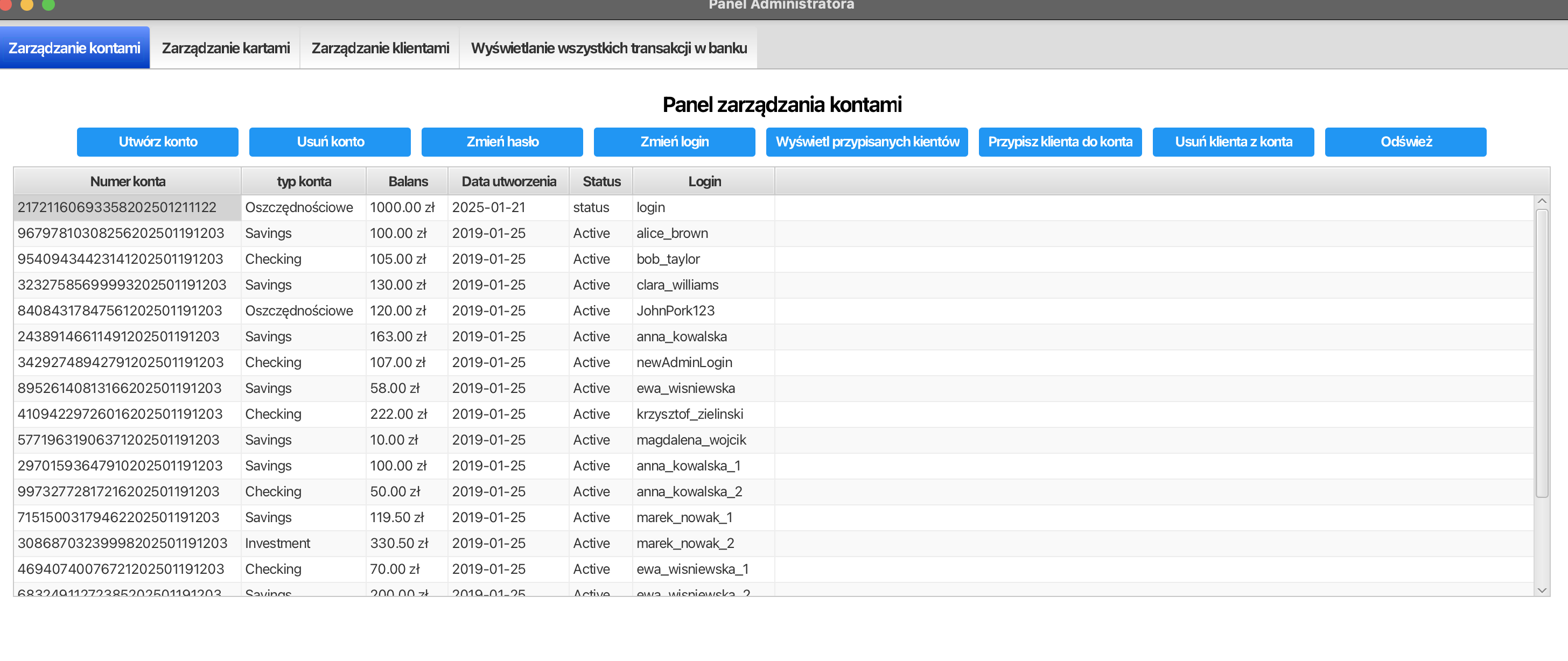
Aplikacja sprawdza czy dane logowania są prawidłowe za pomocą metody z klasy AccountRepository, metoda login z tej klasy wywołuje procedurę do logowania się i jeśli dane są poprawne zwraca numer konta użytkownika. Jeśli proces logowania użytkownika przebiegł pomyślnie ukazuje się okno aplikacji banku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

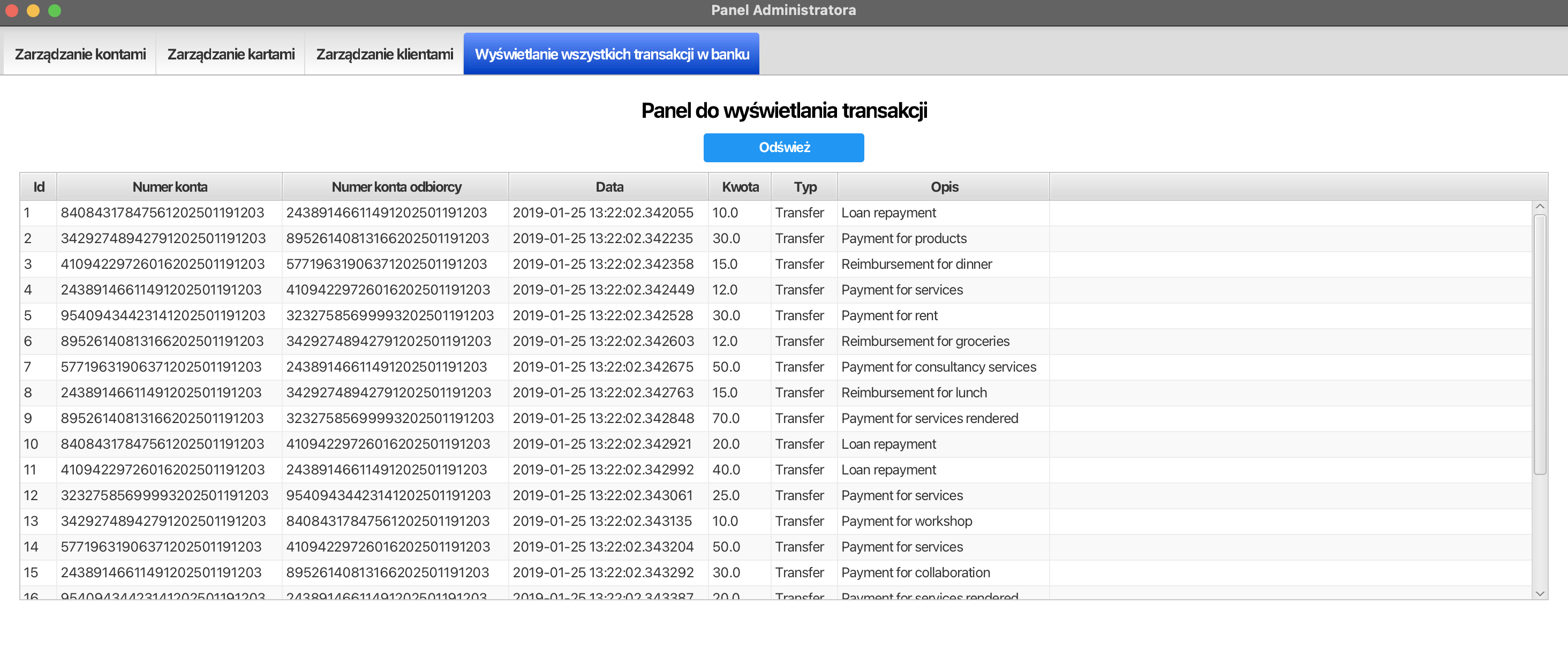
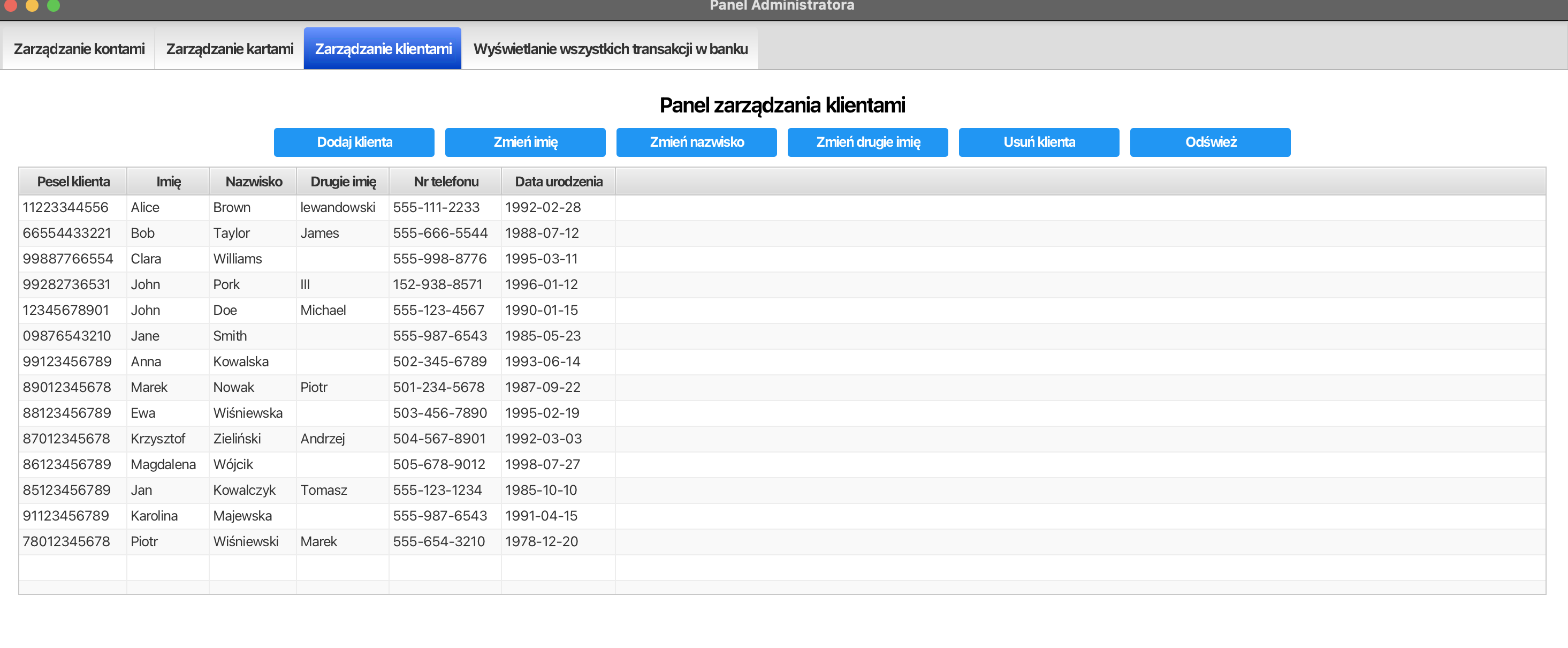
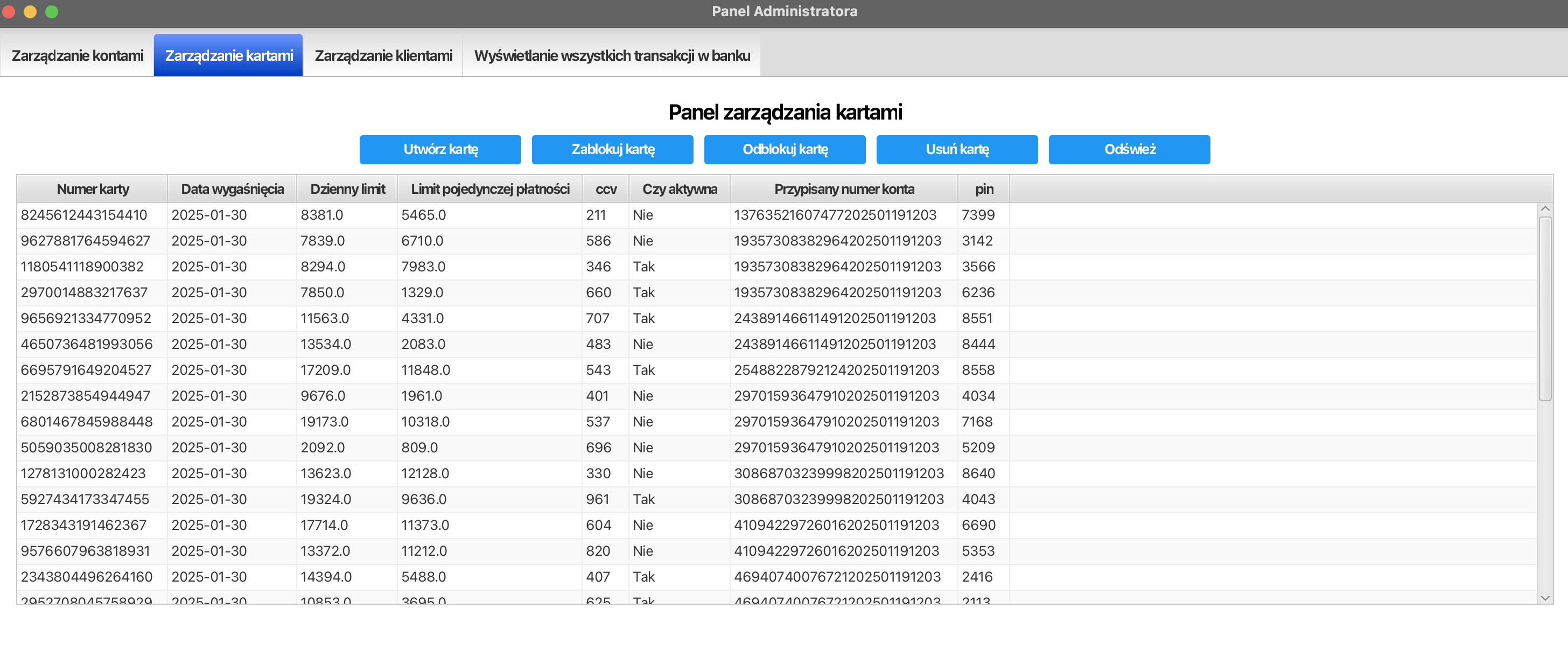
Aplikacja również pozwala zalogować się jako administrator



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



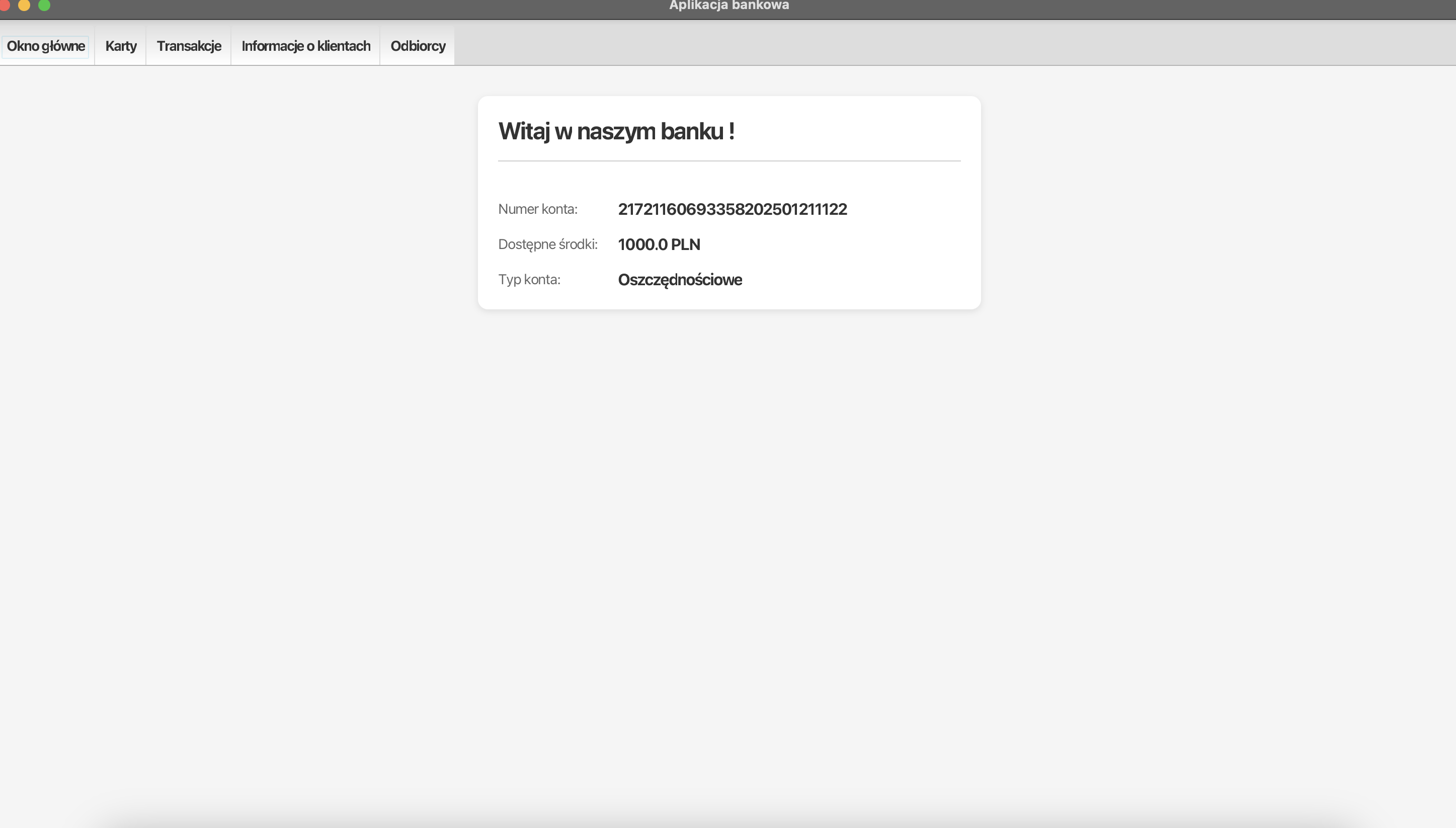


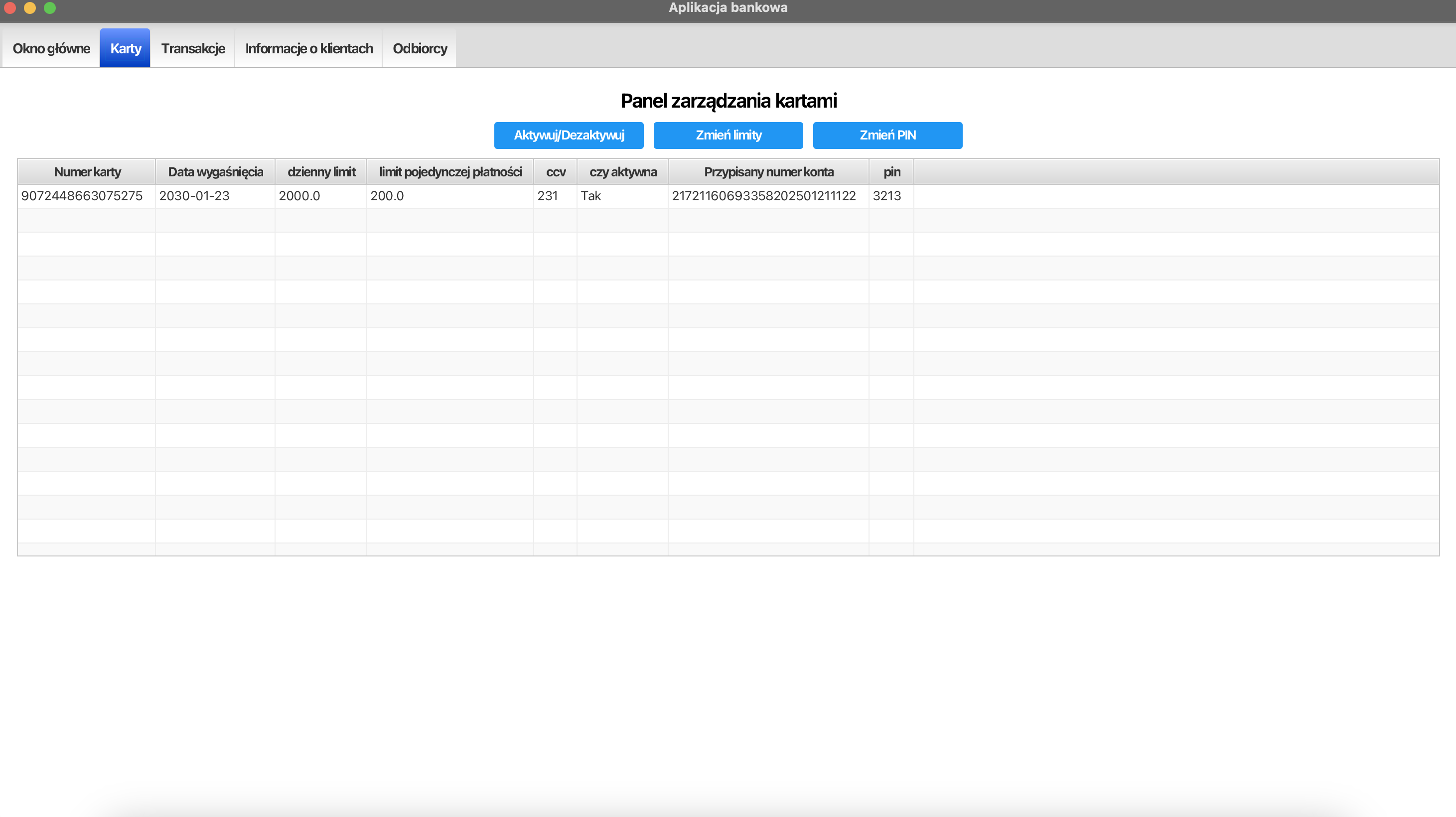
Widok administratora w aplikacji został podzielony na zakładki. Każda zakładka została podzielona na osobną klasę

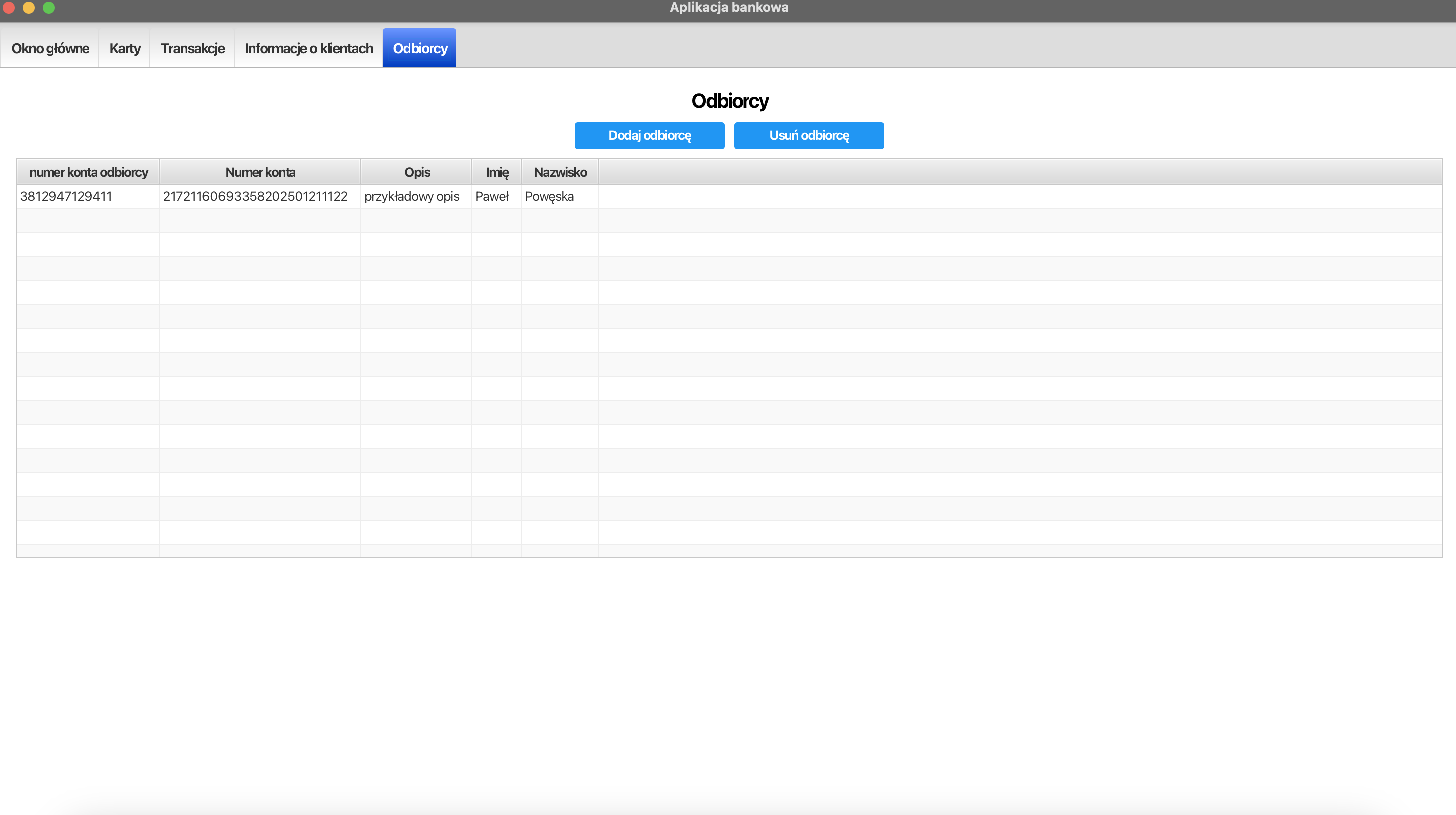
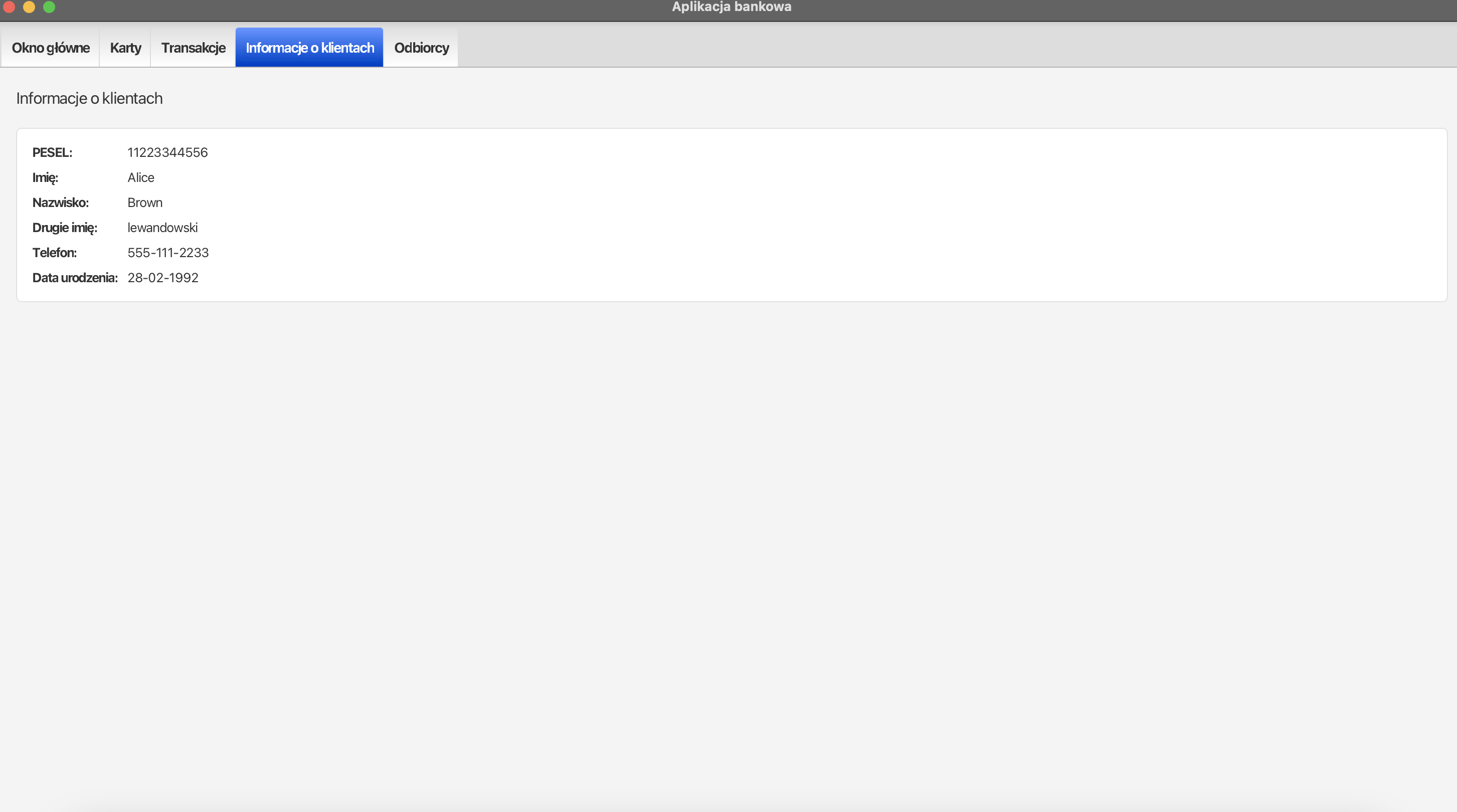
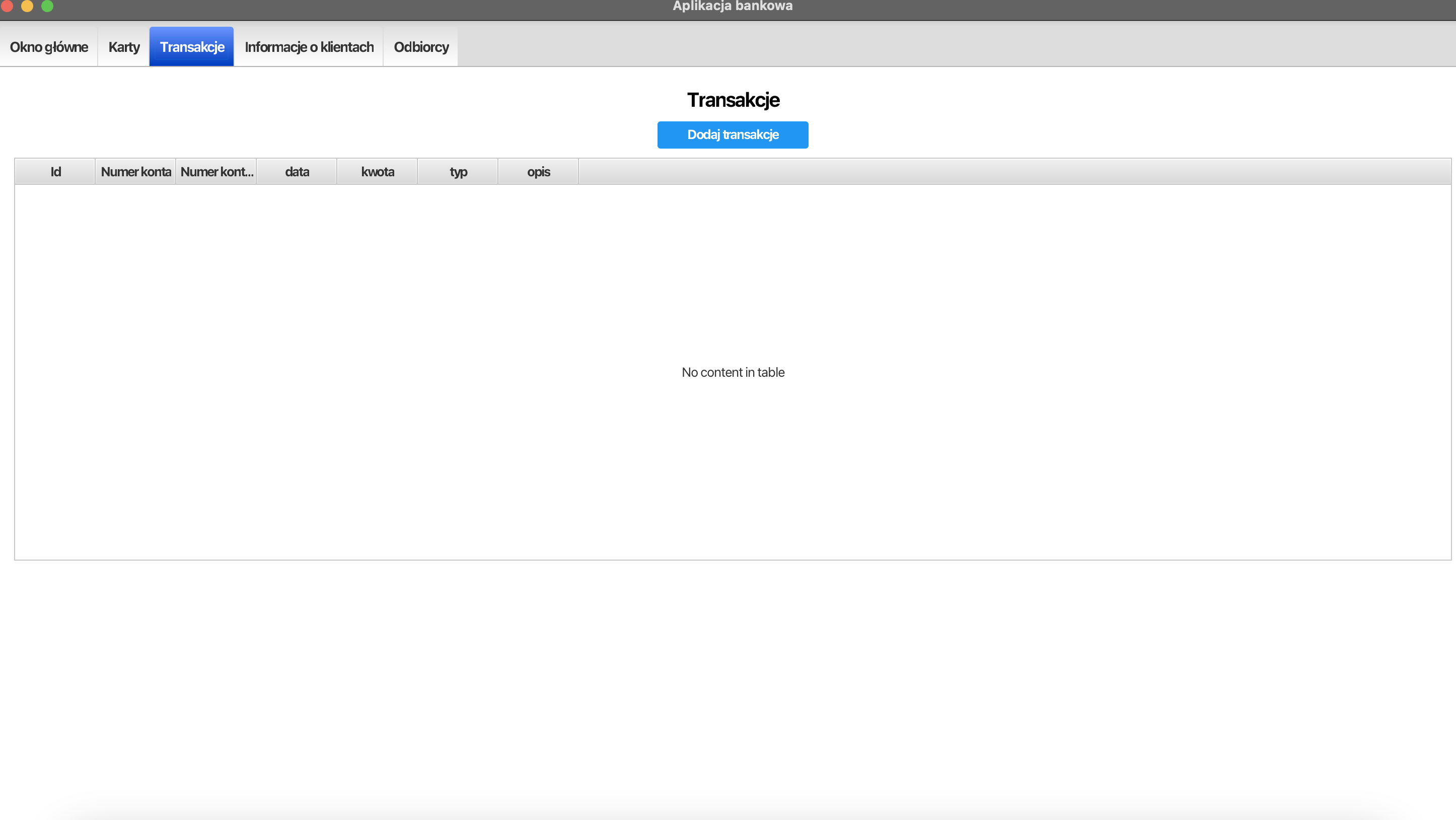
Każda klasa z zakładką wyświetla dane w postaci tabelki oraz przyciski, za pomocą których wywoływane są odpowiednie metody z klas repository, które wywołują procedury lub funkcje w bazie danych

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

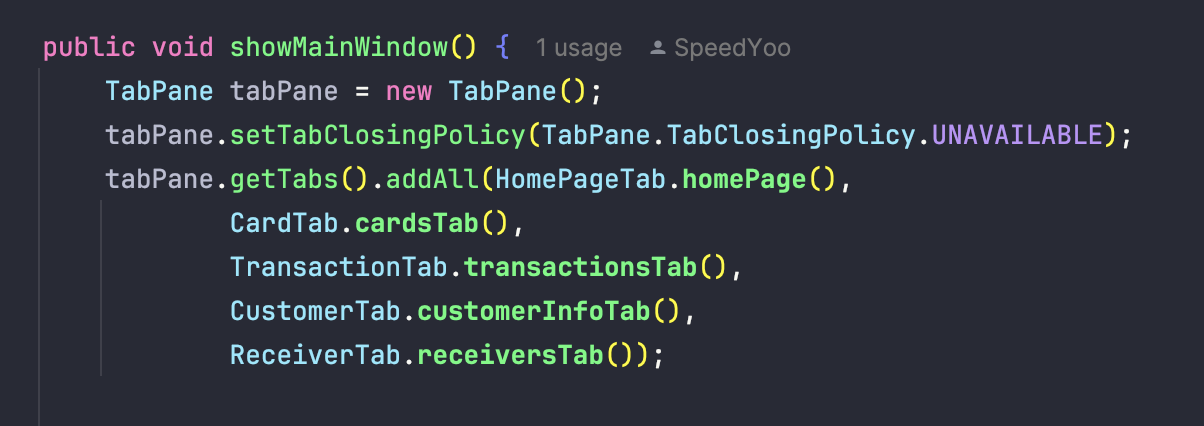
Widok dla klienta







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



# **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

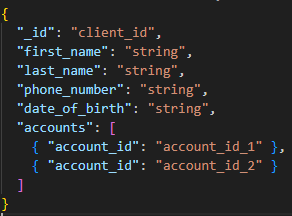
# **Założenia migracja na nierelacyjną bazę danych**

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.

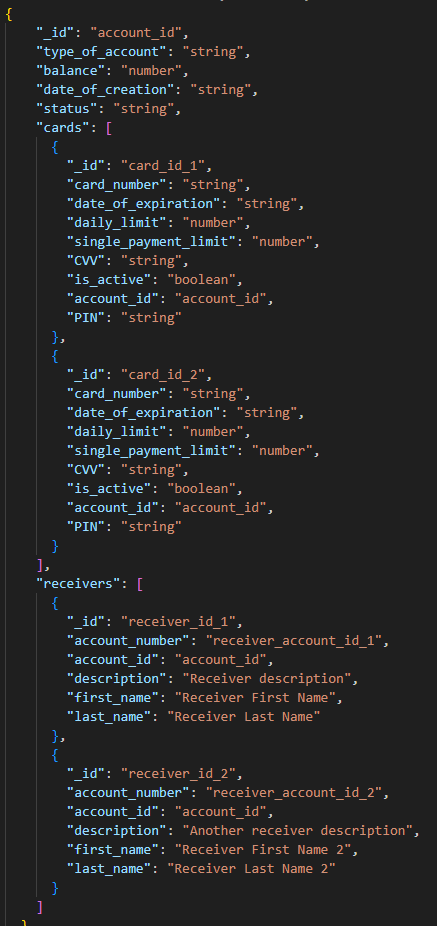
W wyniku migracji tabela client\_account występująca w SQL zostaje porzucona. Dzięki elastyczności nierelacyjnych baz danych nie jest ona konieczna do stworzenia relacji wiele do wielu pomiędzy tabelą account oraz client.

* Client:



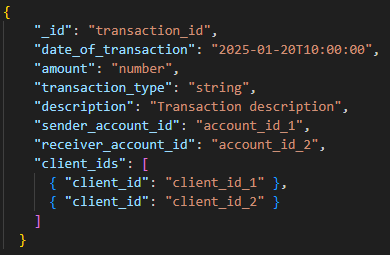
Klient w MongoDB jest dokumentem, który zawiera pole accounts będące tablicą, w której przechowywane są referencje do powiązanych kont. Dzięki tej strukturze, łatwo jest uzyskać dostęp do wszystkich kont powiązanych z danym klientem.

* Account:



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/>