

POLITECHNIKA WROCŁAWSKA

BAZY DANYCH 2

Temat: System obsługujący konta bankowe

Autor:

Sebastian SZKOŁUDA 226166

Radosław AUGUŚCIK 226070

Prowadzący:

dr inż. Roman PTAK

18 listopada 2017

Spis treści

1	Wstęp	2
1.1	Cel projektu	2
1.2	Zakres projektu	2
2	Analiza wymagań	2
2.1	Opis działania i schemat logiczny systemu	2
2.2	Wymagania funkcjonalne	3
2.3	Wymagania niefunkcjonalne	3
2.3.1	Wykorzystywane technologie i narzędzia	3
2.3.2	Wymagania dotyczące rozmiaru bazy danych	3
2.3.3	Wymagania dotyczące bezpieczeństwa systemu	4
3	Projekt systemu	5
3.1	Projekt bazy danych	5
3.1.1	Analiza rzeczywistości i uproszczony model konceptualny	5
3.1.2	Model logiczny i normalizacja	5
3.1.3	Model fizyczny i ograniczenia integralności danych	6
3.1.4	Inne elementy schematu – mechanizmy przetwarzania danych	7
3.1.5	Projekt mechanizmów bezpieczeństwa na poziomie bazy danych	8
3.2	Projekt aplikacji użytkownika	9
3.2.1	Architektura aplikacji i diagramy projektowe	9
3.2.2	Interfejs graficzny i struktura menu	10
3.2.3	Projekt wybranych funkcji systemu	12
3.2.4	Metoda podłączenia do bazy danych - integracja z bazą danych	12
3.2.5	Projekt zabezpieczeń na poziomie aplikacji	12
4	Implementacja systemu baz danych	12
4.1	Tworzenie tabel i definiowanie ograniczeń	13
4.2	Implementacja mechanizmów przetwarzania danych	16
4.3	Implementacja uprawnień i innych zabezpieczeń	17
4.4	Testowanie bazy danych na przykładowych danych	19
5	Implementacja i testy aplikacji	25
5.1	Instalacja i konfigurowanie systemu	25
5.2	Instrukcja użytkownika aplikacji	25
5.3	Testowanie opracowanych funkcji systemu	28
5.4	Omówienie wybranych rozwiązań programistycznych	30
5.4.1	Implementacja interfejsu dostępu do bazy danych	30
6	Podsumowanie	39
7	Literatura	39

1 Wstęp

1.1 Cel projektu

Celem naszego projektu jest utworzenie optymalnego systemu zarządzania kontami bankowymi dla małego, lokalnego banku, posiadającego ograniczoną liczbę kont bankowych.

1.2 Zakres projektu

- Stworzenie bazy danych przechowującej dane klientów oraz dane klientów
- Zaimplementowanie systemu obsługi kont bankowych, który będzie połączony ze wcześniej stworzoną bazą danych
- Optymalizacja całego systemu, aby zmniejszyć jego awaryjność a zarazem zwiększyć szybkość
- Testowanie systemu w celu wyeliminowania jakichkolwiek błędów i przeoczeń

2 Analiza wymagań

2.1 Opis działania i schemat logiczny systemu

Nasz projekt ma obejmować utworzenie spójnego systemu, mającego na celu obsługę kont bankowych i innych usług tego banku oraz połączenie go z bazą danych, która ułatwi zarządzanie. Z założenia ma to być mały lokalny bank. Określenie tego aspektu jest ważne z tego względu iż musimy wiedzieć jak dużą bazę utworzyć, z jakiego oprogramowania skorzystać aby było to jak najbardziej optymalne rozwiązanie oraz jakie funkcje przygotować, aby obsługiwały różnego rodzaju zdarzenia. Kolejnym naszym celem będzie stworzenie funkcji, mających na celu usprawnienie kontaktu pomiędzy klientem, a pracownikiem danej instytucji. Zrobienie tego pomoże w zrealizowaniu celu dotyczącego osiągnięcia maksymalnego pulapu klientów w jak najkrótszym czasie. W naszej opinii jednymi z funkcji, które są konieczne do oprogramowania są:

- Funkcja umożliwiająca kontakt z klientem, poprzez wykorzystanie poczty elektronicznej.
- Transakcje bankowe, które są jednym z najważniejszych aspektów naszego projektu. Mają one na celu umożliwienie klientowi kontroli nad powierzonymi pieniędzmi.
- Wprowadzanie danych klienta, oraz ich edycja. Co pozwoli na jednoznaczną identyfikację klienta. W naszym przypadku będzie to numer PESEL, przypisane do niego imię i nazwisko oraz adres korespondencyjny, jeśli klient zażyczyłby sobie, aby otrzymywać informacje z banku poprzez pocztę, co uwzględnimy w bazie danych.
- Możliwość przeprowadzania wpłat oraz wypłat pieniędzy z konta bankowego.
- Możliwość pobrania kredytu, oczywiście jeśli klient posiada zdolność kredytową.
- Oprogramowanie będzie obsługiwać przelewy, które muszą zostać udokumentowane w odpowiedni sposób.
- Oprogramowanie będzie działać bezawaryjnie (dopuszczamy przerwy trwające do godziny czasu ale nie częściej niż raz w tygodniu).
- Pojawi się możliwość kontaktu z serwisantem w razie problemów z obsługą.
- Z oprogramowania mogą korzystać jedynie przeszkolone osoby, zatrudnione w banku.

2.2 Wymagania funkcjonalne

- Klient, bankier, kierownik muszą przejść proces autoryzacji, żeby móc korzystać z systemu
- Bankier ma możliwość założenia konta bankowego na prośbę klienta
- Bankier ma możliwość wykonanie przelewu na zlecenie klienta
- Bankier ma możliwość udzielenia kredytu klientowi
- Bankier ma możliwość założenia lokaty na prośbę klienta
- Bankier ma możliwość przyjęcia rezygnacji z konta od klienta i usunięcia konta z systemu
- Kierownik może wykonywać te same czynności co bankier (rozszerza jego funkcjonalność)
- Kierownik ma możliwość przeglądania, modyfikacji danych klientów
- Kierownik ma możliwość wysyłania informacji handlowych oraz wiadomości do klientów
- Każdy klient ma możliwość skorzystania z bankomatu po autoryzacji kodem PIN i kartą kredytową

W bankomacie:

- Klient ma możliwość wypłaty pieniędzy
- Klient ma możliwość wydrukowania potwierdzenia z wypłaty pieniędzy
- Klient ma możliwość sprawdzenia stanu konta

W aplikacji desktopowej:

- Klient ma możliwość wykonania przelewu
- Klient ma możliwość sprawdzenia informacji o koncie (w tym stanu konta, aktywnych lokat i kredytów)
- Klient ma możliwość odebrania informacji handlowej wysłanej przez kierownika
- Klient ma możliwość sprawdzenia historii transakcji

2.3 Wymagania нефункциональные

2.3.1 Wykorzystywane technologie i narzędzia

- Językiem, którego będziemy używali do zaimplementowania systemu będzie Java.
- Programem do stworzenia bazy danych będzie MySQL

2.3.2 Wymagania dotyczące rozmiaru bazy danych

Zakłada się, że :

- Bank będzie zatrudniał około 25 bankierów i 5 kierowników
- Bank będzie posiadał zarejestrowanych około 6000 kont bankowych
- Bank umożliwia klientowi korzystanie z jego usług na trzy sposoby:
 - Poprzez aplikację desktopową
 - Poprzez bankomat stanowiący element systemu

– Bezpośrednio w placówce banku

- Bank będzie rejestrował średnio 2 nowe konta bankowe dziennie
- Bank będzie uznawał średnio 1 rezygnację z konta dziennie
- Klienci będą wpłacać pieniądze średnio 400 razy dziennie
- Klienci będą wypłacać pieniądze około 1000 razy dziennie
- Klienci będą przelewać pieniądze (w tym realizować płatności kartą) około 3000 razy dziennie

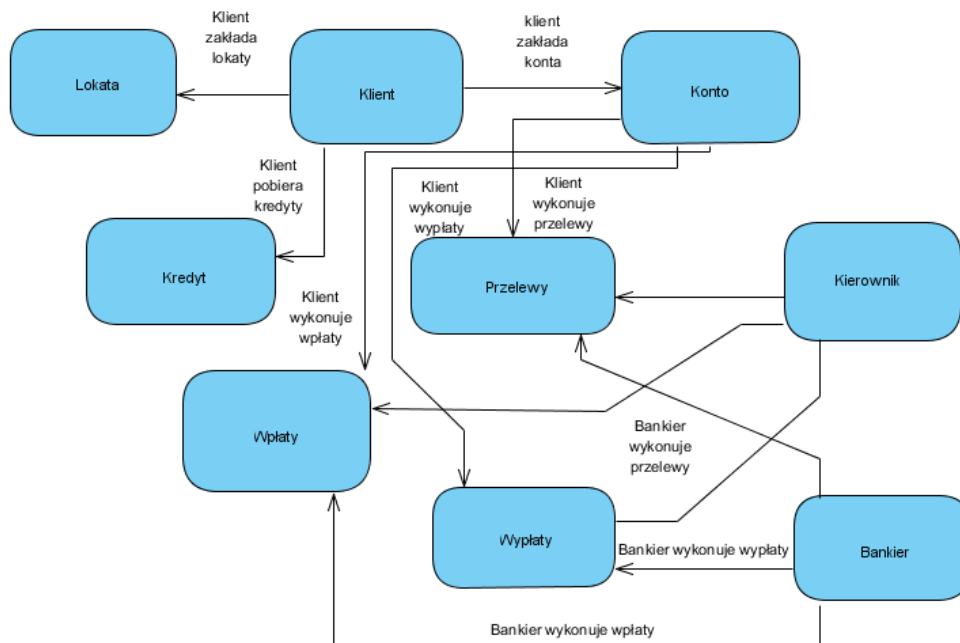
2.3.3 Wymagania dotyczące bezpieczeństwa systemu

- Tworzenie kont może się odbywać tylko poprzez uprawnionych do tego bankierów
- Transakcja, która dotyczy przelewu na kwotę większą niż 50.000 PLN wymaga zatwierdzenia przez kierownika, który sprawdza jej poprawność
- Każdy klient otrzyma swój unikatowy numer ID i ustali hasło, którymi będzie logował się do systemu
- Każdy pracownik jak i klient loguje się na własne konto z przyznanymi uprawnieniami

3 Projekt systemu

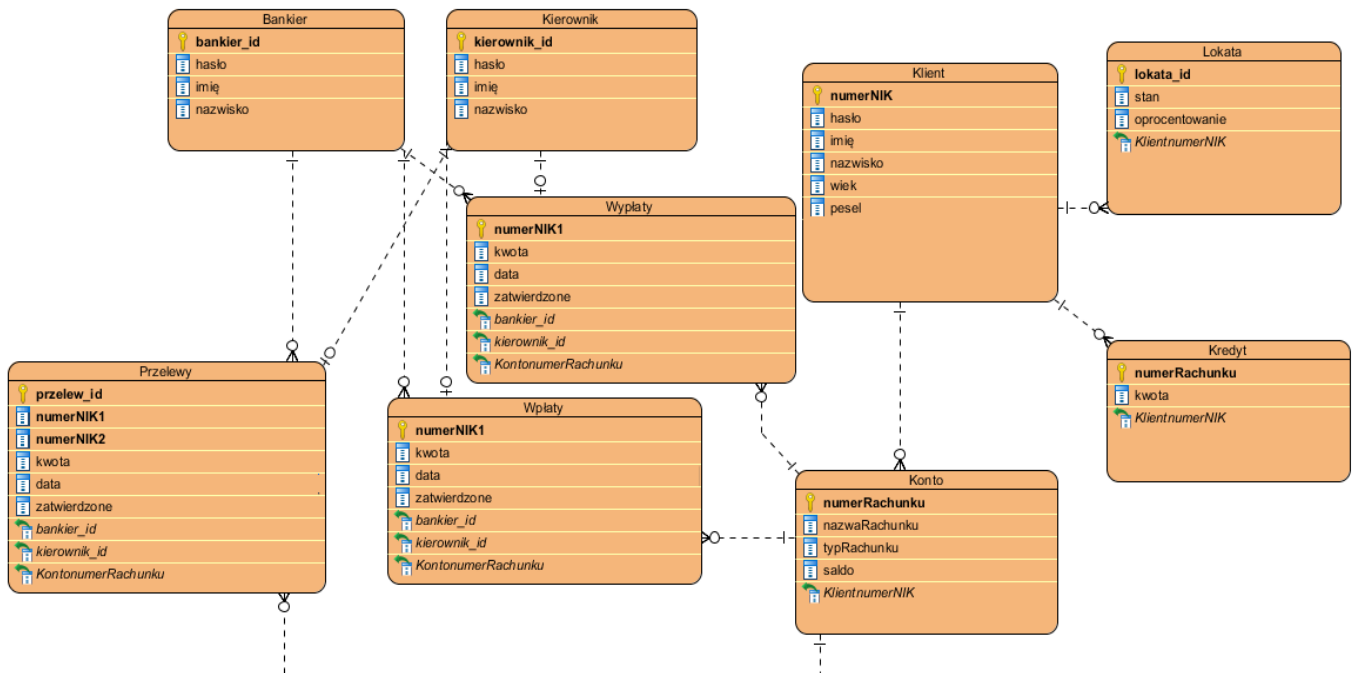
3.1 Projekt bazy danych

3.1.1 Analiza rzeczywistości i uproszczony model konceptualny



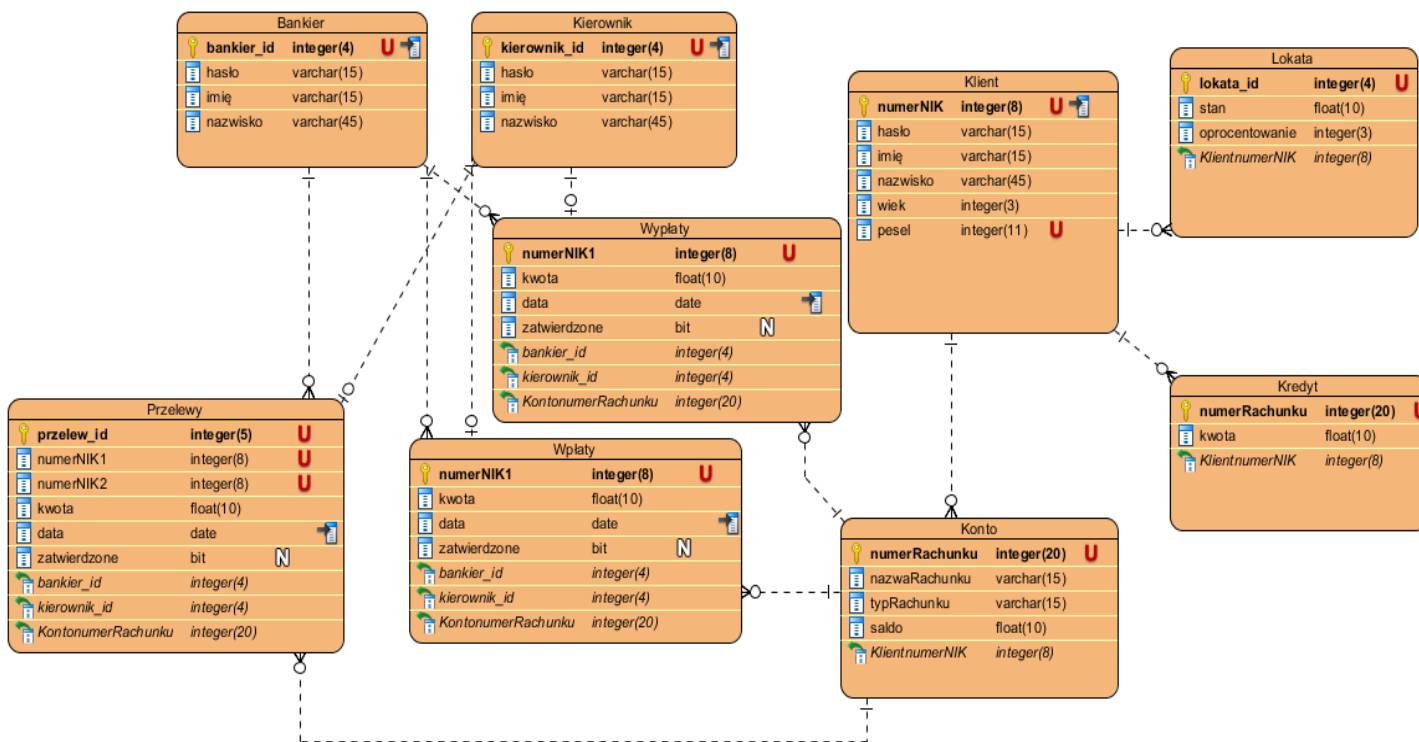
Rysunek 1: Uproszczony model bazy danych

3.1.2 Model logiczny i normalizacja



Rysunek 2: Logiczny model bazy danych

3.1.3 Model fizyczny i ograniczenia integralności danych



Rysunek 3: Fizyczny model bazy danych

3.1.4 Inne elementy schematu – mechanizmy przetwarzania danych

a) Indeksy

W naszej bazie danych indeksy są stworzone w celu przyspieszenia wyszukiwania pracowników, klientów czy też samych transakcji. Indeksy wspomagają operacje INSERT, DELETE, UPDATE, Są ważne w przypadku sortowania.

- W tabelach Kierownik, Bankier, Klient indeks został nałożony na bankier_id, kierownik_id oraz numerNIK, w celu przyspieszenia procesu logowania przy szukaniu danego użytkownika i przyznaniu mu odpowiedniej autoryzacji.
- W tabelach przelewy, wpłaty i wypłaty indeks został nałożony na date z powodu częstego wyszukiwania transakcji według daty. W późniejszym etapie funkcjonowania aplikacji będzie to konieczne jeśli będziemy chcieli wyświetlić wszystkie transakcje z danego dnia lub posortować transakcje według daty.

b) Widoki

Widoki tworzone będą na poziomie aplikacji lecz już teraz możemy zdefiniować co będzie niezbędne

- Widok, który umożliwia klientom banku uzyskać informacje na temat ogólnych informacji o ich transakcjach.
CREATE VIEW 'informacje_ogolne' AS
SELECT COUNT(Wypłaty.k_NumerNIK+Wpłaty.k_numerNIK+Przelewy.k_numerNIK) –Liczba wykonanych transakcji
SUM(Wypłaty.kwota) –Suma wypłat
SUM(Wpłaty.kwota) –Ilość wpłacanych pieniędzy
COUNT(Przelewy.k_numerNIK) –Ilość wykonanych przelewów
FROM Klient;
- Widok, który umożliwia najważniejszym pracownikom banku uzyskać informacje na temat ich klientów.
CREATE VIEW 'informacje_o_klientach' AS
SELECT klient_id, login, hasło, imię, nazwisko, wiek,
numer_NIK, numerRachunku,
COUNT(numerNIK) –Ilość klientów banku
FROM Klient;
- Widok, który umożliwia uzyskanie informacji o własnym koncie:

```
DROP VIEW IF EXISTS `bank`.`informacje_o_koncie`;
DROP TABLE IF EXISTS `bank`.`informacje_o_koncie`;
USE `bank`;
CREATE OR REPLACE VIEW `informacje_o_koncie` AS
SELECT
    kl.numerNIK 'numerNIK', kl.haslo 'haslo', kl.imie 'imie', kl.nazwisko 'nazwisko',
    kl.wiek 'wiek', k.numerRachunku 'numerRachunku',
    k.nazwaRachunku 'nazwaRachunku', k.typRachunku 'typRachunku'
FROM
    Klient kl
JOIN
    Konto k ON kl.numerNIK = k.numerNIK;
```

- Widok, który umożliwia uzyskanie informacji na temat poszczególnych transakcji wykonanych przez klienta
CREATE VIEW 'informacje_o_transakcji' AS
SELECT transakcja_id, kwota, data


```
WHERE (transakcja_id=Wypłaty.transakcja_id) OR
(transakcja_id=Wpłaty.transakcja_id) OR (transakcja_id=Przelewy.transakcja_id)
FROM Transakcje;
```

c) Triggery (Wyzwalacze)

Wyzwalacze mogą ograniczać dostęp do pewnych danych, rejestrować zmiany danych lub nadzorować modyfikacje danych. W MySQL obsługiwane są wyzwalacze dla instrukcji INSERT, UPDATE i DELETE, co w znacznym stopniu nam pomoże, ponieważ jako bank ciągle będziemy dodawać, usuwać lub modyfikować dane klientów.

- CREATE TRIGGER nowy klient trigger
BEFORE UPDATE ON klienci
REFERENCING NEW ROW AS n, OLD ROW AS o
FOR EACH ROW
IF klient id = "1000" THEN
–wykonaj odpowiednie działania;
END IF;

3.1.5 Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

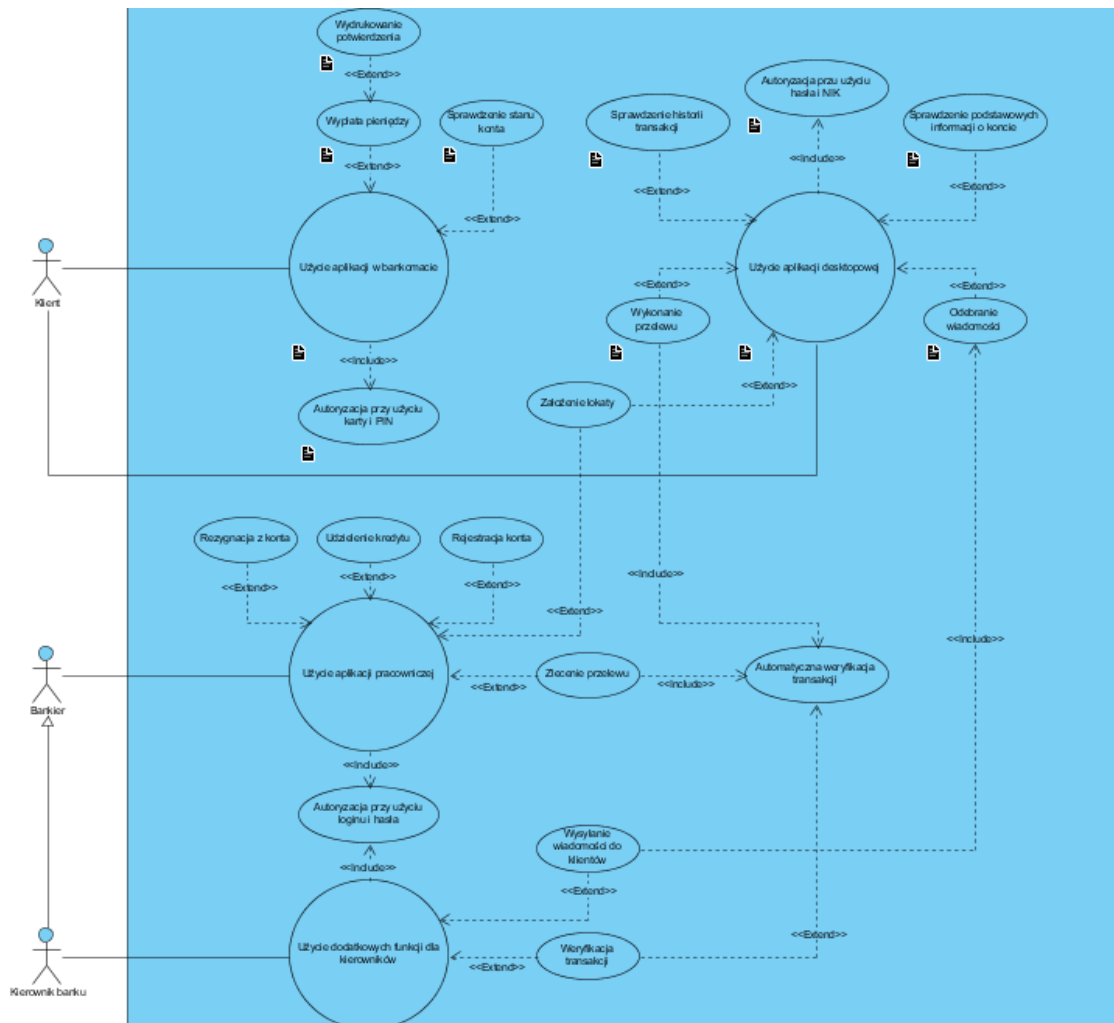
Rola/Tabela	Wpłaty	Wypłaty	Przelewy	Kredyt	Lokata
Kierownik	ALL	ALL	ALL	ALL	ALL
Bankier	INSERT/	INSERT/	INSERT/	INSERT/	INSERT/
	SELECT/	SELECT/	SELECT/	SELECT/	SELECT/
	UPDATE/	UPDATE/	UPDATE/	UPDATE/	UPDATE/
Klient				DELETE	DELETE
	INSERT/	INSERT/	INSERT/	SELECT	SELECT/
	SELECT	SELECT	SELECT		UPDATE

Rysunek 4: Przyznane uprawnienia

- Dostęp do bazy danych posiadać będzie w znacznej części kierownik oraz bankier.
- Aby mieć możliwość przeglądania bazy danych należy podać numer identyfikacyjny i hasło.
- Tylko kierownik będzie posiadać pełny dostęp do bazy danych, wszystkie funkcje będą udostępnione tylko jemu.
- Kierownik będzie musiał zmieniać hasło co 30 dni w celu uniknięcia złamania hasła co mogłoby doprowadzić do próby kradzieży w banku.

3.2 Projekt aplikacji użytkownika

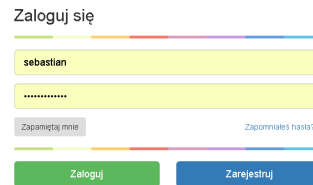
3.2.1 Architektura aplikacji i diagramy projektowe



Rysunek 5: Diagram przypadków użycia

3.2.2 Interfejs graficzny i struktura menu

Poniższe obrazy pokazują proponowany wygląd naszej aplikacji, który oczywiście może ulec zmianie w trakcie przebiegu projektu. Został stworzony w środowisku Java, za pomocą narzędzia programistycznego Eclipse Neon for Java EE Developers. Backend został stworzony za pomocą narzędzia Spring i Hibernate, natomiast frontend przy pomocy jQuery Bootstrap. Całość została zaimplementowana do html, jako, że ma to być docelowo aplikacja webowa.



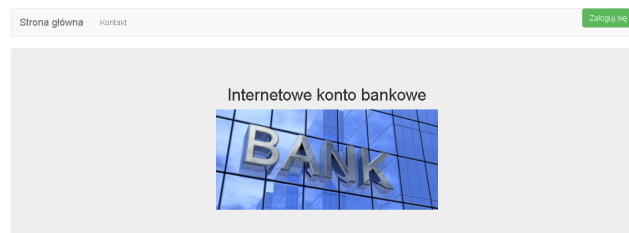
Zaloguj się

sebastian

[Zapomniałem hasła?](#)

[Zaloguj](#) [Zarejestruj](#)

Rysunek 6: Panel logowania



Rysunek 7: Strona główna

Zarejestruj się

Imię
 Nazwisko

Login

Adres Email

Hasło
 Potwierdź Hasło

☐ Zgadzasz się


Rysunek 8: Rejestracja

[Strona główna](#)
[Kontakt](#)

Konto ▾

Lista klientów
 Edycja danych
 Lista transakcji

Włóż



Lista klientów

Id	Login	Hasło	Imię	Nazwisko	Numer konta
1	radek	42c92ca8cb2694f1e6d6bf0d8c33c53	Radek	Augustyk	2147483647

localhost:8080/klient#

Rysunek 9: Menu

Skontaktuj się z nami

Imię

 Wprowadź imię

Adres Email

 Wprowadź Email

Temat

 Wybierz

Wiadomość

Wiadomość

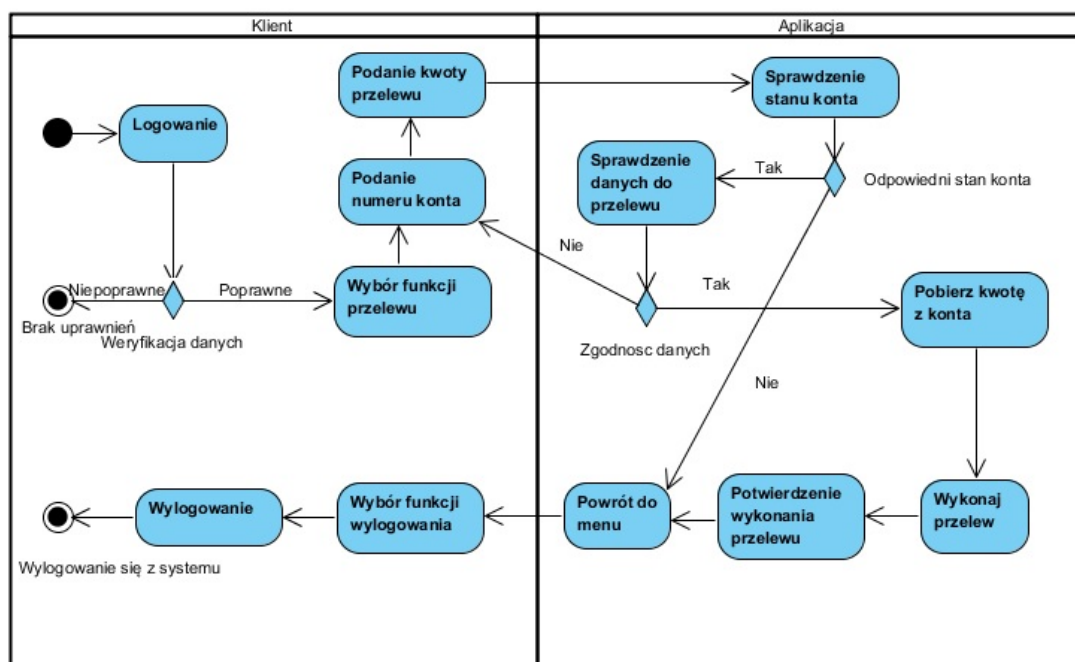
Wyślij wiadomość



Lokalizacja:
 BEOZIE TU COS
 NO I TU TEZ
 ☎ (+48) 756-235-981
 Nasz Adres Email
 first.last@example.com

Rysunek 10: Kontakt

3.2.3 Projekt wybranych funkcji systemu



Rysunek 11: Diagram aktywności wykonywania przelewu

3.2.4 Metoda połączenia do bazy danych - integracja z bazą danych

Baza danych, z której będzie korzystała aplikacja znajduje się na serwerze Java Database Server postawiony przy pomocy Wampserver. Bezpośredni dostęp do bazy można uzyskać poprzez MySQL. Integracja aplikacji z bazą danych zrealizowana będzie za pomocą JDBC, czyli Java DataBase Connectivity. Biblioteki, które są potrzebne, zawierają pakiety `java.sql.*` i są w taki sam sposób dostępne jak kolekcje w pakietach `java.util.*`.

3.2.5 Projekt zabezpieczeń na poziomie aplikacji

- Dostęp do aplikacji zabezpieczony jest za pomocą panelu logowania. Do uzyskania dostępu do zawartości wymagane jest podanie NIK oraz hasła. Jeśli pięć razy pod rząd wpisujemy złe hasło, aplikacja zablokuje dostęp do konta. Wtedy konieczne jest skontaktowanie się z kierownikiem, który będzie miał możliwość odblokowania dostępu.
- Brak aktywności w aplikacji przez 10 minut, będzie wymagał ponownego zalogowania w celu przywrócenia sesji.

4 Implementacja systemu baz danych

Do zamodelowania bazy danych wykorzystano program MySQL Workbench. Funkcjonalność programu pozwala na tzw. „Forward Engineering”, czyli na wygenerowanie potrzebnych skryptów w SQL do wykonania w rzeczywistej bazie danych. Funkcja gwarantuje utworzenie komend obejmujących cały model bazy – od tabel po ręcznie dodanych użytkowników i rekordy. Znacznie przyspiesza do implementację systemu i gwarantuje prawidłowe wykonanie skryptów tworzących.

Serwer bazy danych został utworzony za pomocą Wampserver 3.1.0. Po korektach w przygotowanych widokach, zamodelowana baza została w pełni załadowana na serwer.

4.1 Tworzenie tabel i definiowanie ograniczeń

- Utworzenie schematu:

```
CREATE SCHEMA If NOT EXISTS 'bank' DEFAULT CHARACTER SET 'utf8';
USE 'bank';
```

- Utworzenie tabel razem z indeksami (wszystkie tabele zostały wygenerowane za pomocą MySQL Workbench):

```
CREATE TABLE `bank`.`Konto` ( `numerRachunku` INT(20) NOT NULL ,
`nazwaRachunku` VARCHAR(15) NOT NULL ,
`typRachunku` VARCHAR(15) NOT NULL ,
`saldo` FLOAT(10) NOT NULL ,
`k_numerNIK` INT(8) NULL ,
UNIQUE (`numerRachunku`),
PRIMARY KEY(`numerRachunku`),
CONSTRAINT `fk_konta`
FOREIGN KEY (`k_numerNIK`)
REFERENCES `bank`.`Klient` (`numerNIK`)
ON DELETE RESTRICT
ON UPDATE RESTRICT)
ENGINE = InnoDB;
```

Tabela Konto

```
CREATE TABLE `bank`.`Klient` ( `numerNIK` INT(8) NOT NULL ,
`hasło` VARCHAR(15) NOT NULL , `imie` VARCHAR(15) NOT NULL ,
`nazwisko` VARCHAR(45) NOT NULL , `wiek` INT(3) NOT NULL ,
`pesel` INT(11) NOT NULL,
UNIQUE (`pesel`),
PRIMARY KEY (`numerNIK`))
ENGINE = InnoDB;
```

Tabela Klient

```
CREATE TABLE `bank`.`Kierownik` ( `kierownik_id` INT(4) NOT NULL ,
`hasło` VARCHAR(15) NOT NULL , `imie` VARCHAR(15) NOT NULL ,
`nazwisko` VARCHAR(45) NOT NULL , PRIMARY KEY (`kierownik_id`))
ENGINE = InnoDB;
```

Tabela Kierownik

```
CREATE TABLE `bank`.`Bankier` ( `bankier_id` INT(4) NOT NULL ,
`hasło` VARCHAR(15) NOT NULL , `imie` VARCHAR(15) NOT NULL ,
`nazwisko` VARCHAR(45) NOT NULL , PRIMARY KEY (`bankier_id`))
ENGINE = InnoDB;
```

Tabela Bankier

```

CREATE TABLE `bank`.`Wyplaty` ( `numerNIK` INT(8) NOT NULL ,
`kwota` FLOAT(10) NOT NULL ,
`data` DATE NOT NULL ,
`zatwierdzone` BIT NULL ,
`bankier_id` INT(4) NULL ,
`k_numerrachunku` INT(20) NULL ,
`kierownik_id` INT(4) NULL ,
PRIMARY KEY (`numerNIK`),
INDEX (`data`),
CONSTRAINT `fk_wyplaty`
FOREIGN KEY (`bankier_id`)
REFERENCES `bank`.`Bankier`(`bankier_id`)
ON DELETE RESTRICT
ON UPDATE RESTRICT,
CONSTRAINT `fk_zatwierdzanie`
FOREIGN KEY (`kierownik_id`)
REFERENCES `bank`.`Kierownik`(`kierownik_id`)
ON DELETE RESTRICT
ON UPDATE RESTRICT,
CONSTRAINT `fk_wyplaty1`
FOREIGN KEY (`k_numerrachunku`)
REFERENCES `bank`.`Konto`(`numerRachunku`)
ON DELETE RESTRICT
ON UPDATE RESTRICT)
ENGINE = InnoDB;

```

Tabela Wyłaty

```

CREATE TABLE `bank`.`Wplaty` ( `numerNIK` INT(8) NOT NULL ,
`kwota` FLOAT(10) NOT NULL ,
`data` DATE NOT NULL ,
`zatwierdzone` BIT NULL ,
`bankier_id` INT(4) NULL ,
`k_numerrachunku` INT(8) NULL ,
`kierownik_id` INT(4) NULL ,
PRIMARY KEY (`numerNIK`),
INDEX (`data`),
CONSTRAINT `fk_wplaty`
FOREIGN KEY (`bankier_id`)
REFERENCES `bank`.`Bankier`(`bankier_id`)
ON DELETE RESTRICT
ON UPDATE RESTRICT,
CONSTRAINT `fk_zatwierdzanie2`
FOREIGN KEY (`kierownik_id`)
REFERENCES `bank`.`Kierownik`(`kierownik_id`)
ON DELETE RESTRICT
ON UPDATE RESTRICT,
CONSTRAINT `fk_wplaty1`
FOREIGN KEY (`k_numerrachunku`)
REFERENCES `bank`.`Konto`(`numerRachunku`)
ON DELETE RESTRICT
ON UPDATE RESTRICT)
ENGINE = InnoDB;

```

Tabela Wpłaty

```

CREATE TABLE `bank`.`Przelewy` (
  `przelew_id` INT(5) NOT NULL,
  `numerNIK1` INT(8) NOT NULL ,
  `numerNIK2` INT(8) NOT NULL ,
  `kwota` FLOAT(10) NOT NULL ,
  `data` DATE NOT NULL ,
  `zatwierdzone` BIT NULL ,
  `bankier_id` INT(4) NULL ,
  `k_numerrachunku` INT(8) NOT NULL ,
  `kierownik_id` INT(4) NULL ,
  UNIQUE (`numerNIK1`,`numerNIK2`),
  PRIMARY KEY (`przelew_id`),
  INDEX (`data`),
  CONSTRAINT `fk_przelewu`
  FOREIGN KEY (`bankier_id`)
  REFERENCES `bank`.`Bankier` (`bankier_id`)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT,
  CONSTRAINT `fk_zatwierdzanie1`
  FOREIGN KEY (`kierownik_id`)
  REFERENCES `bank`.`Kierownik` (`kierownik_id`)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT,
  CONSTRAINT `fk_przelewu1`
  FOREIGN KEY (`k_numerrachunku`)
  REFERENCES `bank`.`Konto` (`numerRachunku`)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT)
ENGINE = InnoDB;

```

Tabela Przelewy

```

CREATE TABLE `bank`.`Lokata` (`lokata_id` INT(4) NOT NULL AUTO_INCREMENT,
  `stan` FLOAT(10) NOT NULL ,
  `oprocentowanie` INT(3) NOT NULL ,
  `k_numerNIK` INT(8) NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `fk_lokaty`
  FOREIGN KEY (`k_numerNIK`)
  REFERENCES `bank`.`Klient` (`numerNIK`)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT )
ENGINE = InnoDB;

```

Tabela Lokata

```

CREATE TABLE `bank`.`Kredyt` ( `numerRachunku` INT(20) NOT NULL ,
  `kwota` FLOAT(10) NOT NULL ,
  `k_numerNIK` INT(8) NULL ,
  PRIMARY KEY (`numerRachunku`),
  CONSTRAINT `fk_kredytu`
  FOREIGN KEY (`k_numerNIK`)
  REFERENCES `bank`.`Klient` (`numerNIK`)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT)
ENGINE = InnoDB;

```

Tabela Kredyt

4.2 Implementacja mechanizmów przetwarzania danych

Indeksy tabel zostały wygenerowane jednocześnie z tabelami. Poniżej przedstawiono utworzenie widoków.

```
create view `klienci_lokata_kredyt` as
select imie as Imię, nazwisko as Nazwisko,
numerNIK as 'Numer Indyfikacyjny Klienta',
numerRachunku as 'Numer Rachunku Bankowego'
from
klient kl
inner join kredyt kr on (kl.numerNIK = kr.k_numerNIK)
inner join lokata lo on (kl.numerNIK = lo.k_numerNIK);

'WIDOK KLIENCI_LOKATA_KREDYT'
```

```
create view `wlascciele_kont` as
select imie as Imię, nazwisko as Nazwisko,
numerRachunku as 'Numer Rachunku',
numerNIK as 'Indywidualny Numer Klienta'
from
klient join konto on (k_numerNIK = numerNIK);

'WIDOK WLASCICIELE_KONT'
```

```
create view `ostatnie_przelewy` as
select przelew_id as 'ID przelewu', imie as Imię, nazwisko as Nazwisko,
numerNIK1 as 'Numer ID nadawcy', numerNIK2 as 'Numer ID odbiorcy',
numerRachunku as 'Numer rachunku nadawcy', data as 'Data przelewu'
from
klient
join przelewy on (numerNIK1 = numerNIK)
join konto on (k_numerNIK = numerNIK)
where
przelewy.data > date_sub(current_date(), interval 5 day);

'WIDOK OSTATNIE_PRZELEWY'
```

Następnym etapem było założenie wyzwalaczy:

```
use `bank`;
drop trigger if exists `bank`.`czas_wplaty`;
use `bank`;
create trigger `bank`.`czas_wplaty`
before insert on `wplaty` for each row
begin
set new.data = now();
end
```

```
use `bank`;
drop trigger if exists `bank`.`czas_wyplaty`;
use `bank`;
create trigger `bank`.`czas_wyplaty`
before insert on `wyplaty` for each row
begin
set new.data = now();
end
```

```

use `bank`;
drop trigger if exists `bank`.`czas_przelewy`;
use `bank`;
create trigger `bank`.`czas_przelewy`
before insert on `przelewy` for each row
begin
set new.data = now();
end

```

```

use `bank`;
drop trigger if exists `bank`.`przelewy_insert`;
use `bank`;
create definer = current_user trigger `bank`.`przelewy_insert`
before insert on `przelewy` for each row
begin
if przelewy.kwota > 50000 then
set message_text = 'Transakcja wymaga zatwierdzenia'
end if;
end

```

4.3 Implementacja uprawnień i innych zabezpieczeń

Uprawnienia dla roli użytkowników zostały ustawione zgodnie z tabelą w punkcie 3.1.5. Nadanie uprawnień poprzedza sprawdzenie czy użytkownik o takiej samej nazwie nie istnieje już w systemie i ewentualne usunięcie go. Poniżej przedstawiono kolejne etapy nadawania uprawnień, razem z utworzeniem przykładowych kont dla:

- Kierownika:

```

set sql_mode = '';
grant usage on *.* to k_kierownik;
drop user k_kierownik;
set sql_mode = 'TRADITIONAL,ALLOW_INVALID_DATES';
create user 'k_kierownik' identified by 'Andrzej';

grant all on table `bank`.`bankier` to 'k_kierownik';
grant all on table `bank`.`klient` to 'k_kierownik';
grant all on table `bank`.`konto` to 'k_kierownik';
grant all on table `bank`.`kredyt` to 'k_kierownik';
grant all on table `bank`.`lokata` to 'k_kierownik';
grant all on table `bank`.`przelewy` to 'k_kierownik';
grant all on table `bank`.`wpłaty` to 'k_kierownik';
grant all on table `bank`.`wyплаты` to 'k_kierownik';

```

- Bankiera:

```
set sql_mode = '';
grant usage on *.* to b_bankier;
drop user b_bankier;
set sql_mode = 'TRADITIONAL,ALLOW_INVALID_DATES';
create user 'b_bankier' identified by 'Antek';

grant insert, select, update, delete on table `bank`.`klient` to 'b_bankier';
grant all on table `bank`.`konto` to 'b_bankier';
grant insert, select, update, delete on table `bank`.`kredyt` to 'b_bankier';
grant insert, select, update, delete on table `bank`.`lokata` to 'b_bankier';
grant insert, select, update on table `bank`.`przelewy` to 'b_bankier';
grant insert, select, update on table `bank`.`wplaty` to 'b_bankier';
grant insert, select, update on table `bank`.`wyplaty` to 'b_bankier';
```

- Klienta:

```
set sql_mode = '';
grant usage on *.* to k_klient;
drop user k_klient;
set sql_mode = 'TRADITIONAL,ALLOW_INVALID_DATES';
create user 'k_klient' identified by 'Marian';

grant all on table `bank`.`konto` to 'k_klient';
grant select on table `bank`.`kredyt` to 'k_klient';
grant select, update on table `bank`.`lokata` to 'k_klient';
grant insert, select on table `bank`.`przelewy` to 'k_klient';
grant insert, select on table `bank`.`wplaty` to 'k_klient';
grant insert, select on table `bank`.`wyplaty` to 'k_klient';
```

4.4 Testowanie bazy danych na przykładowych danych

```
-----
-- Data for table `bank`.`Klient`
-----

START TRANSACTION
USE `bank`;
INSERT INTO `bank`.`Klient`(`numerNIK`, `haslo`, `imie`, `nazwisko`, `wiek`)
VALUES ('13253265', 'a5f495f02af0c779116342cff78dfbbc', 'Sebastian', 'Szkoluda', '21');
INSERT INTO `bank`.`Klient`(`numerNIK`, `haslo`, `imie`, `nazwisko`, `wiek`)
VALUES ('13253265', '6656910800c58e1e9e6bc8230805a381', 'Radek', 'Auguscik', '21');
INSERT INTO `bank`.`Klient`(`numerNIK`, `haslo`, `imie`, `nazwisko`, `wiek`)
VALUES ('57253365', '1d7c2923c1684726dc23d2901c4d8157', 'Adam', 'Jan', '42');
INSERT INTO `bank`.`Klient`(`numerNIK`, `haslo`, `imie`, `nazwisko`, `wiek`)
VALUES ('25258215', 'fa27ef3ef6570e32a79e74deca7c1bc3', 'Jan', 'Adam', '36');
COMMIT;

-----
-- Data for table `bank`.`Konto`
-----

START TRANSACTION;
USE `bank`;
INSERT INTO `bank`.`Konto`(`numerRachunku`, `nazwaRachunku`, `typRachunku`, `saldo`, `k_numerNIK`)
VALUES ('09249010440000320094007370', 'Rachunek główny', 'standard', '2321', '13253265');
INSERT INTO `bank`.`Konto`(`numerRachunku`, `nazwaRachunku`, `typRachunku`, `saldo`, `k_numerNIK`)
VALUES ('84 2490 0005 0000 4566 9889 2694', 'Rachunek dodatkowy', 'VIP', '25321', '13253265');
INSERT INTO `bank`.`Konto`(`numerRachunku`, `nazwaRachunku`, `typRachunku`, `saldo`, `k_numerNIK`)
VALUES ('30 1540 1056 2069 5000 0804 0001', 'Rachunek walutowy', 'VIP', '41232', '57253365');
INSERT INTO `bank`.`Konto`(`numerRachunku`, `nazwaRachunku`, `typRachunku`, `saldo`, `k_numerNIK`)
VALUES ('09 2490 1044 2069 5000 0804 0001', 'Rachunek walutowy', 'Standard', '36436', '25258215');
COMMIT;

-----
-- Data for table `bank`.`Bankier`
-----

START TRANSACTION;
USE `bank`;
INSERT INTO `bank`.`Bankier`(`bankier_id`, `haslo`, `imie`, `nazwisko`)
VALUES ('1325', 'a5f495f02af0c779116342cff78dfbbc', 'Sebastian', 'Kowalski');
INSERT INTO `bank`.`Bankier`(`bankier_id`, `haslo`, `imie`, `nazwisko`)
VALUES ('1265', '6656910800c58e1e9e6bc8230805a381', 'Radek', 'Nowak');
INSERT INTO `bank`.`Bankier`(`bankier_id`, `haslo`, `imie`, `nazwisko`)
VALUES ('3365', '1d7c2923c1684726dc23d2901c4d8157', 'Adam', 'Adam');
INSERT INTO `bank`.`Bankier`(`bankier_id`, `haslo`, `imie`, `nazwisko`)
VALUES ('2525', 'fa27ef3ef6570e32a79e74deca7c1bc3', 'Jan', 'Jan');
COMMIT;

-----
-- Data for table `bank`.`Kierownik`
-----

START TRANSACTION;
USE `bank`;
INSERT INTO `bank`.`Kierownik`(`kierownik_id`, `haslo`, `imie`, `nazwisko`)
VALUES ('1265', 'a5f495f02af0c779116342cff78dfbbc', 'Jarek', 'Szkoluda');
INSERT INTO `bank`.`Kierownik`(`kierownik_id`, `haslo`, `imie`, `nazwisko`)
VALUES ('1265', '6656910800c58e1e9e6bc8230805a381', 'Mateusz', 'Auguscik');
INSERT INTO `bank`.`Kierownik`(`kierownik_id`, `haslo`, `imie`, `nazwisko`)
VALUES ('5365', '1d7c2923c1684726dc23d2901c4d8157', 'Jerzy', 'Jan');
INSERT INTO `bank`.`Kierownik`(`kierownik_id`, `haslo`, `imie`, `nazwisko`)
VALUES ('2215', 'fa27ef3ef6570e32a79e74deca7c1bc3', 'Adam', 'Adam');
COMMIT;
```

```

-----
-- Data for table `bank`.`Wplaty`
-----

START TRANSACTION;
USE `bank`;
INSERT INTO `bank`.`Wplaty` ( `numerNIK1`, `kwota`, `data` )
VALUES ( '13253265', '2341', '2016-01-21 15:32:43' );
INSERT INTO `bank`.`Wplaty` ( `numerNIK1`, `kwota`, `data` )
VALUES ( '33253265', '2456', '2016-02-26 11:23:43' );
INSERT INTO `bank`.`Wplaty` ( `numerNIK1`, `kwota`, `data` )
VALUES ( '32253265', '6532', '2017-05-24 21:42:11' );
INSERT INTO `bank`.`Wplaty` ( `numerNIK1`, `kwota`, `data` )
VALUES ( '14253265', '7621', '2014-08-24 22:11:45' );
COMMIT;

-----
-- Data for table `bank`.`Wyplaty`
-----

START TRANSACTION;
USE `bank`;
INSERT INTO `bank`.`Wyplaty` ( `numerNIK1`, `kwota`, `data` )
VALUES ( '13253265', '2641', '2016-11-24 15:32:43' );
INSERT INTO `bank`.`Wyplaty` ( `numerNIK1`, `kwota`, `data` )
VALUES ( '33253265', '6456', '2016-03-16 11:23:43' );
INSERT INTO `bank`.`Wyplaty` ( `numerNIK1`, `kwota`, `data` )
VALUES ( '32253265', '6582', '2017-09-14 21:42:11' );
INSERT INTO `bank`.`Wyplaty` ( `numerNIK1`, `kwota`, `data` )
VALUES ( '14253265', '621', '2014-05-24 22:11:45' );
COMMIT;

-----
-- Data for table `bank`.`Przelewy`
-----

START TRANSACTION;
USE `bank`;
INSERT INTO `bank`.`Przelewy` ( `numerNIK1`, `numerNIK2`, `kwota`, `data` )
VALUES ( '13253265', '33253265', '2341', '2016-01-21 15:32:43' );
INSERT INTO `bank`.`Przelewy` ( `numerNIK1`, `numerNIK2`, `kwota`, `data` )
VALUES ( '33253265', '13253265', '2456', '2016-02-26 11:23:43' );
INSERT INTO `bank`.`Przelewy` ( `numerNIK1`, `numerNIK2`, `kwota`, `data` )
VALUES ( '32253265', '14253265', '6532', '2017-05-24 21:42:11' );
INSERT INTO `bank`.`Przelewy` ( `numerNIK1`, `numerNIK2`, `kwota`, `data` )
VALUES ( '14253265', '32253265', '7621', '2014-08-24 22:11:45' );
COMMIT;

-----
-- Data for table `bank`.`Kredyt`
-----

START TRANSACTION;
USE `bank`;
INSERT INTO `bank`.`Kredyt` ( `numerRachunku`, `kwota` )
VALUES ( '84 2490 0005 0000 4566 9889 2694', '26141' );
INSERT INTO `bank`.`Kredyt` ( `numerRachunku`, `kwota` )
VALUES ( '30 1540 1056 2069 5000 0804 0001', '64256' );
INSERT INTO `bank`.`Kredyt` ( `numerRachunku`, `kwota` )
VALUES ( '32 1541 1056 2069 5000 0804 0001', '65482' );
INSERT INTO `bank`.`Kredyt` ( `numerRachunku`, `kwota` )
VALUES ( '34 1545 1056 2069 5000 0804 0001', '11621' );
COMMIT;

```

```

-----
-- Data for table `bank`.`Lokata`
-----

START TRANSACTION;
USE `bank`;
INSERT INTO `bank`.`Lokata` ( `stan`,`oprocentowanie` )
VALUES ( '3265','15' );
INSERT INTO `bank`.`Lokata` ( `stan`,`oprocentowanie` )
VALUES ( '3565','4' );
INSERT INTO `bank`.`Lokata` ( `stan`,`oprocentowanie` )
VALUES ( '5325','17' );
INSERT INTO `bank`.`Lokata` ( `stan`,`oprocentowanie` )
VALUES ( '1425','3' );
COMMIT;

```

Polecenia SELECT * FROM kolejnych tabel, zwróciły poprawne dane:

bankier_id	haslo	imie	nazwisko
1265	6656910800c58e1e9e6bc8230805a381	Radek	Nowak
1325	a5f495f02af0c779116342cff78dfbbc	Sebastian	Kowalski
2525	fa27ef3ef6570e32a79e74deca7c1bc3	Jan	Jan
3365	1d7c2923c1684726dc23d2901c4d8157	Adam	Adam

Tabela bankier

kierownik_id	haslo	imie	nazwisko
1265	6656910800c58e1e9e6bc8230805a381	Mateusz	Auguscik
1285	a5f495f02af0c779116342cff78dfbbc	Jarek	Szkoluda
2215	fa27ef3ef6570e32a79e74deca7c1bc3	Adam	Adam
5365	1d7c2923c1684726dc23d2901c4d8157	Jerzy	Jan

Tabela kierownik

numerNIK	haslo	imie	nazwisko	wiek
13253265	a5f495f02af0c779116342cff78dfbbc	Sebastian	Szkoluda	21
23251255	6656910800c58e1e9e6bc8230805a381	Radek	Auguscik	21
25258215	fa27ef3ef6570e32a79e74deca7c1bc3	Jan	Adam	36
57253365	1d7c2923c1684726dc23d2901c4d8157	Adam	Jan	42

Tabela klient

numerRachunku	nazwaRachunku	typRachunku	saldo	k_numerNIK
09 2490 1044 2069 5000 0804 0001	Rachunek walutowy	Standard	36436	25258215
09249010440000320094007370	Rachunek główny	standard	2321	13253265
30 1540 1056 2069 5000 0804 0001	Rachunek walutowy	VIP	41232	57253365
84 2490 0005 0000 4566 9889 2694	Rachunek dodatkowy	VIP	25321	13253265

Tabela konto

numerRachunku	kwota	k_numerNIK
30 1540 1056 2069 5000 0804 0001	64256	25258215
32 1541 1056 2069 5000 0804 0001	65482	13253265
34 1545 1056 2069 5000 0804 0001	11621	23251255
84 2490 0005 0000 4566 9889 2694	26141	23251255

Tabela kredyt

id	stan	oprocentowanie	k_numerNIK
1	3265	15	13253265
2	3565	4	13253265
3	5325	17	25258215
4	1425	3	57253365

Tabela lokata

numerNIK2	kwota	data	zatwierdzone	bankier_id	k_numerNIK
33253265	2341	2016-01-21	NULL	1265	NULL
32253265	7621	2014-08-24	NULL	NULL	23251255
14253265	6532	2017-05-24	NULL	NULL	57253365
13253265	2456	2016-02-26	NULL	3365	NULL

Tabela przelewy

kwota	data	zatwierdzone	bankier_id	k_numerNIK
2456	2016-02-26	NULL	NULL	13253265
2341	2016-01-21	NULL	NULL	23251255
6582	2017-09-14	NULL	1265	NULL
7621	2014-08-27	NULL	NULL	57253365

Tabela wpłaty

kwota	data	zatwierdzone	bankier_id	k_numerNIK
2641	2016-11-24	NULL	NULL	13253265
6456	2016-03-16	NULL	1265	NULL
621	2014-05-24	NULL	NULL	23251255
6582	2017-09-14	NULL	2525	NULL

Tabela wypłaty

Testowanie widoków

```
1 • select *
2 from ostatnie_przelewy;
```

<

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	ID przelewu	Imię	Nazwisko	Numer ID nadawcy	Numer ID odbiorcy	Numer rachunku nadawcy	Data przelewu
	1	Mateusz	Caban	1	2	21 2060 0002 5838 6222 5495 5960	2017-12-17
	2	Sebastian	Kowalski	4	1	08 1440 1039 8559 2607 3785 3533	2017-12-17
	3	Kevin	Paprvczka	3	2	18 8229 1015 2842 3801 8037 5917	2017-12-17
	4	Janusz	Jasinski	5	4	31 8110 1023 4997 4168 1795 2900	2017-12-17
	5	Kevin	Paprvczka	3	1	18 8229 1015 2842 3801 8037 5917	2017-12-17

Test widoku ostatnie_przelewy

```
1 • select *
2 from klienci_lokata_kredyt;
```

<

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	Imię	Nazwisko	Numer Identyfikacyjny Klienta	Numer Rachunku Bankowego
	Mateusz	Caban	1	21 2060 0002 5838 6222 5495 5960
	Kevin	Paprvczka	3	18 8229 1015 2842 3801 8037 5917
	Janusz	Jasinski	5	31 8110 1023 4997 4168 1795 2900
	Marcin	Stefanski	6	62 8131 2331 1893 3910 5462 6354

Test widoku klienci_lokata_kredyt

1	•	select *
2		from wlasciciele_kont;

<				
Result Grid		Filter Rows:		Export:
		Wrap Cell Content:		
	Imię	Nazwisko	Numer Rachunku	Indywidualny Numer Klienta
	Mateusz	Caban	21 2060 0002 5838 6222 5495 5960	1
	Kamil	Szymborski	11 1050 1357 5976 7909 3412 1622	2
	Kevin	Paprczka	18 8229 1015 2842 3801 8037 5917	3
	Sebastian	Kowalski	08 1440 1039 8559 2607 3785 3533	4
	Janusz	Jasinski	31 8110 1023 4997 4168 1795 2900	5
	Marcin	Stefanski	62 8131 2331 1893 3910 5462 6354	6
	Juliusz	Brzozowski	76 5683 3410 6171 6003 1258 3365	7
	Pawel	Gornv	16 7063 2558 2777 5978 0502 6048	8

Test widoku wlasciciele_kont

Następnie postanowiliśmy przetestować funkcjonalność bazy, tzn. próbowaliśmy dodawać do niej wartości, których nie mogliśmy dodać, np. dwa te same pesele, dwa te same numery identyfikacyjne lub modyfikacja kluczy podstawowych.

```
insert into `bank`.`klient`
values ('00000009','klient9','Kamil','Paczos','25','90121103965');
```

✖ 1 11:20:04 insert into `bank`.`klient` values ('00000009','klient9','Kamil','Paczos','25','90121103965')
Error Code: 1062. Duplicate entry '90121103965' for key 'pesel'

```
insert into `bank`.`przelewy`
values('0002','00000002','00000001','1000','2017-05-20',1,'0001','11 1050 1357 5976 7909 3412 1622','00000002');
```

✖ 2 11:25:29 insert into `bank`.`przelewy` values('0002','00000002','00000001','1000','2017-05-20',1,'0001','11 1050 1357 5976 7909 ...
Error Code: 1062. Duplicate entry '2' for key 'PRIMARY'

```
alter table `bank`.`klient`
modify numerNIK varchar(10);
```

✖ 3 11:30:24 alter table `bank`.`klient` modify numerNIK varchar(10)
Error Code: 1833. Cannot change column 'numerNIK': used in a foreign key constraint 'fk_konta' of table 'bank.konto'

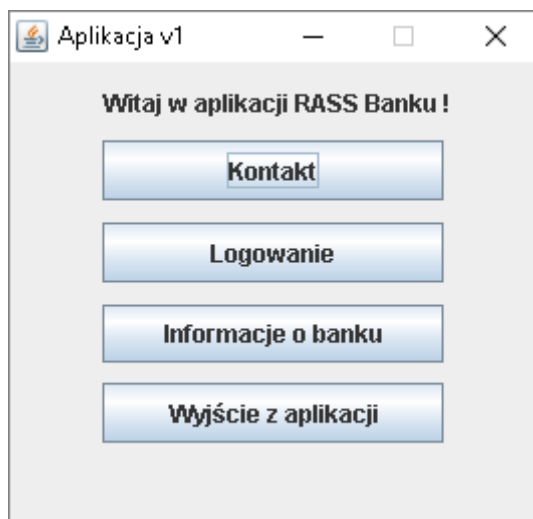
5 Implementacja i testy aplikacji

5.1 Instalacja i konfigurowanie systemu

Do poprawnego działania aplikacji potrzebna jest instalacja i konfiguracja serwera MySQL. Pełna instrukcja instalacji znajduje się w załączniku oraz na stronie <http://taxmachine.pl/taxmachine-2/konfiguracja/konfiguracja-mysql.html>. Kolejnym etapem jest uruchomienie skryptu SQL na bazie danych. W ramach projektu serwer MySQL został już zainstalowany i skonfigurowany. Aplikacja użytkownika nie wymaga instalacji. Należy pobrać plik bank_project.zip, wypakować i uruchomić plik BankApp.jar dwukrotnym przyciskiem myszy. Należy również tutaj nadmienić że aplikacja była implementowana w języku Java przy użyciu pluginu WindowsBuilder w połączeniu z technologią Swing, podłączenie aplikacji do stworzonej w poprzednim etapie bazy danych zostało wykonane przy użyciu JDBC (Java DataBase Connectivity).

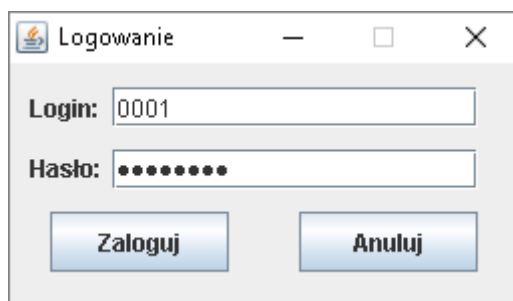
5.2 Instrukcja użytkownika aplikacji

Po uruchomieniu aplikacji pojawia się okno główne w którym możemy uzyskać informacje na temat banku. Aby uzyskać dodatkowe funkcje należy zalogować się loginem i hasłem. Podany odpowiedni login i hasło otwiera nam okno zgodne z autoryzacją, jeśli będą to dane bankiera to ukaże się nam okno z funkcjami przysługującymi bankierowi.



Rysunek 12: Okno główne aplikacji

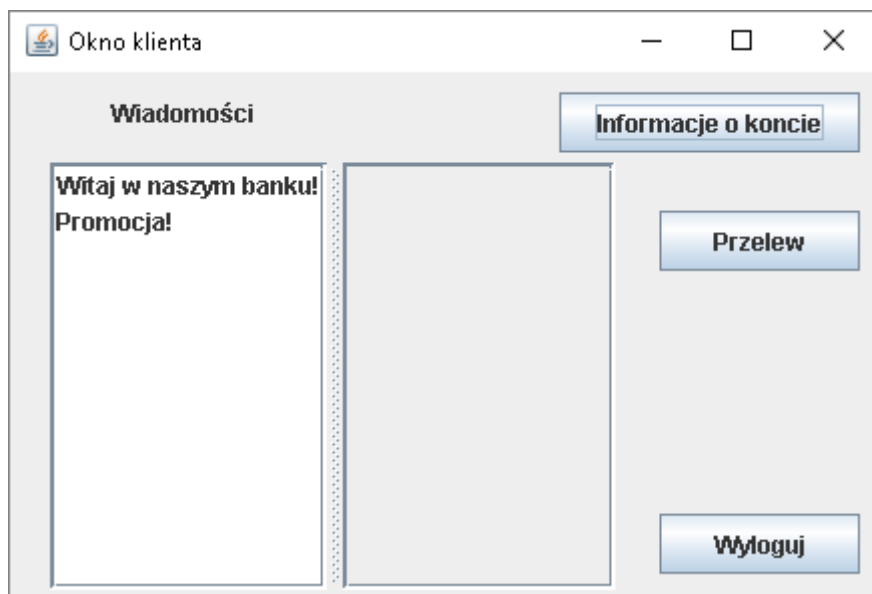
Po kliknięciu na przycisk Logowanie ukaże się nam takie okno:



Rysunek 13: Okno logowania do aplikacji

Jak widać w oknie tym podajemy login i hasło. Hasło zostało zabezpieczone czarnymi kropkami tak aby podczas wprowadzania hasło nie mogło zostać odczytane przez niepożądane osoby.

Gdy już uda nam się przejść proces logowania ukaże nam się jedno z trzech poniższych okienek w zależności od tego kim jesteśmy: klientem, bankierem, kierownikiem banku.



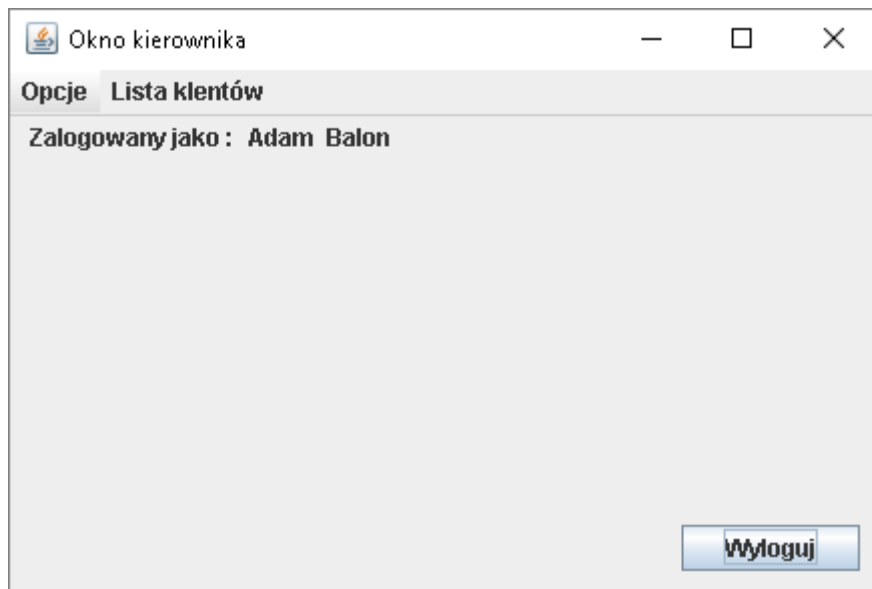
Rysunek 14: Okno klienta

W oknie klienta widoczne są takie funkcjonalności jak informacje o własnym koncie, wiadomości przychodzące na konto oraz przycisk umożliwiający zrobienie przelewu.



Rysunek 15: Okno bankiera

W oknie bankiera mamy więcej funkcjonalności. Bankier może wyświetlić listę danych na temat klientów zarejestrowanych w bazie. Może wykonywać wszystkie typy transakcji takie jak: wpłaty, wypłaty, przelewy, są one widoczne po rozwinięciu pola Transakcje na menu bar. Pojawia się tutaj również możliwość rejestracji nowych klientów.



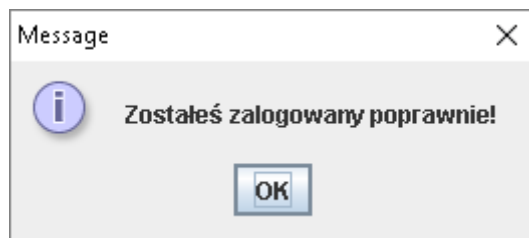
Rysunek 16: Okno kierownika

W oknie kierownika pojawiają się takie funkcjonalności jak w oknie bankiera, lecz są one rozwijane po kliknięciu opcji na menu bar.

W każdym z przypadków pojawia się również przycisk Wyloguj umożliwiający wylogowanie z konta.

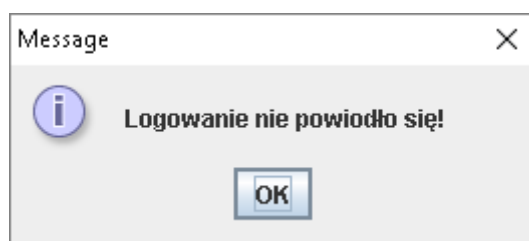
5.3 Testowanie opracowanych funkcji systemu

Przetestowany został system logowania. Gdy logowanie się powiedzie powinno ukazać się nam takie okno:



Rysunek 17: Poprawne zalogowanie do aplikacji

W przeciwnym wypadku będzie to taki komunikat, który będzie oznaczał, że dane nie zostały wprowadzone poprawnie:



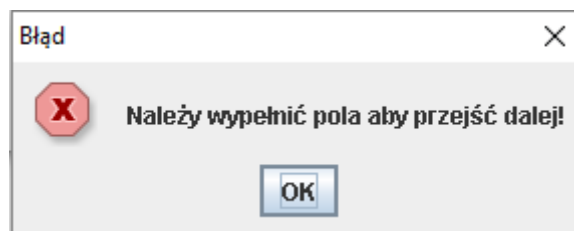
Rysunek 18: Błędne zalogowanie do aplikacji

Został również przetestowany mechanizm rejestracji. Gdy nie wprowadzimy wymaganych pól nie będziemy mogli przejść do kolejnego etapu rejestracji:

A window titled "Rejestracja" with standard Windows window controls (minimize, maximize, close). The form contains six labels with corresponding text input fields: "Imię:", "Nazwisko:", "Pesel:", "Wiek:", "Hasło:", and "Numer NIK:". At the bottom of the form are two buttons: "Przejdź dalej" and "Anuluj".

Rysunek 19: Okno pierwszego etapu rejestracji

Próba przejścia dalej kończy się takim komunikatem:

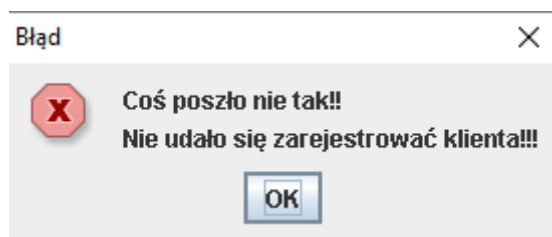


Rysunek 20: Komunikat dla nie wypełnienia wymaganych pól

Gdy już uda nam się przejść ten etap ukaze nam sie kolejne okno:

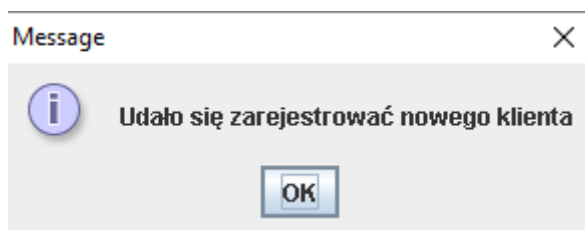
Rysunek 21: Okno drugiego etapu rejestracji

W tym oknie należy również wypełnić wszystkie pola, w przeciwnym wypadku ukaze się taki komunikat:



Rysunek 22: Komunikat dla nie wypełnienia wymaganych pól

Kiedy wszystko wykonamy poprawnie powinien pokazać się taki oto komunikat, który oznacz że udało się zarejestrować nowego klienta:



Rysunek 23: Komunikat poprawnej rejestracji klienta

Nasz nowy klient pojawia się na liście klientów zaraz po rejestracji, wystarczy kliknąć lista klientów na menu bar:



Rysunek 24: Komunikat poprawnej rejestracji klienta

5.4 Omówienie wybranych rozwiązań programistycznych

5.4.1 Implementacja interfejsu dostępu do bazy danych

Klasa łącząca aplikację z bazą danych za pomocą JDBC:

```
public class ConnectionDBC {
    private Connection myConn;
    public ConnectionDBC() throws Exception{
        Class.forName("com.mysql.jdbc.Driver");

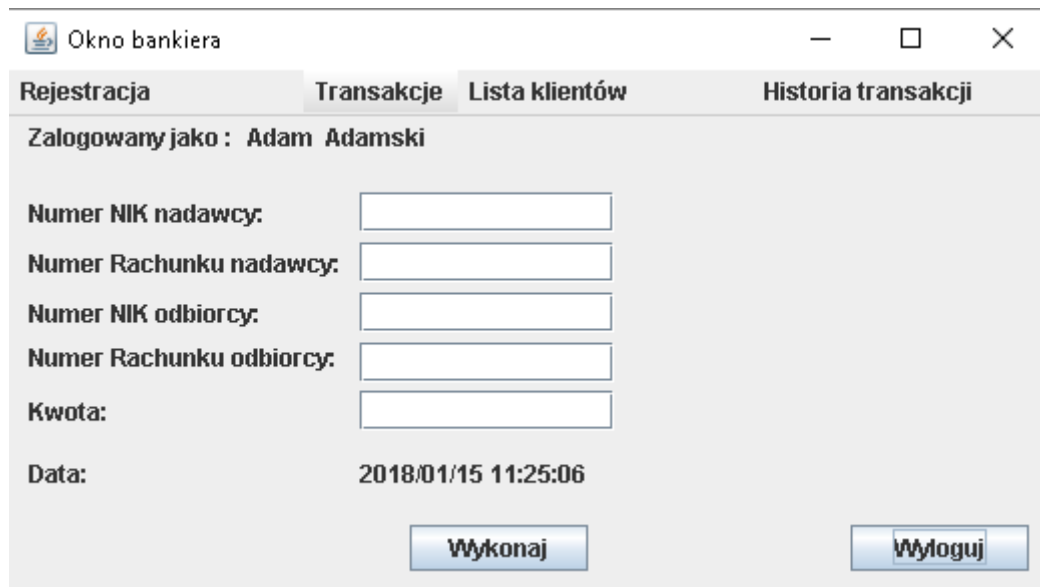
        Properties props = new Properties();
        props.load(new FileInputStream("bank.properties"));

        String user = props.getProperty("user");
        String password = props.getProperty("password");
        String dburl = props.getProperty("dburl");

        myConn = DriverManager.getConnection(dburl, user, password);

        System.out.println("DB connection succesful to: " + dburl);
    }
    public Connection getMyConn() {
        return myConn;
    }
    public void setMyConn(Connection myConn) {
        this.myConn = myConn;
    }
}
```

Przykład dostępu do bazy danych przy wykonywaniu przelewów:



Rysunek 25: Okno przelewu

```
String s1SQL = "SELECT * from Konto where numerRachunku = " + numerNadawcy;
String s2SQL = "SELECT * from Konto where numerRachunku = " + numerOdbiorcy;
String s3SQL = "UPDATE Konto SET saldo = ? WHERE numerRachunku = ? ";
String s4SQL = "UPDATE Konto SET saldo = ? WHERE numerRachunku = ? ";
try {
    ConnectionDBC con = new ConnectionDBC();
    System.out.println("Bylem tu!!");
    PreparedStatement myStmt = con.getMyConn().prepareStatement(s1SQL);
    ResultSet rs = myStmt.executeQuery(s1SQL);
    PreparedStatement myStmt1 = con.getMyConn().prepareStatement(s1SQL);
    ResultSet rs1 = myStmt1.executeQuery(s2SQL);

    while (rs.next()) {
        numerRachunkuN = rs.getString("numerRachunku");
        saldoN = rs.getFloat("saldo");
        System.out.println(numerRachunkuN);
        System.out.println(saldoN);
    }
    while (rs1.next()) {
        numerRachunkuO = rs1.getString("numerRachunku");
        saldoO = rs1.getFloat("saldo");
    }

    String s5SQL = "Insert into 'przelewy' (przelew_id,numerNIK1,numerNIK2,"
+ "kwota,data,zatwierdzone,f_bankier_id,f_numerRachunku,f_kierownik_id)"
+ " values (?,?,?,?,?,?,?,?,?,?)";
    PreparedStatement ps = con.getMyConn().prepareStatement(s5SQL);
    ps.setInt(1, 13);
    ps.setString(2, numerNIK1);
    ps.setString(3, numerNIK2);
    ps.setFloat(4, kwota);
    ps.setString(5, data);
    ps.setBoolean(6, true);
```



```

ps.setInt(7, 1);
ps.setString(8, numerNadawcy);
ps.setInt(9, 4);
ps.execute();
newSaldoN = saldoN - kwota;
newSaldoO = saldoO + kwota;
PreparedStatement myStmt2 = con.getMyConn().prepareStatement(s3SQL);
myStmt2.setFloat(1, newSaldoN);
myStmt2.setString(2, numerRachunkuN);
myStmt2.executeUpdate();
PreparedStatement myStmt3 = con.getMyConn().prepareStatement(s4SQL);
myStmt3.setFloat(1, newSaldoO);
myStmt3.setString(2, numerRachunkuO);
myStmt3.executeUpdate();
} catch (Exception e) {
    System.out.println("Cos nie taak!!");
    e.printStackTrace();
}}

```

Przykład dostępu do bazy danych przy wykonywaniu wpłat:

Rysunek 26: Okno wpłaty

```

String s1SQL = "SELECT * from Konto where numerRachunku = " + numerNadawcy;
String s3SQL = "UPDATE Konto SET saldo = ? WHERE numerRachunku = ? ";

try {
    ConnectionDBC con = new ConnectionDBC();
    System.out.println("Bylem tu!!");
    PreparedStatement myStmt = con.getMyConn().prepareStatement(s1SQL);
    //myStmt.setString(1, numerNadawcy);
    ResultSet rs = myStmt.executeQuery(s1SQL );

    while (rs.next()) {
        numerRachunkuN = rs.getString("numerRachunku");
        saldoN = rs.getFloat("saldo");
    }
}

```

```

System.out.println(numerRachunkuN);
System.out.println(saldoN);
}
String s5SQL = "Insert into 'wplaty'
(numerNIK,kwota,data,zatwierdzone,bankier_id,k_numerrachunku)"
+ " values (?,?,?,?,?,?,?)";

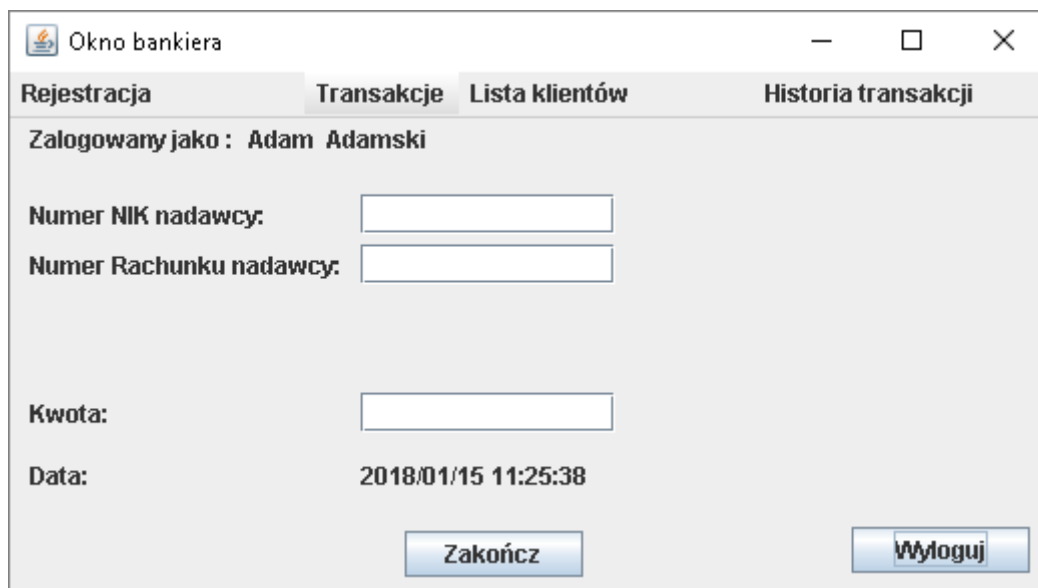
PreparedStatement ps = con.getMyConn().prepareStatement(s5SQL);
ps.setString(1, numerNIK1);
ps.setFloat(2, kwota);
ps.setString(3, data);
// ps.setInt(6, 1);
ps.setBoolean(4, false);
ps.setInt(5, 1);
ps.setString(6, numerNadawcy);
// ps.setInt(7, 4);
ps.execute();

newSaldoN = saldoN + kwota;
PreparedStatement myStmt2 = con.getMyConn().prepareStatement(s3SQL);
myStmt2.setFloat(1, newSaldoN);
myStmt2.setString(2, numerRachunkuN);
myStmt2.executeUpdate();

} catch (Exception e1) {
System.out.println("Cos nie taak!!");
e1.printStackTrace();
}

```

Przykład dostępu do bazy danych przy wykonywaniu wypłat:



Rysunek 27: Okno wypłaty

```
String s1SQL = "SELECT * from Konto where numerRachunku = " + numerNadawcy;  
String s3SQL = "UPDATE Konto SET saldo = ? WHERE numerRachunku = ? ";
```

```
try {  
    ConnectionDBC con = new ConnectionDBC();  
    System.out.println("Bylem tu!!");  
    PreparedStatement myStmt = con.getMyConn().prepareStatement(s1SQL);  
    //myStmt.setString(1, numerNadawcy);  
    ResultSet rs = myStmt.executeQuery(s1SQL );
```

```
    while (rs.next()) {  
        numerRachunkuN = rs.getString("numerRachunku");  
        saldoN = rs.getFloat("saldo");  
        System.out.println(numerRachunkuN);  
        System.out.println(saldoN);  
    }  
    String s5SQL = "Insert into 'wyplaty' "  
        "(numerNIK,kwota,data,zatwierdzone,bankier_id,k_numerrachunku)"  
        + " values (?,?,?,?,?,?,?)";  
    PreparedStatement ps = con.getMyConn().prepareStatement(s5SQL);  
    ps.setString(1, numerNIK1);  
    ps.setFloat(2, kwota);  
    ps.setString(3, data);  
    // ps.setInt(6, 1);  
    ps.setBoolean(4, false);  
    ps.setInt(5, 1);  
    ps.setString(6, numerNadawcy);  
    // ps.setInt(7, 4);  
  
    ps.execute();
```

```

newSaldoN = saldoN - kwota;
PreparedStatement myStmt2 = con.getMyConn().prepareStatement(s3SQL);
myStmt2.setFloat(1, newSaldoN);
myStmt2.setString(2, numerRachunkuN);
myStmt2.executeUpdate();
} catch (Exception e1) {
System.out.println("Cos nie taak!!");
e1.printStackTrace();
}

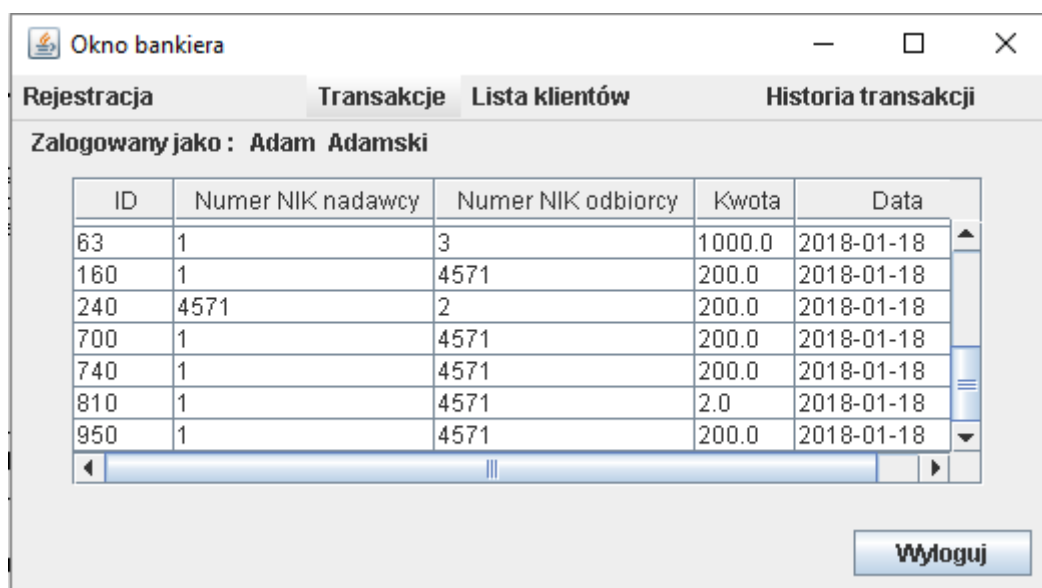
```

Kolejną rzeczą jest historia transakcji oraz lista klientów. Zostały one wykonane w opraciu o komponenty JTable oraz JScrollPane, ze względu na lepszy dostęp do informacji o klientach oraz ich transakcjach.

Listy te wyglądają następująco:



Rysunek 28: Lista klientów



Rysunek 29: Lista transakcji

Sama implementacja prezentuje się następująco. W przypadku listy klientów skorzystaliśmy z KlientDAO (Data Access Object) ze względu na zamianę danych poszczególnego wiersza w bazie danych na obiekt klienta z atrybutami zgodnymi z kolumnami w tabeli klient. Zostało to zrobione przy pomocy zaimplementowanej metody ConvertRowToClient():

```
private Client convertRowToClient(ResultSet myRs) throws SQLException{
    int numerNIK = myRs.getInt("numerNIK");
    String haslo = myRs.getString("haslo");
    String imie = myRs.getString("imie");
    String nazwisko = myRs.getString("nazwisko");
    int wiek = myRs.getInt("wiek");
    BigDecimal pesel = myRs.getBigDecimal("pesel");
    Client tempClient = new Client(numerNIK, haslo, imie, nazwisko, wiek, pesel);

    return tempClient;
}
```

Natomiast samo utworzenie tabeli wygląda następująco:

```
DefaultTableModel model = new DefaultTableModel();
Object[] columnsName = new Object[6];
```

```
columnsName[0] = "numerNIK";
columnsName[1] = "haslo";
columnsName[2] = "imie";
columnsName[3] = "nazwisko";
columnsName[4] = "wiek";
columnsName[5] = "pesel";
```

```
model.setColumnIdentifiers(columnsName);
Object[] rowData = new Object[6];
```

```
try {
    ClientDAO dao = new ClientDAO();
    for(int i=0;i<dao.getAllClients().size();i++) {
        rowData[0] = dao.getAllClients().get(i).getNumerNIK();
        rowData[1] = dao.getAllClients().get(i).getHaslo();
        rowData[2] = dao.getAllClients().get(i).getImie();
        rowData[3] = dao.getAllClients().get(i).getNazwisko();
        rowData[4] = dao.getAllClients().get(i).getWiek();
        rowData[5] = dao.getAllClients().get(i).getPesel();

        model.addRow(rowData);
    }
    table.setModel(model);
    table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
    table.getColumnModel().getColumn(4).setPreferredWidth(50);
    table.getColumnModel().getColumn(5).setPreferredWidth(105);
} catch (Exception e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
}
```

```
});
```

Jednym z mechanizmów ochrony danych jest ukrywanie wpisywanych haseł zaimplementowane przy pomocy komponentu

```
JPasswordField field = new JPasswordField();
```

Sam mechanizm logowania wygląda następująco:

```
try {
    ConnectionDBC con = new ConnectionDBC();
    Statement myStmt = con.getMyConn().createStatement();
    ResultSet myRs =
    myStmt.executeQuery("SELECT * FROM bankier where bankier_id = " + value1);
    Statement myStmt1 = con.getMyConn().createStatement();
    ResultSet myRs1 =
    myStmt1.executeQuery("select * from klient where numerNIK = " + value1);
    Statement myStmt2 = con.getMyConn().createStatement();
    ResultSet myRs2 =
        myStmt2.executeQuery("select * from kierownik where kierownik_id = " + value1);
    while (myRs.next()) {
        user1 = myRs.getInt("bankier_id");
        pass1 = myRs.getString("haslo");
        name1 = myRs.getString("imie");
        name = myRs.getString("imie");
        surname = myRs.getString("nazwisko");
    }
    while (myRs1.next()) {
        user2 = myRs1.getInt("numerNIK");
        pass2 = myRs1.getString("haslo");
        name = myRs1.getString("imie");
        surname = myRs1.getString("nazwisko");
    }
    while (myRs2.next()) {
        user3 = myRs2.getInt("kierownik_id");
        pass3 = myRs2.getString("haslo");
        name = myRs2.getString("imie");
        surname = myRs2.getString("nazwisko");
    }

} catch (Exception e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
if(value1 == user2 && value2.equals(pass2)) {
    JOptionPane.showMessageDialog(null," Zostales zalogowany poprawnie!");
    ClientFrame klient = new ClientFrame(numerNIK);
    klient.clientFrame();
    frame.dispose();
}
else if(value1 == user1 && value2.equals(pass1)) {
    JOptionPane.showMessageDialog(null," Zostales zalogowany poprawnie!");

    BankierFrame bankier = new BankierFrame(numerNIK);
    bankier.bankierFrame();
    frame.dispose();
}
```

```

else if(value1 == user3 && value2.equals(pass3)) {
OptionPane.showMessageDialog(null," Zostales zalogowany poprawnie!");
KierownikFrame kierownik = new KierownikFrame(numerNIK);
kierownik.kierownikFrame();
frame.dispose();
}
else if(textField_Login.getText().equals("")&&textField_Haslo.getText().equals(""))
OptionPane.showMessageDialog(null, "Login i haslo nie zostaly podane!");
}
else {
OptionPane.showMessageDialog(null, "Logowanie nie powiodlo sie!");
}
}
}

```

6 Podsumowanie

Nasz projekt miał na celu stworzenie systemu obsługującego konta bankowe. Zrozumieć ten temat można na wiele różnych sposobów. Mogła to być aplikacja przeznaczona jedynie dla klientów, która umożliwiałaby im wykonywanie różnych aktywności związanych z ich kontem. Mogła to też być aplikacja skierowana do bankiera siedzącego w banku, który dzięki niej mógłby zarządzać kontami wszystkich klientów. Temat ten więc miał wiele płaszczyzn, na których można było go zrealizować. W naszym projekcie zdecydowaliśmy się na połączenie tych wszystkich płaszczyzn i stworzyliśmy aplikację skierowaną zarówno do klienta jak i bankiera czy kierownika. Patrząc na to z perspektywy czasu, nie był to zbyt trafny wybór, ze względu na to, że nie zdążyliśmy dopracować naszej aplikacji. Nie pojawiły się w niej takie funkcjonalności jak kredyt czy też lokata. Od strony wizualnej nasza aplikacja również nie jest zbyt dopracowana. Niemniej udało nam się zawrzeć w projekcie większość naszych założeń, które działały poprawnie. Największy problem sprawiło nam ułożenie modelu logicznego bazy, ponieważ nie mieliśmy do końca pomysłu jak dokładnie chcemy, aby nasz system wyglądał. Przez to realizacja tego etapu zajęła nam nieco więcej czasu niż zakładaliśmy. Po przebrnięciu przez ten etap, reszta projektu poszła nam całkiem dobrze. Podłączenie naszej aplikacji do bazy danych oraz jej zaprojektowanie poszło nam bardzo sprawnie. Dzięki temu projektowi mogliśmy przypomnieć sobie co nieco o bazach danych, a także o programowaniu w technologii Java, ponieważ na co dzień, nie pracujemy za bardzo w tej technologii. Mimo to, poradziliśmy sobie z tym projektem w zadowalającym nas stopniu.

7 Literatura

- <https://www.javatpoint.com/example-to-connect-to-the-mysql-database> - strona zawiera poradnik jak podłączyć się do bazy danych w technologii JDBC.
- <https://dev.mysql.com/doc/employee/en/> - strona poświęcona wszelkim informacjom na temat tworzenia bazy danych w MySQL.
- <http://leo.ugr.es/elvira/devel/Tutorial/Java/uiswing/components/intermediate.html> - zawiera informacje na temat komponentów w technologii Swing (Java)
- <https://www.developer.com/java/creating-a-jdbc-gui-application.html> - strona zawierająca informacje na temat tworzenia aplikacji desktopowej z wykorzystaniem bazy danych.
- <https://www.latex-tutorial.com/> - strona która pomogła w sprawach edycyjnych dokumentacji.
- <https://webroad.pl/inne/426-instalacja-lokalnego-zestawu-apache-mysql-i-php-wamp-server> - na stronie tej zawarte są informacje na temat lokalnego serwera utworzonego przy pomocy WampServer.
- https://www.visual-paradigm.com/support/documents/vpuserguide/3563/3564/85378_conceptual_1.html - informacje na temat tworzenia diagramów bazy danych w programie VisualParadigm 14.2.