

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

PROJEKT Z BAZ DANYCH

System bazodanowy obsługujący konto bankowe

AUTOR:

Maksim Zakharau

PROWADZĄCY ZAJĘCIA:

Dr inż. Robert Wójcik, K30W04D03

Indeks: 256629

OCENA PRACY:

Wrocław, 2022

Spis treści

Spis rysunków	4
1. Wstęp.....	6
1.1. Cel projektu	6
1.2. Zakres projektu	6
2. Analiza wymagań	7
2.1. Opis działania i schemat logiczny systemu	7
2.2. Wymagania funkcjonalne.....	8
2.2.1. Diagram przypadków użycia	8
2.2.2. Scenariusze wybranych przypadków użycia	8
2.3. Wymagania niefunkcjonalne.....	10
2.3.1. Wykorzystywane technologie i narzędzia	10
2.3.2. Wymagania dotyczące bezpieczeństwa systemu	11
2.4. Przyjęte założenia projektowe	11
3. Projekt systemu	12
3.1. Projekt bazy danych	12
3.1.1. Analiza rzeczywistości i uproszczony model konceptualny.....	12
3.1.2. Model logiczny i normalizacja.....	13
3.1.3. Model fizyczny i ograniczenia integralności danych.....	14
3.2. Projekt aplikacji użytkownika.....	15
3.2.1. Architektura aplikacji i diagramy projektowe.....	15
3.2.2. Interfejs graficzny i struktura menu	16
3.2.3. Projekt wybranych funkcji systemu	17
3.2.4. Metoda podłączania do bazy danych – integracja z bazą danych.....	20
3.2.5. Projekt zabezpieczeń na poziomie aplikacji	21
4. Implementacja systemu.....	22
4.1. Realizacja bazy danych	22
4.1.1. Tworzenie tabel i definiowanie ograniczeń	22
4.2. Realizacja elementów aplikacji	23
4.2.1. Obsługa menu	23
4.2.2. Walidacja i filtracja.....	26
4.2.3. Implementacja interfejsu dostępu do bazy danych	27
4.2.4. Implementacja wybranych funkcjonalności systemu.....	29
4.2.5. Implementacja mechanizmów bezpieczeństwa	31
5. Testowanie systemu	32
5.1. Testowanie opracowanych funkcji systemu	32
5.1.1. Testowanie tworzenia konta	32
5.1.2. Testowanie usuwania konta	34
5.1.3. Testowanie wpłaty.....	36
5.1.4. Testowanie przelewu	38

5.2. Testowanie mechanizmów bezpieczeństwa	41
5.3. Wnioski z testów	44
6. Podsumowanie	45
Literatura	46

Spis rysunków

RYSUNEK 1. Schemat komunikacji i struktura systemu	6
RYSUNEK 2. Diagram przypadków użycia	9
RYSUNEK 3. Uproszczony model konceptualny bazy danych	11
RYSUNEK 4. Model logiczny bazy danych	12
RYSUNEK 5. Model fizyczny bazy danych	12
RYSUNEK 6. Diagram klas aplikacji	14
RYSUNEK 7. Struktura okien aplikacji	15
RYSUNEK 8. Diagram stanów - Kredytowanie	17
RYSUNEK 9. Diagram stanów - Przelew	18
RYSUNEK 10. Diagram sekwencji dla podłączenia do bazy danych	19
RYSUNEK 11. Rejestracja użytkownika	24
RYSUNEK 12. Błąd rejestracji 1	25
RYSUNEK 13. Tabela testowa 1	32
RYSUNEK 14. Test rejestracji 2	33
RYSUNEK 15. Tabela testowa 2	33
RYSUNEK 16. Test usuwania 1	34
RYSUNEK 17. Test usuwania 2	35
RYSUNEK 18. Tabela testowa 3	35
RYSUNEK 19. Testowanie wpłaty 1	36
RYSUNEK 20. Tabela testowa 4	36
RYSUNEK 21. Testowanie wpłaty - Okno danych	36
RYSUNEK 22. Testowanie wpłaty – Okno potwierdzenia	37

RYSUNEK 23. Testowanie wpłaty 2	38
RYSUNEK 24. Tabela testowa 5	38
RYSUNEK 25. Testowanie przelewu 1	38
RYSUNEK 26. Testowanie przelewu 2	39
RYSUNEK 27. Tabela testowa 5	39
RYSUNEK 28. Testowanie przelewu – Komunikat poprawności	39
RYSUNEK 29. Testowanie przelewu 3	40
RYSUNEK 30. Testowanie przelewu 4	40
RYSUNEK 31. Tabela testowa 7	41
RYSUNEK 32. Testowanie zabezpieczeń – Logowanie	41
RYSUNEK 33. Testowanie zabezpieczeń – Niepoprawna kwota 1	42
RYSUNEK 34. Testowanie zabezpieczeń – Niepoprawna kwota 2	42
RYSUNEK 35. Testowanie zabezpieczeń – Niepoprawna kwota 3	42
RYSUNEK 36. Testowanie zabezpieczeń – Numer konta	43
RYSUNEK 37. Testowanie zabezpieczeń – Rejestracja 1	43
RYSUNEK 38. Testowanie zabezpieczeń – Rejestracja 2	43

1. Wstęp

1.1. Cel projektu

Cel projektu: projekt oraz implementacja bazy danych oraz prostego interfejsu użytkownika przeznaczonych do obsługi z poziomu aplikacji konta bankowego.

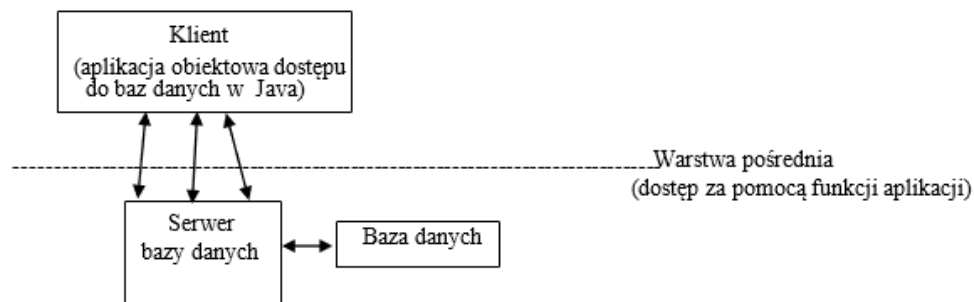
1.2. Zakres projektu

W projekcie będzie zrealizowana graficzna aplikacja użytkownika, umożliwiająca dostęp do konta bankowego, realizująca podstawowe funkcje systemu bankowego, w oparciu o zaprojektowaną zgodnie z wymaganiami relacyjną bazę danych.

2. Analiza wymagań

2.1. Opis działania i schemat logiczny systemu

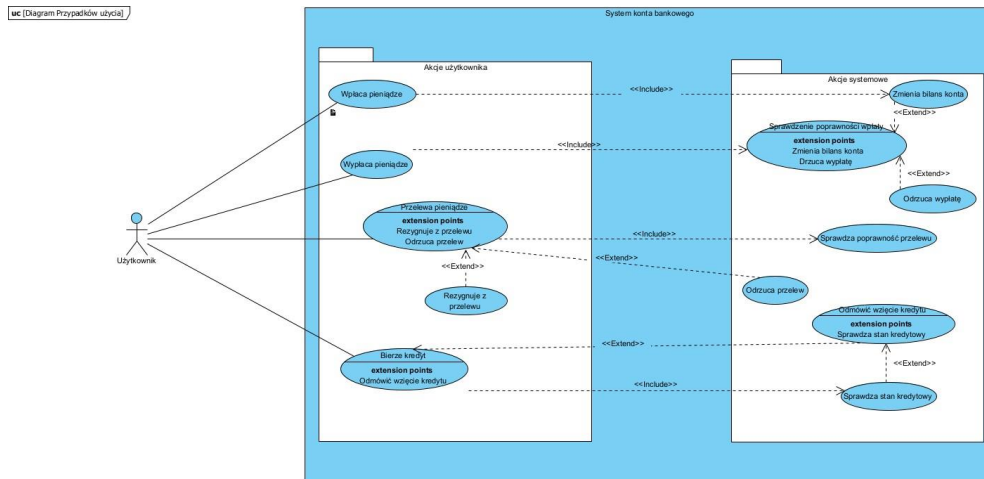
System umożliwiać będzie dostęp do konta bankowego w oparciu o relacyjną bazę danych (tabele opisujące dane o koncie, np. lista transakcji, bilans konta itd.). W szczególności, dla bazy danych o znanej strukturze możliwy będzie dostęp do danych z poziomu aplikacji desktopowej, a także wykonywanie za pomocą tej aplikacji określonych operacji (np. przelew, kredytowanie i inne).



RYSUNEK 1. Schemat komunikacji i struktura systemu

2.2. Wymagania funkcjonalne

2.2.1. Diagram przypadków użycia



RYSUNEK 2. Diagram przypadków użycia

2.2.2. Scenariusze wybranych przypadków użycia

PU Zmiana bilansu konta

Cel: zmiana bilansu konta

Warunki wstępne: Wywołanie z wpłaceniem, wypłaceniem pieniędzy, przelewem lub wzięciem kredytu.

Warunki końcowe: Bilans konta jest powiększony lub pomniejszony o odpowiednią kwotę.

Przebieg:

1. Należy podać ilość pieniędzy, o którą należy zmienić stan konta.
2. Wywołanie sprawdzenia poprawności zmiany bilansu konta
3. W przypadku braku poprawności, wywołanie odmowy zmiany bilansu.

PU Wpłacanie pieniędzy

Cel: zmiana bilansu konta

Warunki wstępne: Podanie ilości pieniędzy do wpłaty przez użytkownika. Warunki końcowe: Bilans konta jest powiększony o odpowiednią kwotę.

Przebieg:

1. Należy podać ilość pieniędzy, o którą należy zwiększyć stan konta.
2. Wywołanie **PU zmiany bilansu konta**.

PU Wypłacanie pieniędzy

Cel: zmiana bilansu konta

Warunki wstępne: Podanie ilości pieniędzy do wypłaty przez użytkownika. Warunki końcowe: Bilans konta jest pomniejszony o odpowiednią kwotę.

Przebieg:

1. Należy podać ilość pieniędzy, o którą należy zmniejszyć stan konta.
2. Wywołanie **PU zmiany bilansu konta**.

PU Przelewanie pieniędzy

Cel: Zmiana bilansu konta przelewającego i odbiorcy.

Warunki wstępne: Podanie kwoty i adresata przez użytkownika.

Warunki końcowe: Bilans konta nadawcy jest pomniejszony o daną kwotę, a odbiorcy jest zwiększony.

Przebieg:

1. Należy podać ilość pieniędzy do przelewu i numer konta odbiorcy.
2. Wywoływanie **PU sprawdzanie poprawności przelewu**.
3. W przypadku braku poprawności, wywoływanie **PU odrzucanie przelewu**.
4. Jeśli przelew jest poprawny, wywoływanie **PU zmiana bilansu konta** u konta nadawcy i konta odbiorcy o określoną kwotę.

PU Odrzucanie przelewu

Cel: Odrzucenie przelewu

Warunki wstępne: Może być wywołany przez **PU przelewanie pieniędzy**. Warunki końcowe: Rezygnacja z procesu przelewania pieniędzy.

Przebieg:

1. Odrzucanie przelewu jest wywołane przez **PU przelewanie pieniędzy** w przypadku braku poprawności przelewu.
2. Zatrzymanie dalszego przebiegu **PU przelewanie pieniędzy**.

PU Sprawdzanie poprawności przelewu

Cel: Sprawdzenie poprawności przelewu

Warunki wstępne: Jest wywoływany przez **PU przelewanie pieniędzy**.

Warunki końcowe: Określenie, czy sugerowany przelew jest poprawny.

1. Sprawdzanie poprawności przelewu jest wywoływane przez **PU przelewanie pieniędzy** w celu określenia poprawności przelewu.
2. Sprawdzanie czy kwota przelewu nie jest większa od dostępnych środków i czy konto odbiorcy istnieje.

PU Rezygnacja z wzięcia przelewu

Cel: Użytkownik rezygnuje z wykonanego przelewu.

Warunki wstępne: Użytkownik podaje id istniejącego przelewu.

Warunki końcowe: Przelew jest cofany.

1. Użytkownik podaje, z którego przelewu chce zrezygnować.
2. Wywołanie **PU przelewanie pieniędzy** przy zamianie nadawcy i odbiorcy.

PU Branie kredytu

Cel: Użytkownik bierze kredyt na konto.

Warunki wstępne: Użytkownik podaje kwotę kredytu.

Warunki końcowe: Zmiana bilansu konta użytkownika o określoną kwotę i zmiana stanu kredytowego.

1. Użytkownik podaje kwotę kredytu.
2. Wywołanie **PU sprawdzanie stanu kredytowego**, aby określić, czy możliwe jest wzięcie kredytu.
3. W przypadku możliwości wzięcia kredytu, wywoływane jest **PU zmiana bilansu konta** i zmiana stanu kredytowego o określoną kwotę.
4. W innym przypadku, wywołanie **PU odmowa wzięcia kredytu**.

PU Odmowa wzięcia kredytu

Cel: System odmawia użytkownikowi wzięcia kredytu.

Warunki wstępne: Może być wywoływane przez **PU branie kredytu**. Warunki końcowe: rezygnacja z procesu wzięcia kredytu.

1. **PU odmowa wzięcia kredytu** jest wywoływana przez **PU branie kredytu**.
2. Zatrzymanie dalszego przebiegu **PU branie kredytu**.

PU Sprawdzanie stanu kredytowego

Cel: System sprawdza stan kredytowy użytkownika.

Warunki wstępne: Jest wywoływane przez **PU branie kredytu**. Warunki końcowe: Określenie możliwości brania kredytu.

1. Jest wywoływane przez **PU branie kredytu** w celu określenia możliwości brania kredytu.

Jeśli kwota kredytu przekracza stan kredytowy, nie pozwala na branie kredytu.

2.3. Wymagania нефункциональные

2.3.1. Wykorzystywane technologie i narzędzia

Baza danych będzie obsługiwana za pośrednictwem serwera bazy danych MySQL [2] oraz serwera lokalnego MySQL [2]. Interfejs użytkownika zostanie zrealizowany w postaci aplikacji obiektowej w języku Java [3] za pomocą technologii JavaFX [4]. Do specyfikacji funkcji systemu wykorzystany zostanie zunifikowany język modelowania UML [1].

2.3.2. Wymagania dotyczące bezpieczeństwa systemu

Dostęp do aplikacji będzie umożliwiony za pomocą loginu i hasła. Baza danych będzie modyfikowalna tylko przez użytkownika Admin oraz przez poszczególne metody aplikacji, niedostępne dla zmiany użytkowników.

2.4. Przyjęte założenia projektowe

W projekcie będzie zrealizowany 3-warstwowy model komunikacji użytkownika i bazy danych za pomocą serwera lokalnego. W zastosowanym modelu częściowe przetwarzanie danych (funkcje biznesowe) jest wykonywane po stronie aplikacji (aplikacja Java [2]), zarządzanie danymi oraz częściowe ich przetwarzanie (np. widoki i inne operacje bazodanowe) są realizowane też po stronie aplikacji. Dostęp do bazy danych realizowany będzie z wykorzystaniem funkcji aplikacji, które komunikują się bezpośrednio z serwerem bazodanowym.

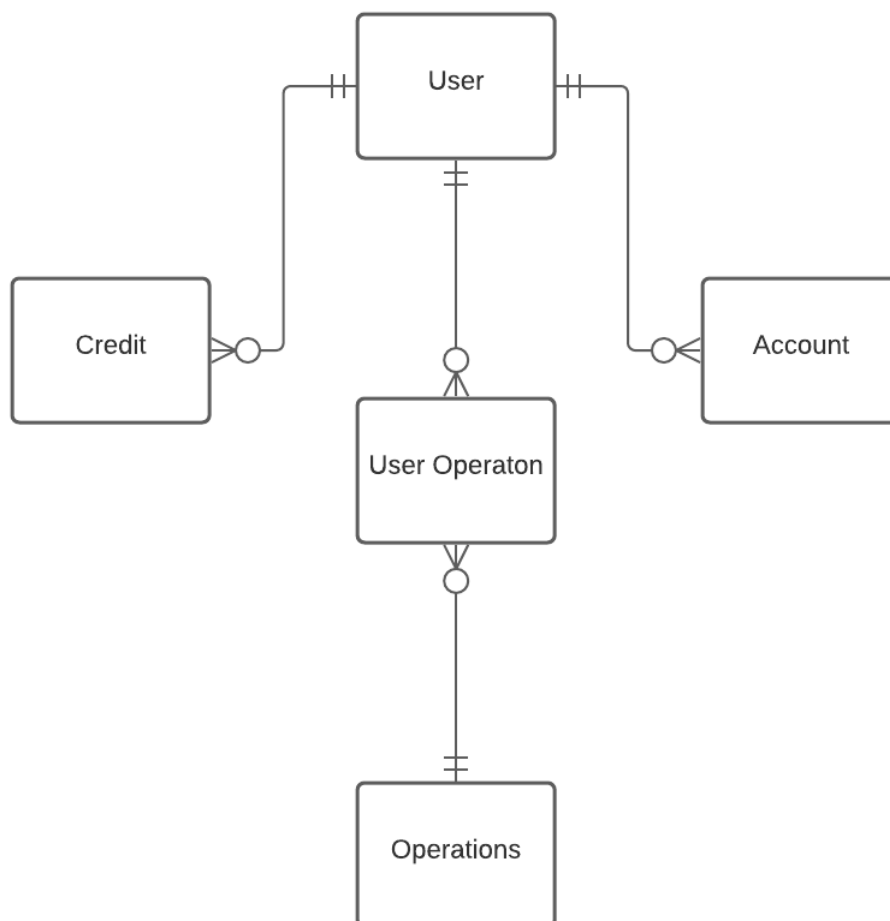
3. Projekt systemu

3.1. Projekt bazy danych

3.1.1. Analiza rzeczywistości i uproszczony model konceptualny

Aplikacja wymaga następujących funkcji: przechowywanie danych konta, danych kredytowych, danych o operacjach kont oraz typów operacji.

Zgodnie z wymaganiami aplikacji należy przyjąć następujący model konceptualny:



RYSUNEK 3. Uproszczony model konceptualny bazy danych

3.1.2. Model logiczny i normalizacja

Założone było przechowywanie następujących danych:

Użytkownik – Login, hasło, numer konta, imię, nazwisko

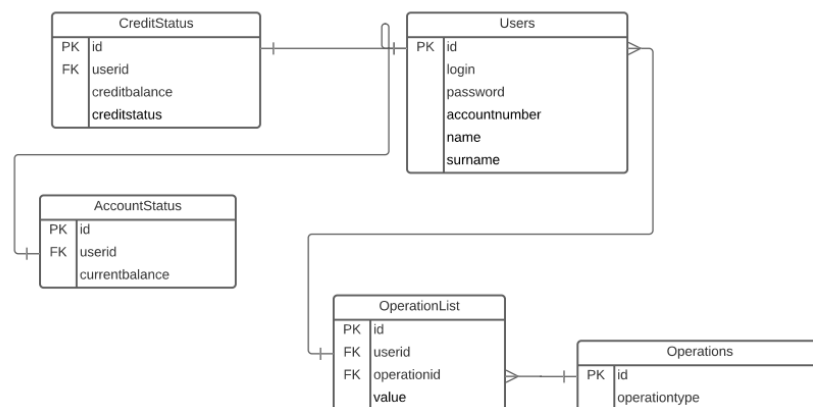
Kredytowanie – Identyfikator użytkownika, stan kredytowy, bilans kredytowy

Status konta - Identyfikator użytkownika, bilans konta

Lista operacji – Typ operacji, identyfikator użytkownika, wartość operacji

Operacji – Nazwa operacji

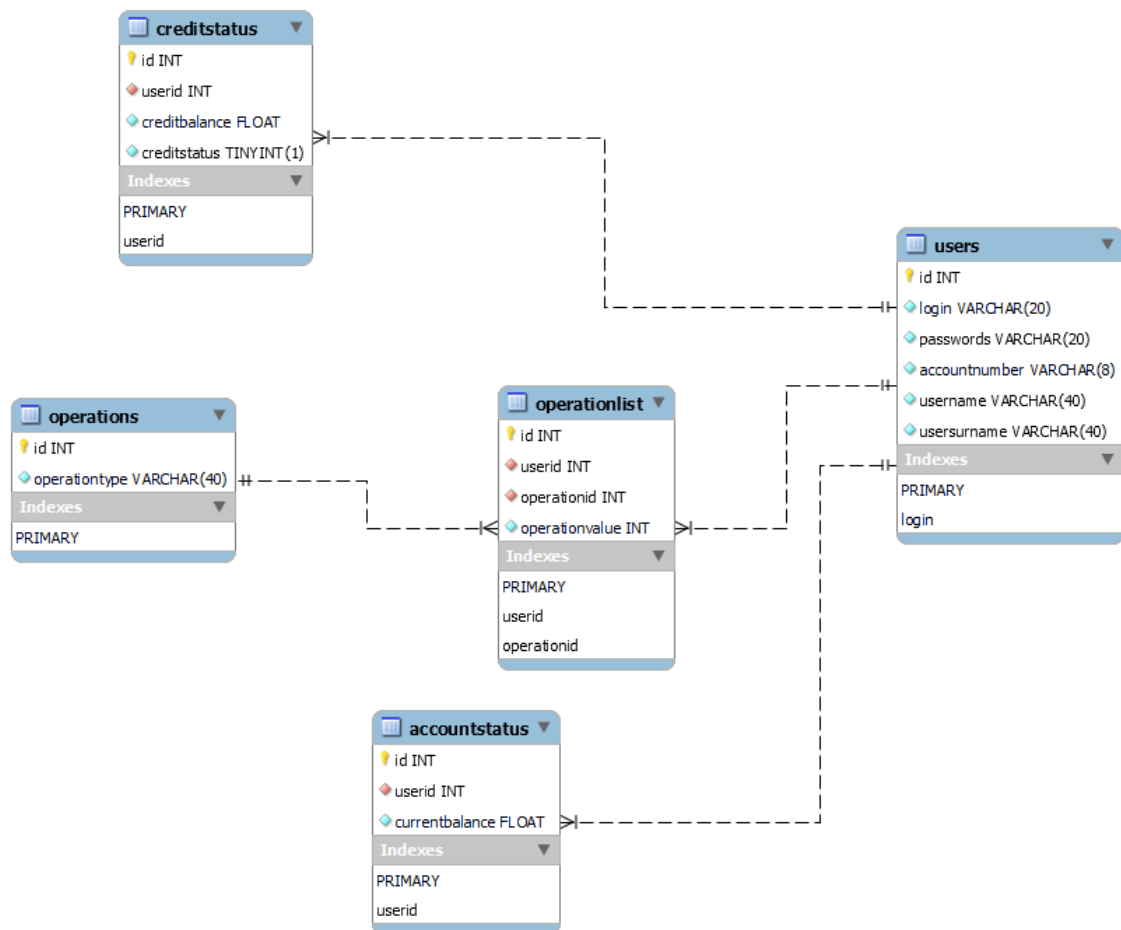
Zgodnie z założeniami projektowymi zostały określone następujące pola bazy danych.



RYSUNEK 3. Model logiczny bazy danych

3.1.3. Model fizyczny i ograniczenia integralności danych

Do realizacji bazy danych stosuję się MySQL. Posiada on określone typy danych, więc został zaprojektowany model fizyczny na podstawie tych typów.



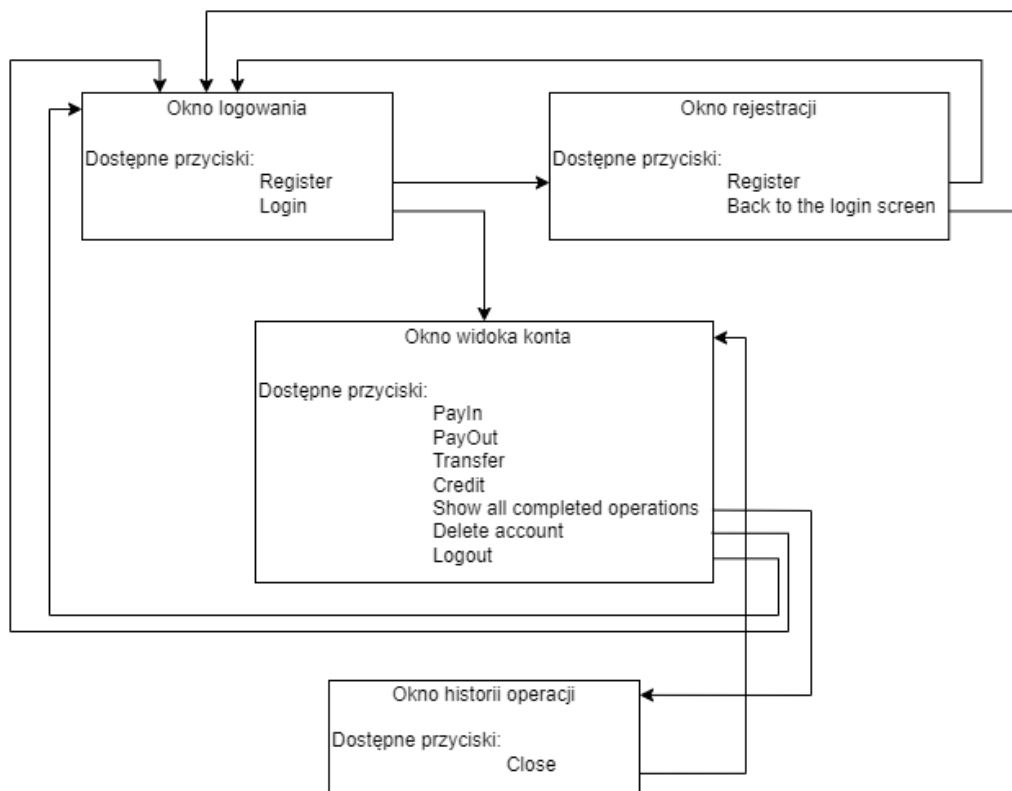
RYSUNEK 5. Model fizyczny bazy danych

3.2.2. Interfejs graficzny i struktura menu

Interfejs graficzny będzie zawierał 4 okna graficzne:

1. Okno logowania (główne okno przy uruchomieniu operacji)
2. Okno rejestracji (okno, za pomocą którego użytkownik może zarejestrować się do systemu)
3. Okno widoku konta (pokazuje użytkownikowi informację o koncie wraz z dostępnymi operacjami)
4. Okno historii operacji (pokazuje użytkownikowi okno z informacją o dokonanych operacjach)

Struktura okien wraz z operacjami przycisków jest przedstawiona modelem:



RYSUNEK 7. Struktura okien aplikacji

3.2.3. Projekt wybranych funkcji systemu

Zgodnie z przypadkami użycia zostały zaprojektowane następujące funkcje:

- Logowanie
- Rejestracja(tworzenie konta)
- Przelew
- Kredytowanie
- Wpłata
- Wypłata
- Udostępnianie wykonanych operacji
- Usuwanie konta

Rejestracja odbywa się z podaniem następujących danych:

- Imię
- Nazwisko
- Login
- Hasło

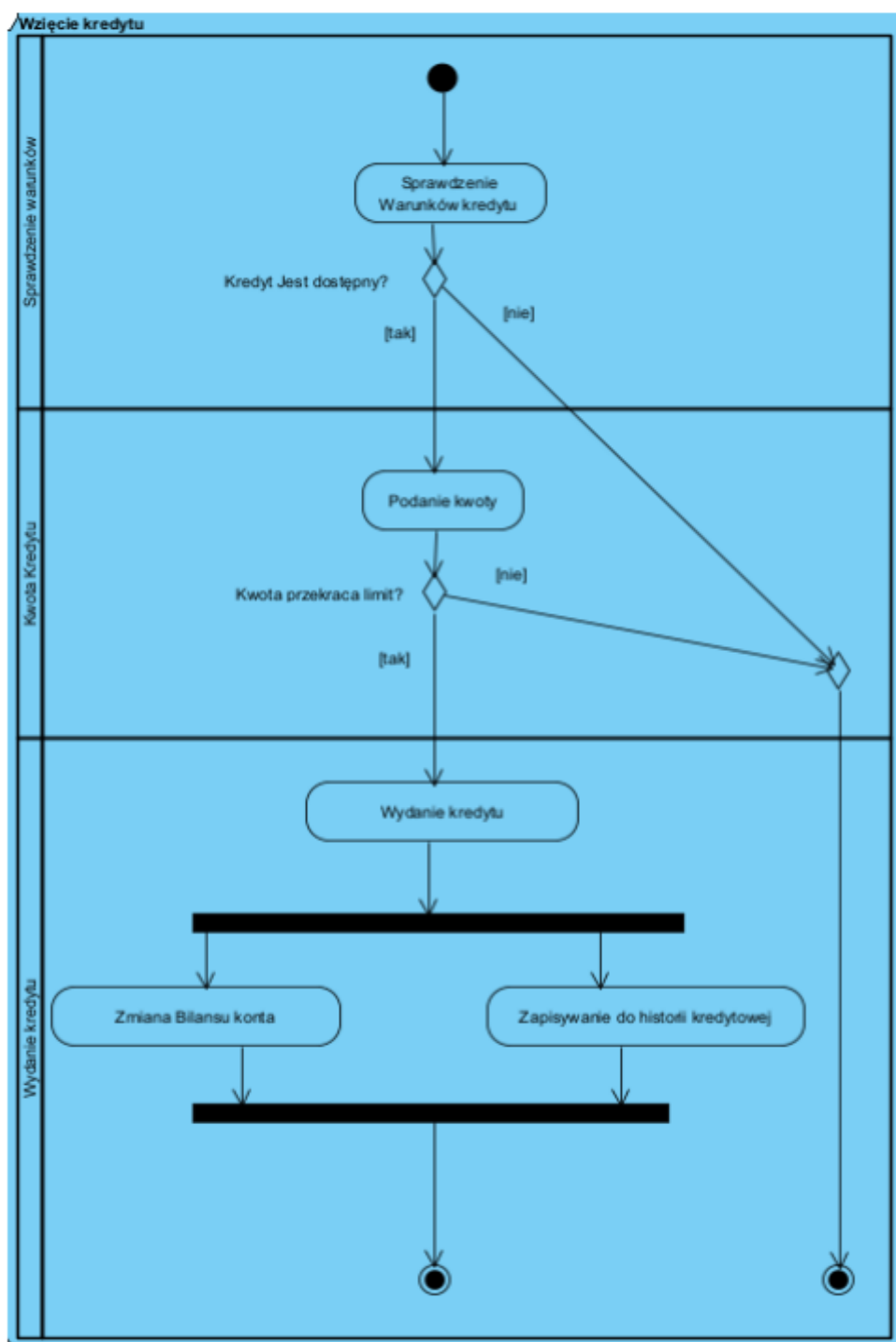
Dla dokonywania wpłaty, wypłaty oraz kredytowania należy podać kwotę, dla dokonywania przelewu – kwotę i numer konta odbiorcy.

Przelew wykonuję się za pomocą numeru konta, jest on unikalny i nadaje się automatycznie przy rejestracji.

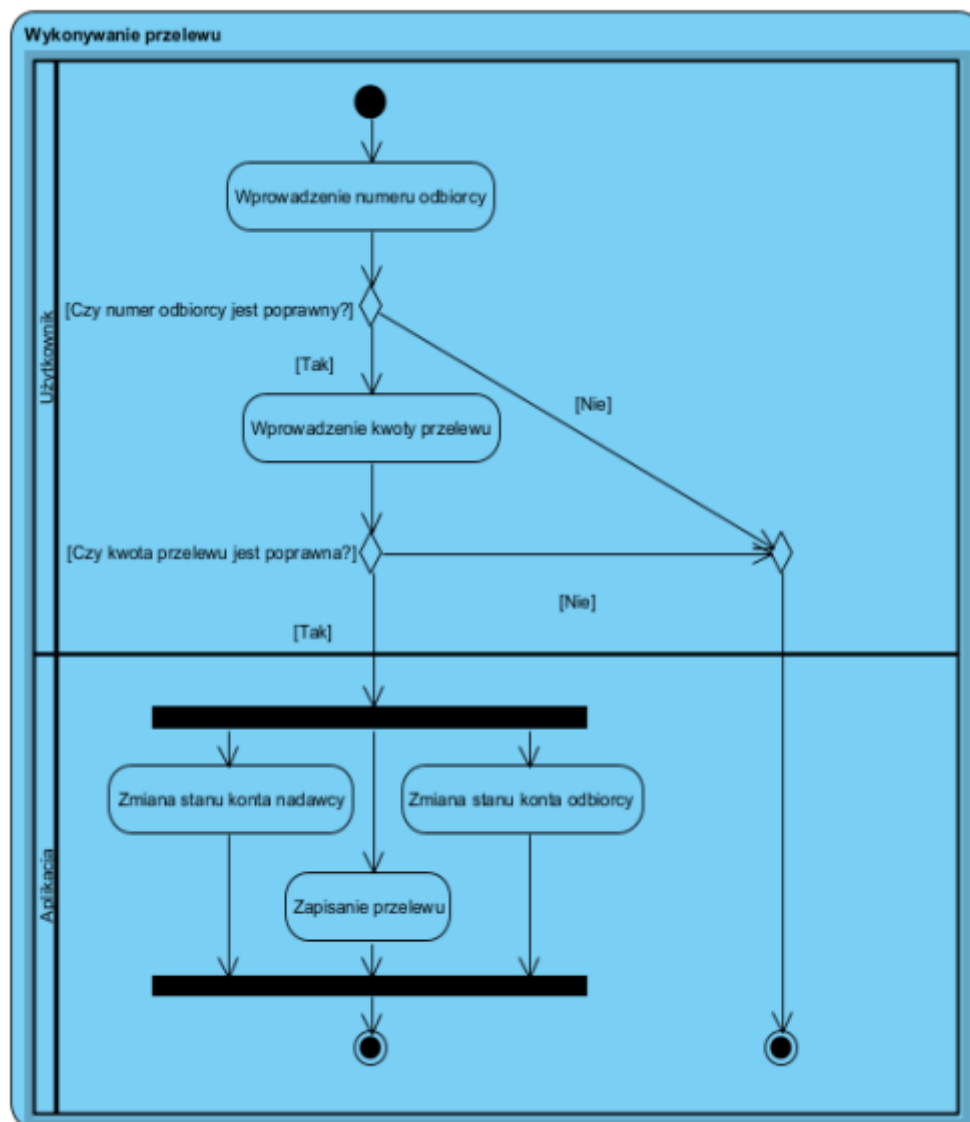
Wykonane operacje zachowują się od początku stworzenia konta, nie ma limitu na pokazywane operacje.

Kredytowanie odbywa się na podstawie bieżącego salda kredytowego, nie wolno wziąć kredytu więcej, niż ustalona kwota. Saldo kredytowe jest aktualizowane w ciągu trwania konta, i zmienia się po wzięciu lub oddaniu kredytu.

Przykładowe diagramy stanów dla operacji kredytowania i przelewu:



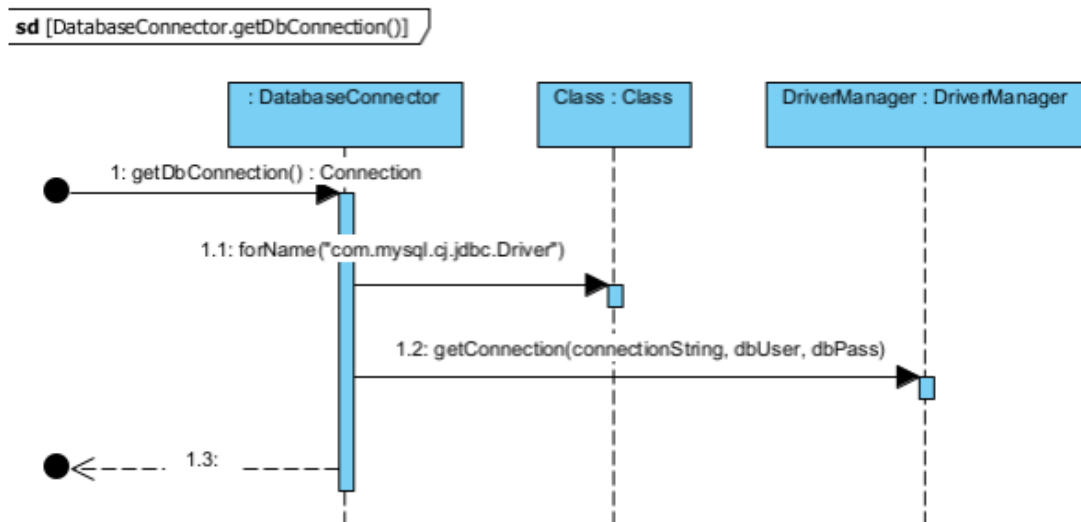
RYSUNEK 8. Diagram stanów - Kredytowanie



RYSUNEK 9. Diagram stanów - Przelew

3.2.4. Metoda podłączania do bazy danych – integracja z bazą danych

Dla podłączenia do bazy danych stosuje się MySQL JDBC Connector, oraz klasa DatabaseConnector. Klasa przechowuje jedną metodę, niezbędną do podłączenia, której diagram sekwencji jest umieszczony poniżej:



RYSUNEK 10. Diagram sekwencji dla metody podłączenia do bazy danych

Po podłączeniu wysyłamy zapytania w postaci tekstowych zapytań SQL z poziomu aplikacji. Klasą, przechowującą i wysyłającą zapytania jest `ModelController`

3.2.5. Projekt zabezpieczeń na poziomie aplikacji

Uwzględniając to, że aplikacja musi zarządzać środkami klientów, przy projektowaniu zabezpieczeń były przyjęte następujące założenia:

Cały program:

- Użytkownik nie może zostawić pustym pole dla wprowadzenia danych

Rejestracja:

- Użytkownik nie może wprowadzić nie tekstowe imię i nazwisko
- Użytkownik nie może podać loginu mniejszego od 6 symboli i hasła mniejszego od 8 symboli

Logowanie:

- Użytkownik nie może zalogować się z błędnym loginem lub hasłem

Przelew:

- Użytkownik musi wprowadzić kwotę przelewu, większą od 0
- Użytkownik musi wprowadzić numer konta odbiorcy posiadający dokładnie 8 cyfr w zakresie od 10000000 do 99999999
- Użytkownik nie może przelać pieniądze na swoje konto
- Użytkownik nie może przelać pieniądze na nie istniejące konto

Wpłata i wypłata:

- Użytkownik musi wprowadzić kwotę, większą od 0
- Użytkownik musi wprowadzić kwotę przelewu, większą od 0
- Użytkownik nie może wypłacić więcej środków, niż posiada na obecny moment

Kredytowanie:

- Użytkownik musi wprowadzić kwotę kredytu, większą od 0
- Użytkownik nie może wziąć kredyt na sumę, większą od obecnego stanu kredytowego
- Użytkownik nie może wziąć kredyt, jak jego status kredytowy mu to nie pozwala

Usuwanie konta:

- Użytkownik musi zatwierdzić usuwanie konta

4. Implementacja systemu

4.1. Realizacja bazy danych

Baza danych została zrealizowana za pomocą MySQL i środowiska MySQL Workbench 8.0. Dostęp do bazy danych przez i operacjach na niej posiada tylko administrator bazy. Baza danych jest zabezpieczona hasłem.

4.1.1. Tworzenie tabel i definiowanie ograniczeń

Na podstawie zaprojektowanych diagramów były stworzone 5 tabel, przechowujących dane. Wszystkie tabeli posiadają ograniczenie NOT NULL. Pola **id** są automatycznie inkrementowane (AUTO_INCREMENT), bo są to główne klucze tabel. Wszystkie klucze **Foreign Key** są automatycznie odnawiane przy aktualizacji danych w powiązanych tabelach i zabraniają usuwanie z powiązanych tabel, jak jest jeszcze informacja w tabeli, na którą wskazuje klucz. Pole **login** posiada identyfikator UNIQUE, bo login jest unikalny do każdego użytkownika.

```
create table Users(  
  id int auto_increment not null primary key,  
  login varchar (20) not null unique,  
  passwords varchar(20) not null,  
  accountnumber varchar(8) not null,  
  username varchar(40) not null,  
  usersurname varchar(40) not null  
);  
  
create table CreditStatus(  
  id int primary key not null auto_increment,  
  userid int not null,  
  creditbalance float not null,  
  creditstatus boolean not null,  
  foreign key (userid) references Users (id) on update cascade on delete cascade  
);  
  
create table AccountStatus(  
  id int auto_increment not null primary key,  
  userid int not null,  
  currentbalance float not null,  
  foreign key (userid) references Users(id) on update cascade on delete cascade  
);
```

```

create table Operations(
id int not null primary key auto_increment,
operationtype varchar(40) not null
);

create table OperationList(
id int primary key not null auto_increment,
userid int not null,
operationid int not null,
operationvalue int not null,
foreign key (userid) references Users(id) on update cascade on delete cascade,
foreign key (operationid) references Operations(id) on update cascade on delete restrict
);

```

4.2. Realizacja elementów aplikacji

4.2.1. Obsługa menu

Obsługa menu jest realizowana za pomocą klas ApplicationRegisterController, ApplicationOperationsController, ApplicationLoginController, ApplicationAccountController, dziedziczących po klasie ApplicationRootController. Klasy te są kontrolerami odpowiednich warstw graficznych aplikacji.

Kod klasy ApplicationRootController:

```

package com.bankaccount.applicationview;

import com.bankaccount.applicationsource.ModelController;
import javafx.event.Event;
import javafx.fxml.FXMLLoader;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.io.IOException;
import java.util.Locale;
import java.util.Objects;

public class ApplicationRootController {

    public ModelController initializeController() {
        return new ModelController();
    }
}

```

```

        public void openNewScene(Event event, Object o, String path) {
            FXMLLoader loader = new FXMLLoader();

            loader.setLocation(Objects.requireNonNull(o.getClass().getResource(path)));

            try {
                loader.load();
            } catch (IOException e) {
                e.printStackTrace();
            }

            Parent root = loader.getRoot();
            Stage stage = new Stage();
            stage.setScene(new Scene(root));
            stage.getIcons().add(new Image("https://icons.iconarchive.com/icons/dooffy/characters/256/And-icon.png"));
            stage.setTitle("Bank Application");
            stage.show();
            Node node = (Node) event.getSource();
            Stage prevStage = (Stage) node.getScene().getWindow();
            prevStage.close();
        }

        public void createAlertBox(String type, String text) {
            String typeOf = type.toUpperCase(Locale.ENGLISH);
            Alert alert = new Alert(Alert.AlertType.valueOf(typeOf));
            alert.setContentText(text);
            Button closeButton = new Button();
            closeButton.setOnAction(actionEvent -> {
                alert.close();
            });
            Stage stage = (Stage) alert.getDialogPane().getScene().getWindow();
            stage.getIcons().add(new Image("https://icons.iconarchive.com/icons/dooffy/characters/256/And-icon.png"));
            alert.show();
        }
    }
}

```

Są tu trzy ogólne metody initializeController, createAlertBox i openNewScene.

Metoda initializeController zwraca nowy obiekt klasy ModelController, która zarządza połączeniem z bazą danych.

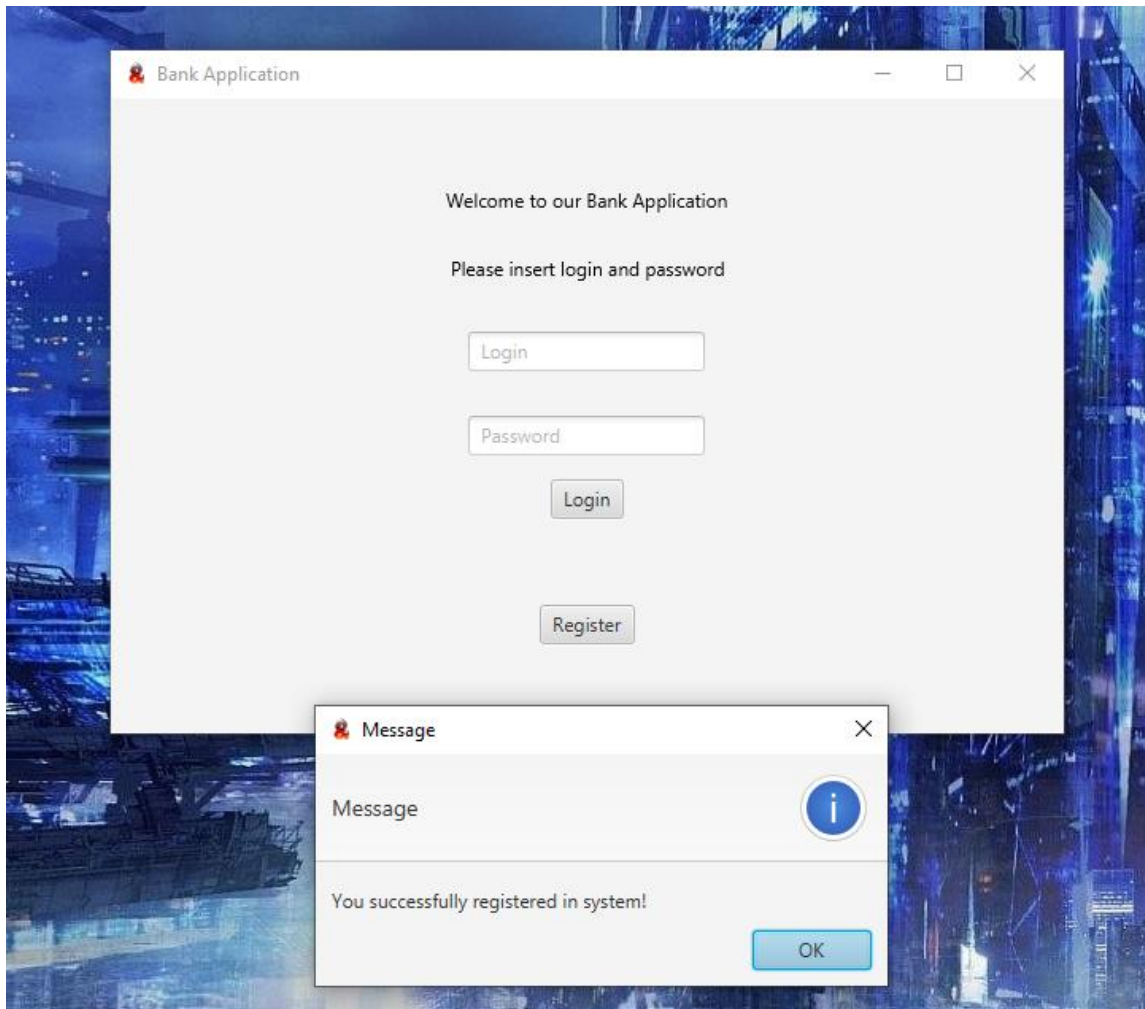
Metoda createAlertBox tworzy nowe okienko z informacją dla użytkownika, zawiera w sobie 3 typy informacji: okienko błędu, okienko informacyjne i okienko potwierdzenia.

Metoda `openNewScene` tworzy nową warstwę aplikacji inicjalizując odpowiednią warstwę i jej kontroler.

Przykład tworzenia nowej warstwy wraz z kontrolerem oraz okienka informacyjnego:

```
if (initializeController().addNewUser(nameText, surnameText, loginText, passwordText)) {  
    openNewScene(event, this, "application-login-screen.fxml");  
    createAlertBox("information", "You successfully registered in system!");  
}
```

W wyniku dostajemy nowe okno aplikacji i powiadomienie o poprawnym logowaniu w systemie:



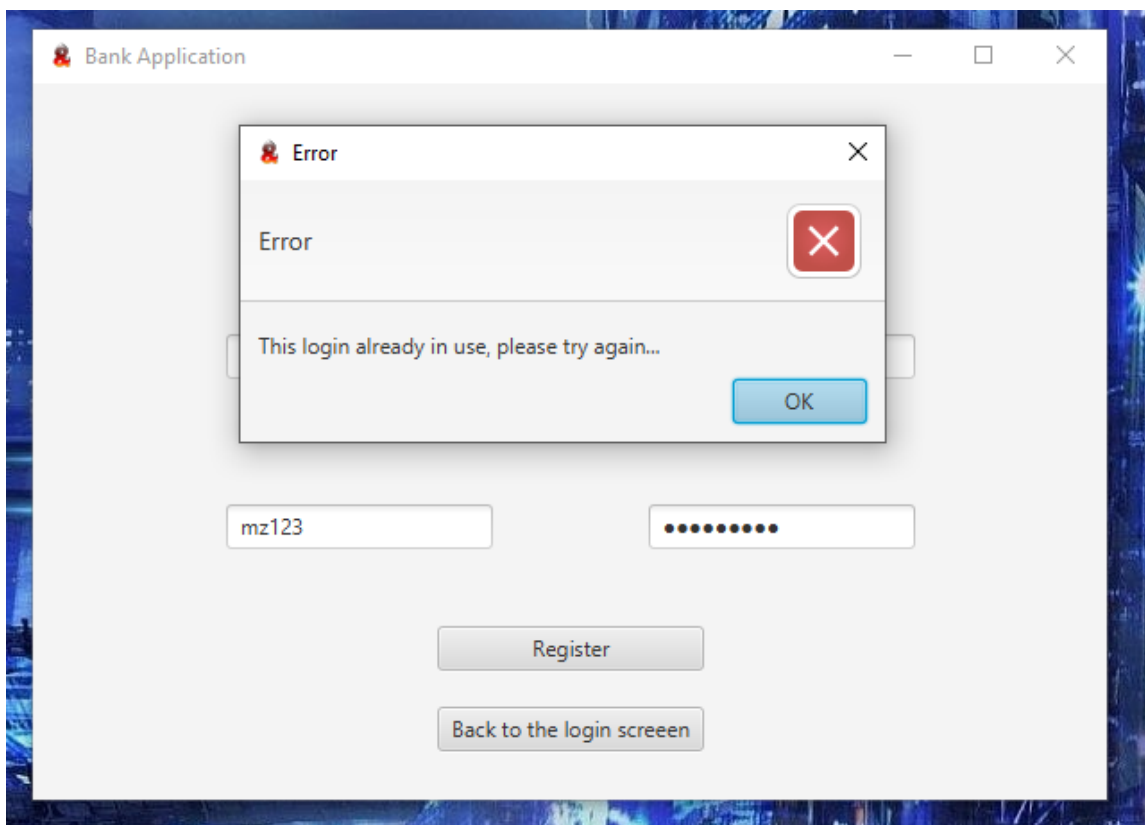
RYSUNEK 11. Rejestracja użytkownika

4.2.2. Walidacja i filtracja

Wszystkie dane zostały przefiltrowane na poziomie aplikacji, na przykład przy rejestracji nie możemy użyć loginu, który już jest w systemie. To zostało zaimplementowane w sposób, podany poniżej:

```
String checkLogin = "select exists (select login from users where login  
= ' " + login + " ');";  
PreparedStatement preparedStatement =  
dbConnector.getConnection().prepareStatement(checkLogin);  
ResultSet rsLogin = preparedStatement.executeQuery();  
if (rsLogin.next()) {  
    if (rsLogin.getInt(1) == 1) {  
        return false;  
    }  
}
```

Tworzymy zapytanie do tabeli, przechowującej dane o wszystkich loginach, i, jeżeli login już istnieje, odrzucamy zapytanie o tworzenie konta z odpowiednią informacją:



RYSUNEK 12. Błąd rejestracji 1

4.2.3. Implementacja interfejsu dostępu do bazy danych

Za dostęp do bazy danych odpowiadają dwie klasy: ModelController oraz DatabaseConnector. Klasa ModelController jest klasą, przechowującą zapytania SQL, DatabaseController odpowiada za bezpośrednie podłączenie do bazy danych.

Kod klasy DatabaseConnector:

```
package com.bankaccount.application;

import java.sql.*;

public class DatabaseConnector implements Config {
    static Connection dbConnection;

    public Connection getDbConnection() throws ClassNotFoundException,
        SQLException {
        String connectionString = "jdbc:mysql://" + dbHost + ":" + dbPort
+ "/" + dbUser + "/" + dbPass;
        Class.forName("com.mysql.cj.jdbc.Driver");
        dbConnection = DriverManager.getConnection(connectionString,
        dbUser, dbPass);
        return dbConnection;
    }
}
```

W tej klasie używamy JDBCConnector do podłączenia do bazy danych, wszystkie dane do podłączenia przechowują się w interfejsie Config.

Klasa ModelController przechowuje wszystkie zapytania do bazy danych.

Przykładowy kod metody, wysyłającej zapytania do bazy danych:

```
public boolean addNewUser(String userName, String userSurname, String
login, String password) throws SQLException, ClassNotFoundException {

    try {
        //checking is login is unique
        String checkLogin = "select exists (select login from users where
login = " + login + " );";
        PreparedStatement preparedStatement =
        dbConnector.getDbConnection().prepareStatement(checkLogin);
        ResultSet rsLogin = preparedStatement.executeQuery();
        if (rsLogin.next()) {
            if (rsLogin.getInt(1) == 1) {
                return false;
            } else {
                //inserting to table users
                String insertUsers = "insert into users (login,
passwords, accountnumber, username, usersurname) values (?, ?, ?, ?, ?);";
                try {
                    int number = (int) (Math.random() * 899999999) +
100000000;
                    PreparedStatement preparedStatementUsers =
                    dbConnector.getDbConnection().prepareStatement(insertUsers);
                    preparedStatementUsers.setString(1, login);
                    preparedStatementUsers.setString(2, password);
                }
            }
        }
    }
}
```

```

        preparedStatementUsers.setString(3, "" + number);
        preparedStatementUsers.setString(4, username);
        preparedStatementUsers.setString(5, userSurname);
        preparedStatementUsers.executeUpdate();
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }
    String getUserID = "select id from users where login = "
+ login + ";";
    ResultSet resultSet = null;
    try {
        PreparedStatement preparedStatementUserID =
dbConnector.getConnection().prepareStatement(getUserID);
        resultSet = preparedStatementUserID.executeQuery();
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }

    int id = 0;

    if (resultSet.next()) {
        id = resultSet.getInt(1);
    }
    //inserting to table credit status
    String insertCreditStatus = "insert into
creditstatus(userid, creditbalance, creditstatus) values (?, ?, ?);";

    try {
        PreparedStatement preparedStatementCreditStatus =
dbConnector.getConnection().prepareStatement(insertCreditStatus);

        preparedStatementCreditStatus.setInt(1, id);
        preparedStatementCreditStatus.setFloat(2, 0);
        preparedStatementCreditStatus.setBoolean(3, true);
        preparedStatementCreditStatus.executeUpdate();
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }

    //inserting to table account status
    String insertAccountStatus = "insert into
accountstatus(userid, currentbalance) values (?, ?);";
    try {
        PreparedStatement preparedStatementAccountStatus =
dbConnector.getConnection().prepareStatement(insertAccountStatus);
        preparedStatementAccountStatus.setInt(1, id);
        preparedStatementAccountStatus.setFloat(2, 0);
        preparedStatementAccountStatus.executeUpdate();
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }
    resultSet.close();
    return true;
}
} catch (SQLException | ClassNotFoundException e) {
    e.printStackTrace();
}
return false;
}

```

Metoda ta dodaje nowego użytkownika, i jest zrealizowana za pomocą klasy PreparedStatement, w której Instrukcja SQL jest prekompilowana i przechowywana w obiekcie PreparedStatement. Ten obiekt może być następnie użyty do wydajnego wielokrotnego wykonania tej instrukcji.

4.2.4. Implementacja wybranych funkcjonalności systemu

W systemie zostały zaimplementowane następujące funkcje:

- Rejestracja
- Logowanie
- Operacje użytkownika (np. Kredytowanie, Przelew itd.)
- Usuwanie konta użytkownika

Poniżej jest przedstawiony przykładowy kod dwóch zaimplementowanych funkcji:

Kod procesu rejestracji:

```
void initialize() {
    registerButton.setOnAction(event -> {
        String loginText = login.getText().trim();
        String passwordText = password.getText().trim();
        String nameText = nameField.getText().trim();
        String surnameText = surnameField.getText().trim();

        if (!loginText.equals("") || !passwordText.equals("") ||
            !nameText.equals("") || !surnameText.equals("")) {
            if (!NumberUtils.isCreatable(nameText) ||
                !NumberUtils.isCreatable(surnameText)) {
                if (nameText.length() < 20 || surnameText.length() < 20)
                {
                    if (loginText.length() > 6 || passwordText.length()
                        > 8)
                    {
                        try
                        {
                            if
                                (initializeController().addNewUser(nameText, surnameText, loginText,
                                passwordText))
                                {
                                    openNewScene(event, this, "application-
                                    login-screen.fxml");
                                    createAlertBox("information", "You
                                    successfully registered in system!");
                                } else createAlertBox("error", "This login
                                    already in use, please try again...");
                                } catch (SQLException | ClassNotFoundException
                                    e)
                                {
                                    e.printStackTrace();
                                }
                            } else createAlertBox("Error", "Cant create login
                                    lower than 6 symbols and password lower than 8 symbols");
                                }else createAlertBox("Error", "Name and surname cant be
                                    bigger than 20 symbols");
                                }else createAlertBox("Error", "Incorrect name or surname,
                                    please retry");
                                }
                        }
                    }
                }
            }
        }
    }
}
```

```

    } else createAlertBox("Error", "Some field is empty, please
retry");

});

```

Kod procesu logowania:

```

void initialize()
{

    loginButton.setOnAction(event ->
    {

        String loginText = loginField.getText().trim();
        String passwordText = passwordField.getText().trim();

        if (!loginText.equals("") || !passwordText.equals("")) {
            if (loginText.length() < 20 || passwordText.length() < 20) {
                try
                {
                    if (!initializeController().findAccount(loginText,
passwordText))
                    {
                        createAlertBox("Error", "Login or password is
incorrect, or login is does not exist!");
                    }
                    else
                    {
                        FXMLLoader loader = new FXMLLoader();

                        loader.setLocation(Objects.requireNonNull(getClass().getResource("appli
cation-account.fxml")));
                        try
                        {
                            loader.load();
                        }
                        catch (IOException e)
                        {
                            e.printStackTrace();
                        }
                        Parent root = loader.getRoot();
                        Stage stage = new Stage();
                        stage.setScene(new Scene(root));

                        //initializing account values
                        ApplicationAccountController controller =
loader.getController();
                        controller.setLogin(loginText);
                        controller.accountInitialize(loginText);

                        stage.getIcons().add(new
Image("https://icons.iconarchive.com/icons/dooffy/characters/256/And-
icon.png"));

                        stage.show();
                        Node node = (Node) event.getSource();
                        Stage prevStage = (Stage)
node.getScene().getWindow();
                        prevStage.close();
                    }
                }
            }
            catch (SQLException | ClassNotFoundException e)
            {
                e.printStackTrace();
            }
        }
        else createAlertBox("Error", "Login or password cant be
bigger than 20 symbols!");
    } else createAlertBox("Error", "Login or password field cant be
empty!");
    }
}

```

```
});

registerButton.setOnAction(event -> {
    registerButton.getScene().getWindow().hide();
    openNewScene(event, this, "application-register-screen.fxml");
});
```

4.2.5. Implementacja mechanizmów bezpieczeństwa

Wszystkie metody, w których jest interakcja z użytkownikiem, zostały zabezpieczone od niepoprawnego wprowadzenia danych. Na przykład w przypadku logowania na błędny login, lub z użyciem błędnego hasła, operacja jest odrzucana przez system.

Przykładowy kod procesu:

```
if (!initializeController().findAccount(loginText, passwordText)) {
    createAlertBox("Error", "Login or password is incorrect, or login is
does not exist!");
}
```

```
public Boolean findAccount(String login, String password) throws
SQLException, ClassNotFoundException {
    try {
        //watching is login exist in database
        String checkLogin = "select exists (select login from users where
login = " + login + "';";
        PreparedStatement preparedStatementLogin =
dbConnector.getConnection().prepareStatement(checkLogin);
        ResultSet rsLogin = preparedStatementLogin.executeQuery();
        if (rsLogin.next()) {
            if (rsLogin.getInt(1) == 0) {
                return false;
            }
            // watching password is correct
        } else {
            String checkPassword = "select passwords from users where
login = " + login + "';";
            PreparedStatement preparedStatementPassword =
dbConnector.getConnection().prepareStatement(checkPassword);
            ResultSet rsPassword =
preparedStatementPassword.executeQuery();
            if (rsPassword.next()) {
                if (!rsPassword.getString(1).equals(password)) {
                    return false;
                } else {
                    return true;
                }
            }
            rsPassword.close();
        }
        rsLogin.close();
    } catch (SQLException | ClassNotFoundException e) {
        e.printStackTrace();
    }
    return false;
}
```

5. Testowanie systemu

5.1. Testowanie opracowanych funkcji systemu

Przetestowane zostały następujące funkcje systemu:

- Tworzenie konta
- Usuwanie konta
- Przelew
- Wpłata
- Wypłata

5.1.1. Testowanie tworzenia konta

Na początku tabela przechowująca dane o użytkownikach wygląda w następujący sposób:

	id	login	passwords	accountnumber	username	usersurname
▶	1	mz123	mz123	35603377	Maksim	Zakharau
	2	jk123	jk123	53177379	Jan	Kowalski
	3	login123	qazsew123	51315745	Name	Surname
	4	login1111	qazsew123	53768029	Name	Surname
*	NULL	NULL	NULL	NULL	NULL	NULL

RYSUNEK 13. Tabela testowa 1

Spróbujemy wprowadzić następujące dane w aplikacji:

Login: nowyuztytkownik

Hasło: nowyuztytkownik

Imię: Nowy

Nazwisko: Użytkownik

Bank Application

Registration

Please insert required information

Nowy Użytkownik

nowyuzytownik

Register

Back to the login screen

RYSUNEK 14. Test rejestracji 2

Po wciśnięciu przycisku Register dane są dodane i można zauważyć, że teraz one są w tabeli Users.

	id	login	passwords	accountnumber	username	usersurname
▶	1	mz123	mz123	35603377	Maksim	Zakharau
	2	jk123	jk123	53177379	Jan	Kowalski
	3	login123	qazsew123	51315745	Name	Surname
	4	login1111	qazsew123	53768029	Name	Surname
	5	nowyuzytownik	nowyuzytownik	27433688	Nowy	Użytkownik
*	NULL	NULL	NULL	NULL	NULL	NULL

RYSUNEK 15. Tabela testowa 2

5.1.2. Testowanie usuwania konta

Zalogujemy się na konto za pomocą aplikacji:

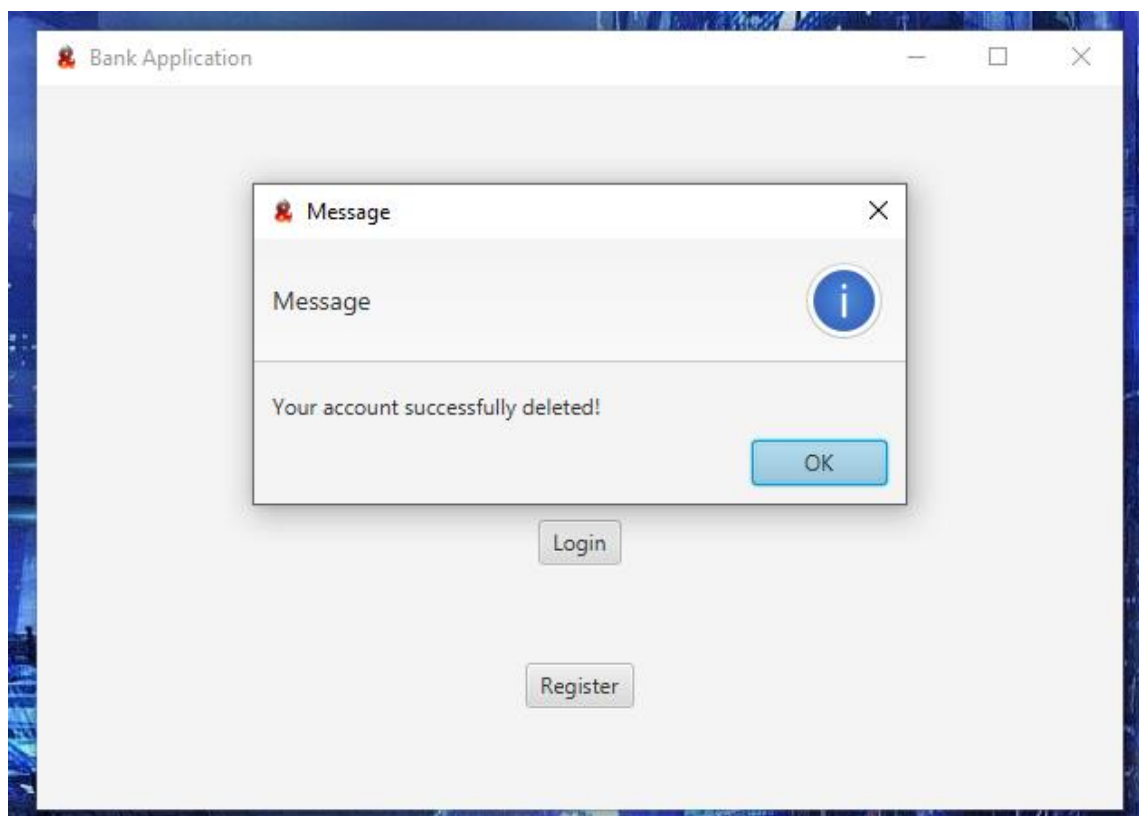
The screenshot shows a web application window with two main sections: 'Your account info:' and 'Available operations:'. The 'Your account info:' section contains several input fields with the following values: Login: nowyuzytownik, Name: Nowy, Surname: Użytkownik, Account number: 27433688, Current balance: 0.0, Credit balance: 0.0, and Credit available for next value: 10000.0. The 'Available operations:' section contains buttons for PayIn, PayOut, Transfer, Credit, Show all completed operations, Delete account, and Logout.

RYSUNEK 16. Test usuwania 1

Tabela z użytkownikami wygląda w następujący sposób:

	id	login	passwords	accountnumber	username	usersurname
▶	1	mz123	mz123	35603377	Maksim	Zakharau
	2	jk123	jk123	53177379	Jan	Kowalski
	3	login123	qazsew123	51315745	Name	Surname
	4	login1111	qazsew123	53768029	Name	Surname
	5	nowyuzytownik	nowyuzytownik	27433688	Nowy	Użytkownik
•	NULL	NULL	NULL	NULL	NULL	NULL

Po wciśnięciu przycisku Delete Account widzimy następujący komunikat, sygnalizujący o poprawnym usunięciu konta:



RYSUNEK 17. Test usuwania 2

W bazie danych tego użytkownika też już nie istnieje:

	id	login	passwords	accountnumber	username	usersurname
▶	1	mz123	mz123	35603377	Maksim	Zakharau
	2	jk123	jk123	53177379	Jan	Kowalski
	3	login123	qazsew123	51315745	Name	Surname
	4	login1111	qazsew123	53768029	Name	Surname
*	NULL	NULL	NULL	NULL	NULL	NULL

RYSUNEK 18. Tabela testowa 3

5.1.3. Testowanie wpłaty

Zalogujemy się na istniejącego użytkownika:

The screenshot shows a window titled 'Your account info:' and 'Available operations:'. The 'Your account info:' section contains several text boxes with the following values: Login: mz123, Name: Maksim, Surname: Zakharau, Account number: 35603377, Current balance: 400.0, Credit balance: 0.0, and Credit available for next value: 10000.0. The 'Available operations:' section contains buttons for PayIn, PayOut, Transfer, Credit, Show all completed operations, Delete account, and Logout.

RYSUNEK 19. Testowanie wpłaty 1

	id	login	passwords	accountnumber	username	usersurname	id	userid	currentbalance
▶	1	mz123	mz123	35603377	Maksim	Zakharau	1	1	400
	2	jk123	jk123	53177379	Jan	Kowalski	2	2	150
	3	login123	qazsew123	51315745	Name	Surname	3	3	0
	4	login1111	qazsew123	53768029	Name	Surname	4	4	0

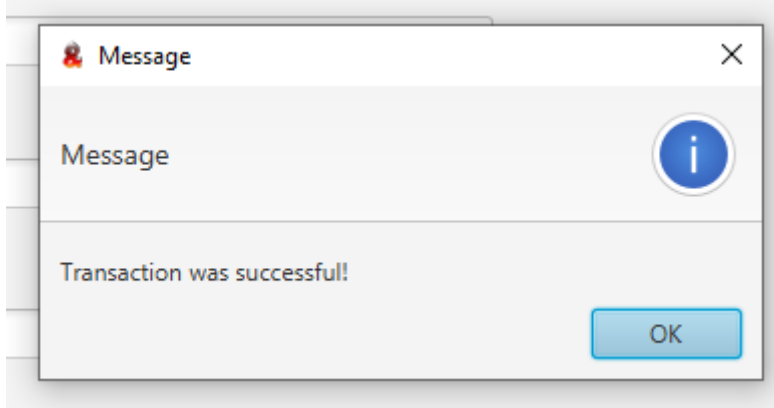
RYSUNEK 20. Tabela testowa 4

Po wciśnięciu przycisku PayIn otworzy się okno wpłaty:

The screenshot shows a dialog box titled 'Payment transaction' with a close button (X). The text inside says 'Please enter value of transaction' next to a question mark icon. Below this is a text input field. At the bottom are 'OK' and 'Cancel' buttons.

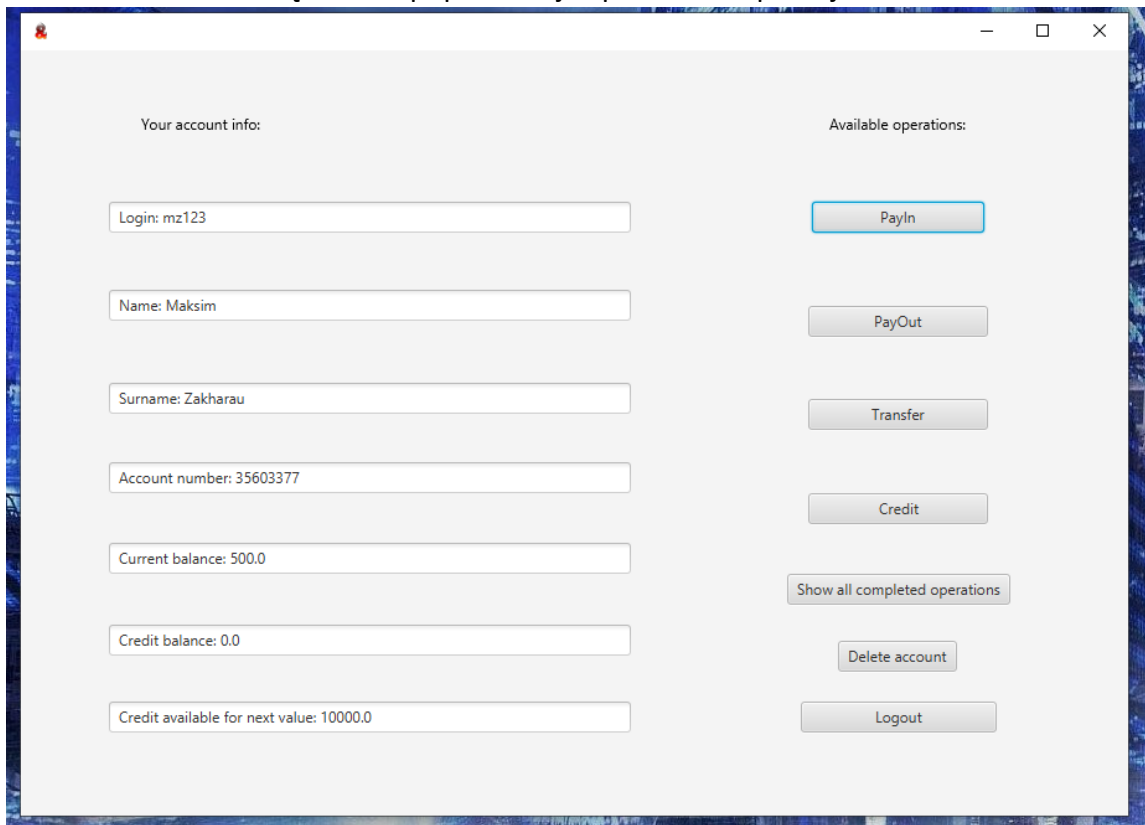
RYSUNEK 21. Testowanie wpłaty - Okno danych

Wprowadzimy kwotę 100 i wyniku dostajemy powiadomienie o poprawnej wpłacie:



RYSUNEK 22. Testowanie wpłaty – Okno potwierdzenia

Jest widać że bilans się zmienił poprawnie jak po stronie aplikacji:



RYSUNEK 23. Testowanie wpłaty 2

Tak i po stronie bazy danych:

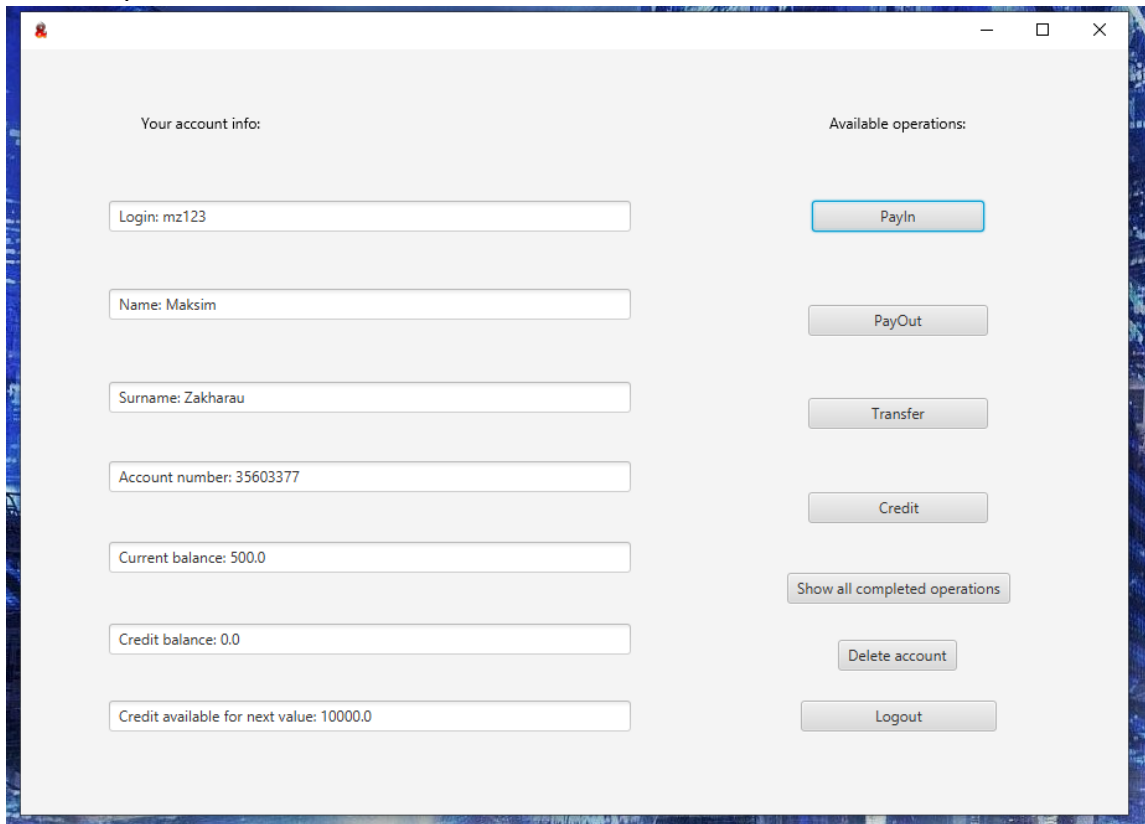
	id	login	passwords	accountnumber	username	usersurname	id	userid	currentbalance
▶	1	mz123	mz123	35603377	Maksim	Zakharau	1	1	500
	2	jk123	jk123	53177379	Jan	Kowalski	2	2	150
	3	login123	qazsew123	51315745	Name	Surname	3	3	0
	4	login1111	qazsew123	53768029	Name	Surname	4	4	0

RYSUNEK 24. Tabela testowa 5

5.1.4. Testowanie przelewu

Z zalogowanym użytkownikiem **mz123** spróbujemy wykonać operacje przelewania środków na inne konto (użytkownik **jk123**)

Konto użytkownika mz123:

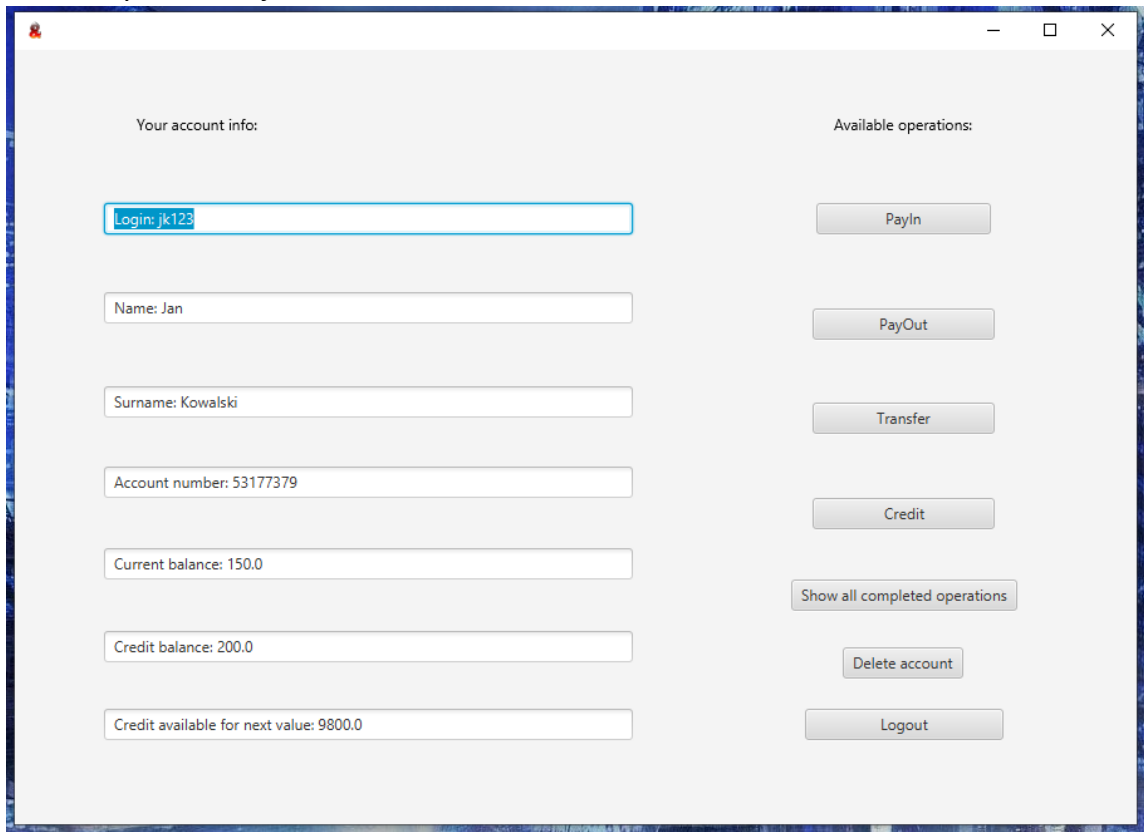


The screenshot displays a web application window with a light gray background. It is divided into two main sections: 'Your account info:' on the left and 'Available operations:' on the right. The 'Your account info:' section contains six text input fields, each with a label and a value: 'Login: mz123', 'Name: Maksim', 'Surname: Zakharau', 'Account number: 35603377', 'Current balance: 500.0', and 'Credit balance: 0.0'. Below these is a field for 'Credit available for next value: 10000.0'. The 'Available operations:' section contains five buttons: 'PayIn' (highlighted with a blue border), 'PayOut', 'Transfer', 'Credit', and 'Show all completed operations'. At the bottom of this section are two more buttons: 'Delete account' and 'Logout'.

Section	Field/Label	Value
Your account info:	Login:	mz123
	Name:	Maksim
	Surname:	Zakharau
	Account number:	35603377
	Current balance:	500.0
	Credit balance:	0.0
Credit available for next value:		10000.0
Available operations:	PayIn	Button
	PayOut	Button
	Transfer	Button
	Credit	Button
	Show all completed operations	Button
Delete account		Button
Logout		Button

RYSUNEK 25. Testowanie przelewu 1

Konto użytkownika jk123:



RYSUNEK 26. Testowanie przelewu 2

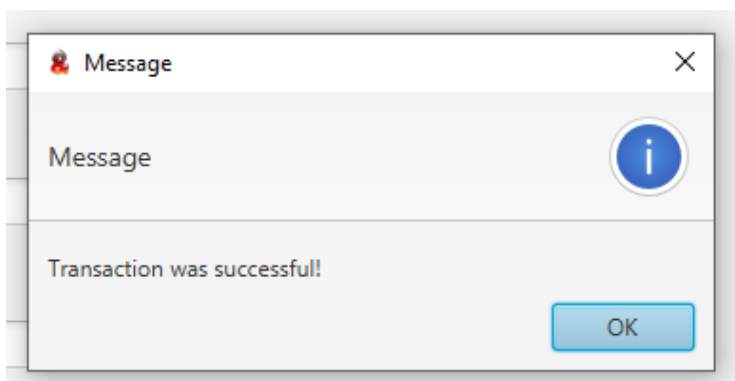
Widok po stronie bazy danych:

	id	login	passwords	accountnumber	username	usersurname	id	userid	currentbalance
▶	1	mz123	mz123	35603377	Maksim	Zakharau	1	1	500
	2	jk123	jk123	53177379	Jan	Kowalski	2	2	150
	3	login123	qazsew123	51315745	Name	Surname	3	3	0
	4	login1111	qazsew123	53768029	Name	Surname	4	4	0

RYSUNEK 27. Tabela testowa 5

Wykonamy operację przelewu środków, wprowadzając numer konta użytkownika **jk123** (53177379) i kwotę 100.

Dostajemy komunikat o poprawnej operacji:



RYSUNEK 28. Testowanie przelewu – Komunikat poprawności

Kwota bilansu zmieniła się dla użytkownika **mz123:**

Your account info:

Login: mz123

Name: Maksim

Surname: Zakharau

Account number: 35603377

Current balance: 400.0

Credit balance: 0.0

Credit available for next value: 10000.0

Available operations:

PayIn

PayOut

Transfer

Credit

Show all completed operations

Delete account

Logout

RYSUNEK 29. Testowanie przelewu 3

Jak i dla użytkownika **jk123:**

Your account info:

Login: jk123

Name: Jan

Surname: Kowalski

Account number: 53177379

Current balance: 250.0

Credit balance: 200.0

Credit available for next value: 9800.0

Available operations:

PayIn

PayOut

Transfer

Credit

Show all completed operations

Delete account

Logout

RYSUNEK 30. Testowanie przelewu 4

Po stronie bazy danych też można zobaczyć zmiany:

	id	login	passwords	accountnumber	username	usersurname	id	userid	currentbalance
▶	1	mz123	mz123	35603377	Maksim	Zakharau	1	1	400
	2	jk123	jk123	53177379	Jan	Kowalski	2	2	250
	3	login123	qazsew123	51315745	Name	Surname	3	3	0
	4	login1111	qazsew123	53768029	Name	Surname	4	4	0

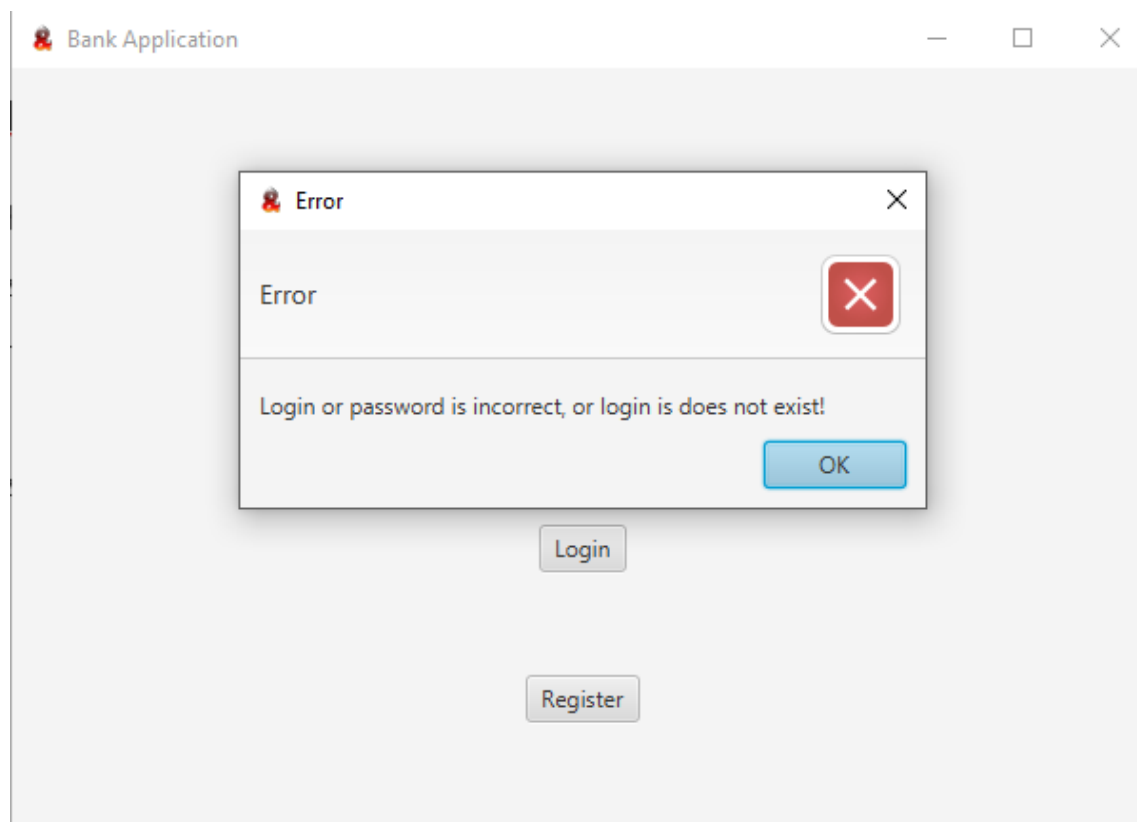
RYSUNEK 31. Tabela testowa 7

5.2. Testowanie mechanizmów bezpieczeństwa

Zostały też przetestowane mechanizmy bezpieczeństwa aplikacji:

- Zabezpieczenie logowania:

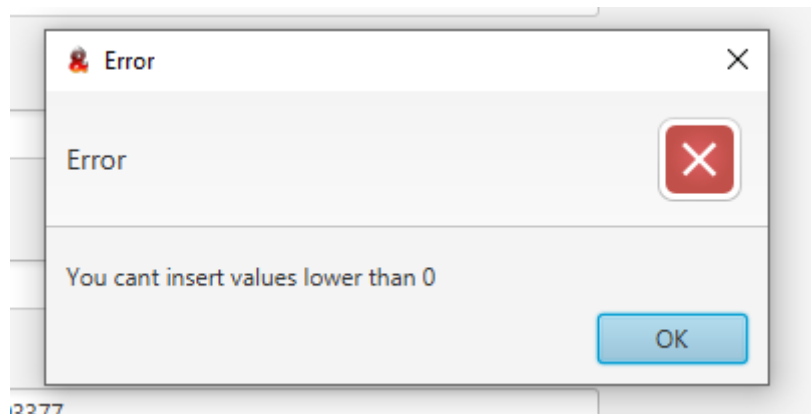
Po wprowadzeniu nieistniejącego loginu lub błędnego hasła aplikacja wyrzuca komunikat:



RYSUNEK 32. Testowanie zabezpieczeń - Logowanie

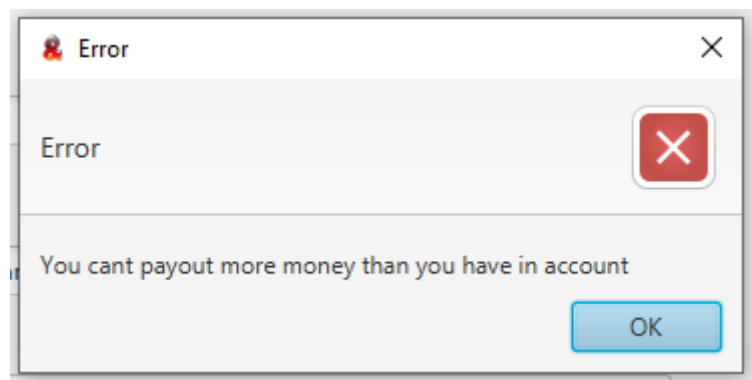
- Zabezpieczenie od niepoprawnej kwoty operacji:

Przy wprowadzeniu kwoty operacji mniejszej od zera dostajemy następujący komunikat:



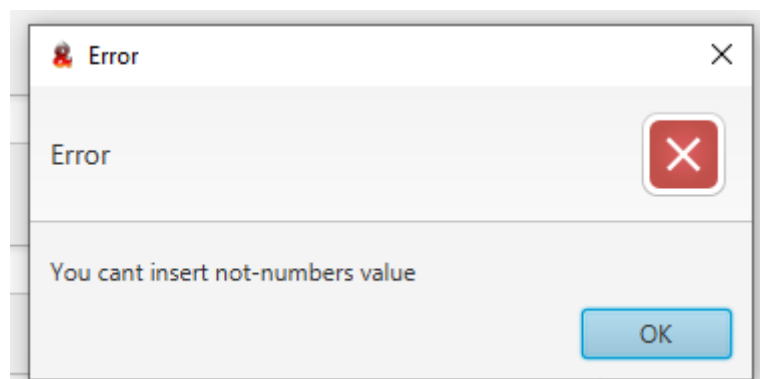
RYSUNEK 33. Testowanie zabezpieczeń – Niepoprawna kwota 1

Przy wprowadzeniu kwoty wypłaty lub przelewu większej od bilansu konta dostajemy następujący komunikat:



RYSUNEK 34. Testowanie zabezpieczeń – Niepoprawna kwota 2

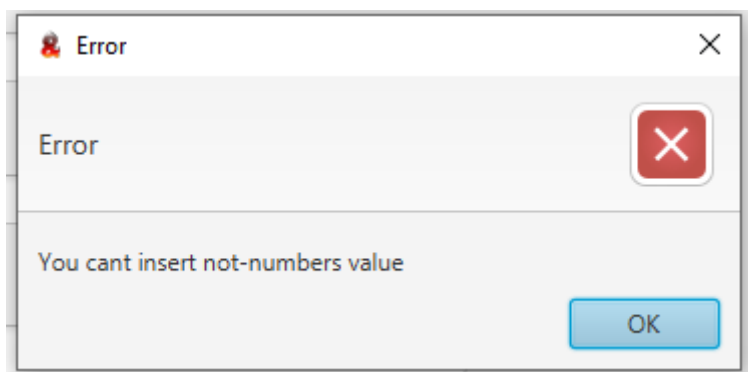
Przy wprowadzeniu znaków nie będących liczbami dostajemy następujący komunikat:



RYSUNEK 35. Testowanie zabezpieczeń – Niepoprawna kwota 3

- Zabezpieczenie od niepoprawnego numeru konta odbiorcy przelewu:

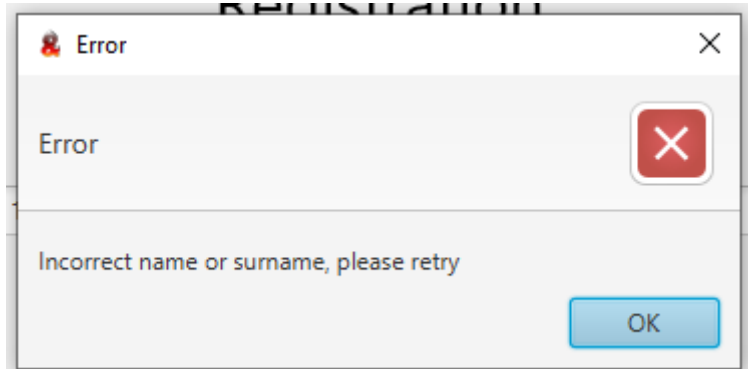
Przy wprowadzeniu niepoprawnego numeru konta dostajemy następujący komunikat:



RYSUNEK 36. Testowanie zabezpieczeń – Numer konta

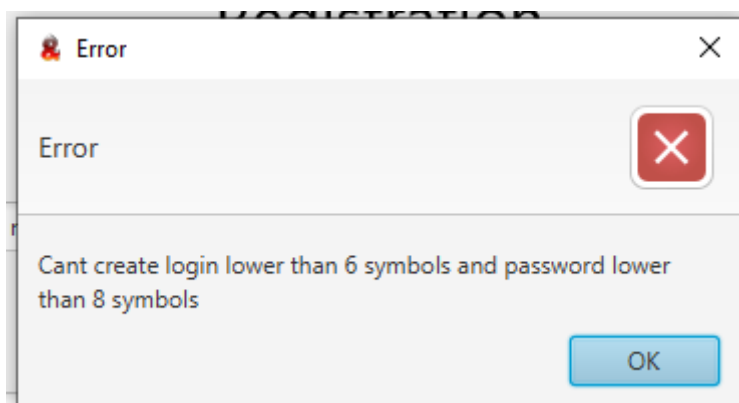
- Zabezpieczenie rejestracji:

Przy wprowadzeniu niepoprawnego imienia i nazwiska dostajemy następujący komunikat:



RYSUNEK 37. Testowanie zabezpieczeń – Rejestracja 1

Przy wprowadzeniu loginu mniejszego od 6 znaków, lub hasła mniejszego od 8 znaków dostajemy następujący komunikat:



RYSUNEK 38. Testowanie zabezpieczeń – Rejestracja 2

5.3. Wnioski z testów

Przy testowaniu systemu był wykryty błąd, który polegał na możliwości wprowadzenia imienia i nazwiska, większego od rozmiaru tabeli w bazie danych. Powodowało to błąd w aplikacji `SqlInjectionError`. Błąd był zlikwidowany dodaniem instrukcji warunkowej:

```
if (nameText.length() < 20 || surnameText.length() < 20) {  
    //some code here  
}else createAlertBox("Error", "Name and surname can't be bigger than 20  
symbols");
```

Poza tym wszystkie funkcje systemu i zabezpieczenia działały poprawnie. Z tego można wywnioskować, że cały system został zaprojektowany i zaimplementowany w sposób poprawny i jest gotowy do użycia.

6. Podsumowanie

Celem projektu było stworzenie aplikacji graficznej i bazy danych, które obsługują konto bankowe. W trakcie projektu były zastosowane technologie MySQLServer 8.0, JavaFX oraz język projektowy UML. Przy projektowaniu aplikacji zostały użyte takie wzorce projektowe, jak MVC i Singleton. Przy projektowaniu bazy danych były stworzone diagramy CDM, LDM oraz PDM. Przy projektowaniu aplikacji zostały użyte diagramy klas, aktywności, maszyny stanów, przypadków użycia. W trakcie testowania były przetestowane wszystkie metody systemu, wykryte i poprawione błędy, szczegółowo opisane w odpowiedniej części. Projekt pozwolił zrozumieć, jak projektować systemy obsługujące bazy danych i implementować interakcje z bazą danych z poziomu aplikacji. W trakcie projektu udało się zaprojektować i zaimplementować cały system konta bankowego, obsługujący się za pomocą bazy danych MySQL i aplikacji Java.

Literatura

- [1] Russ Miles, Kim Hamilton, **UML 2.0. Wprowadzenie**, *Helion, Gliwice, 2011*
- [2] Marcin Lis, **MySQL. Darmowa baza danych. Ćwiczenia praktyczne. Wydanie II**, *Helion, Gliwice, 2007*
- [3] Joshua Bloch, **Java. Efektywne programowanie. Wydanie III**, *Helion, Gliwice, 2014*
- [4] Anton Epple, **JavaFX8**, *dpunkt Verlag, Berlin, 2008*