



**Kolegium Nauk Przyrodniczych
Uniwersytet Rzeszowski**

**Przedmiot:
Bazy Danych**

Projekt:

Baza danych dla aplikacji bankowej

Wykonał:

Oskar Paśko 117987

Eliza Tworkowska 119003

Michał Żychowski 123639

Prowadzący: dr hab. Barbara Pękala, prof. UR

Rzeszów 2023

Spis treści

1.	Opis projektu	3
1.1.	Opis	3
1.2.	Diagram ERD	4
2.	Opis struktur tabel	5
3.	Opis funkcji	11
4.	Opis procedur	13
5.	Prezentacja warstwy użytkowej projektu	19
6.	System kontroli wersji	23
7.	Odtworzenie bazy danych	23
7.1.	Wymagania wstępne	23
7.2.	Importowanie bazy danych	23
7.3.	Otworzenie GUI	24

1. Opis projektu

1.1. Opis

Baza danych „bankapp” została stworzona w celu obsługi aplikacji bankowej. Ta baza danych przechowuje informacje o oddziałach banku, ich pracownikach i klientach.

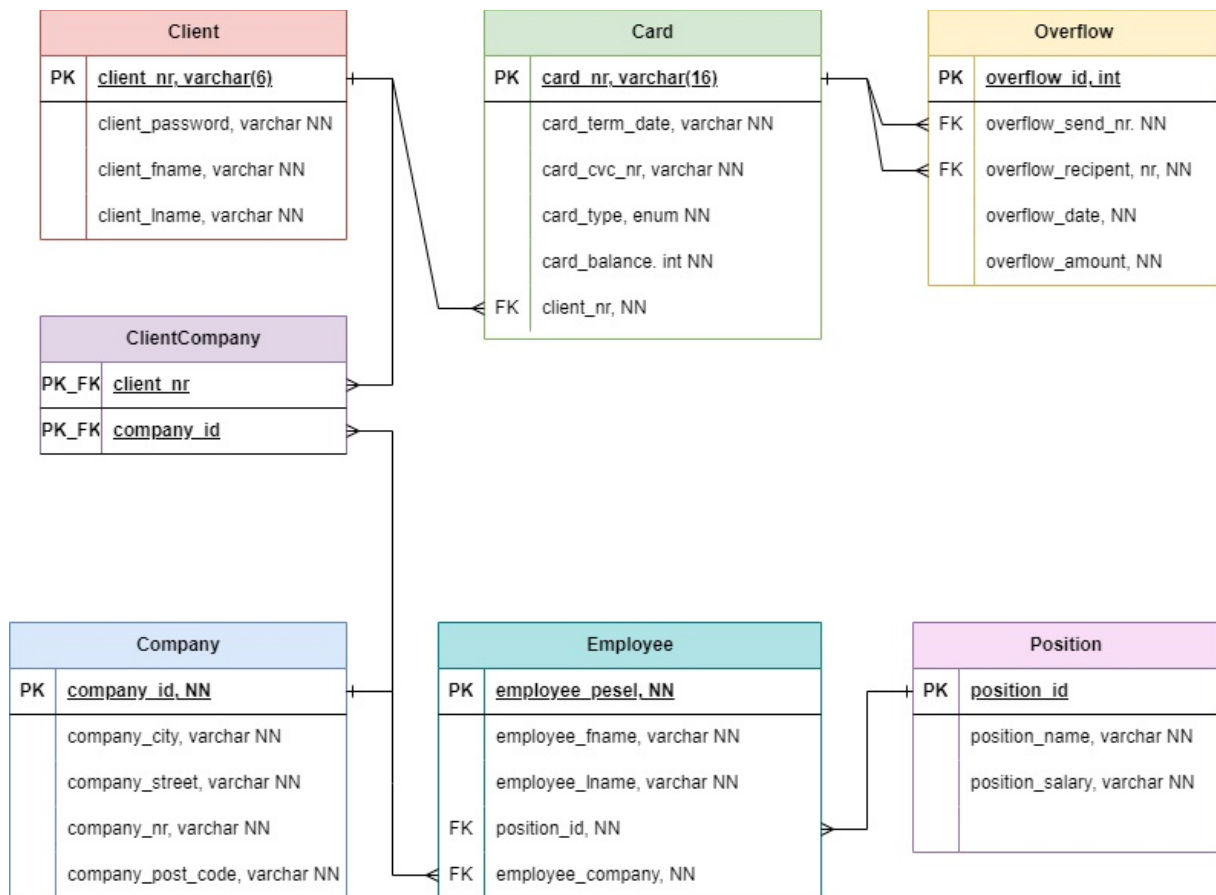
W tabeli „Company” przechowywane są różne oddziały banku, do których przypisani są pracownicy i klienci. Tabela „Employee” zawiera informacje o pracownikach, takie jak PESEL, imię, nazwisko oraz połączona jest z tabelą „Position”, gdzie przechowywane są stanowiska pracowników. Tabela „Client” przechowuje dane o klientach banku, w tym numer klienta, hasło, imię i nazwisko, i jest ona połączona z tabelą „Card”, gdzie przechowywane są informacje o kartach. Tabela „Card” połączona jest z tabelą „Overflow”, gdzie przechowywane są informacje o przelewach.

Tabele są połączone za pomocą kluczy obcych, co umożliwia składanie złożonych zapytań i analizowanie danych w kontekście różnych relacji między tabelami.

Stworzono również czternaście widoków (Views) w celu lepszej prezentacji danych. Widoki są używane w niektórych funkcjach i procedurach.

Projekt zawiera również przykładowe GUI aplikacji, do prezentacji funkcjonalności bazy, napisane w języku PHP.

1.2. Diagram ERD



Rysunek 1. Diagram ERD

Na powyższym rysunku przedstawiono diagram ERD bazy danych. Jak na nim widzimy wszystkie tabele połączone są relacją jeden do wielu.

2. Opis struktur tabel

```
CREATE TABLE client
(
  client_nr VARCHAR(6) PRIMARY KEY NOT NULL UNIQUE CHECK(char_length(client_nr) = 6),
  client_password VARCHAR(100) NOT NULL,
  client_fname VARCHAR(100) NOT NULL,
  client_lname VARCHAR(100) NOT NULL
)ENGINE=InnoDB;
```

Rysunek 1.

Tabela „client”:

Opis: Tabela przechowuje informacje o klientach.

Pola:

- client_nr – unikalny numer klienta (VARCHAR(6) – ograniczony do 6 znaków, PRIMARY KEY – klucz główny, NOT NULL – wymagany, CHECK(char_length(client_nr) = 6) – sprawdza czy numer ma 6 znaków (wymagane 6 znaków))
- client_password - hasło klienta (VARCHAR(100) – ograniczony do 100 znaków, NOT NULL – wymagany)
- client_fname - imię klienta (VARCHAR(100) – ograniczony do 100 znaków, NOT NULL – wymagany)
- client_lname - nazwisko klienta (VARCHAR(100) – ograniczony do 100 znaków, NOT NULL – wymagany)

```

CREATE TABLE card
(
    card_nr VARCHAR(16) PRIMARY KEY NOT NULL UNIQUE CHECK(char_length(card_nr)=16),
    card_term_data DATE NOT NULL,
    card_cvc_number VARCHAR(3) NOT NULL CHECK(char_length(card_cvc_number)=3),
    card_type ENUM('Debetowa', 'Kredytowa'),
    card_balance DECIMAL(11,2) NOT NULL,
    client_nr VARCHAR(6) NOT NULL,

    FOREIGN KEY (client_nr)
    REFERENCES client(client_nr)
    ON UPDATE CASCADE
    ON DELETE RESTRICT
) ENGINE=InnoDB;

```

Rysunek 2.

Tabela „card”:

Opis: Tabela przechowuje informacje o kartach.

Pola:

- card_nr – unikalny numer karty (VARCHAR(16) – ograniczony do 16 znaków, PRIMARY KEY – klucz główny, NOT NULL – wymagany, CHECK(char_length(card_nr)=16) – sprawdza czy numer ma 16 znaków (wymagane 16 znaków))
- card_term_data – termin ważności karty (DATE – format daty, NOT NULL – wymagany)
- card_cvc_number – numer CVC (VARCHAR(3) – ograniczony do 3 znaków, NOT NULL – wymagany, CHECK(char_length(card_cvc_number)=3) – sprawdza czy numer ma 3 znaki (wymagane 3 znaki))
- card_type – typ karty (typ karty czy jest debetowa czy kredytowa)
- card_balance – saldo na karcie (DECIMAL(11,2) – ograniczony do 11 cyfr, w tym 2 po przecinku, NOT NULL – wymagany)
- client_nr – unikalny numer klienta (VARCHAR(6) – ograniczony do 6 znaków, FOREIGN KEY – klucz obcy, NOT NULL – wymagany)

```

CREATE TABLE overflow
(
    overflow_id INT PRIMARY KEY NOT NULL AUTO INCREMENT,
    overflow_send_number VARCHAR(16) NOT NULL,
    overflow_recipient_number VARCHAR(16) NOT NULL,
    overflow_data DATE NOT NULL,
    overflow_amount DECIMAL(11,2) NOT NULL,

    FOREIGN KEY (overflow_send_number)
    REFERENCES card (card_nr)
    ON UPDATE CASCADE
    ON DELETE NO ACTION,

    FOREIGN KEY (overflow_recipient_number)
    REFERENCES card (card_nr)
    ON UPDATE CASCADE
    ON DELETE NO ACTION
)ENGINE=InnoDB;

```

Rysunek 3.

Tabela „overflow”:

Opis: Tabela przechowuje informacje o przelewach.

Pola:

- overflow_id – unikalny ID przelewu (INT, PRIMARY KEY – klucz główny, NOT NULL – wymagany, AUTO_INCREMENT – auto inkrementowany)
- overflow_send_number – numer karty wysyłającego przelew (VARCHAR(16) – ograniczony do 16 znaków, NOT NULL – wymagany, FOREIGN KEY – klucz obcy do card_nr z tabeli „card”)
- overflow_recipient_number – to numer karty odbiorcy (VARCHAR(16) – ograniczony do 16 znaków, NOT NULL – wymagany, FOREIGN KEY – klucz obcy do card_nr z tabeli „card”)
- overflow_date – data przelewu (DATE – format daty, NOT NULL – wymagany)
- overflow_amount – kwota przelewu (DECIMAL(11,2) – ograniczony do 11 cyfr, w tym 2 po przecinku, NOT NULL – wymagany)

```
CREATE TABLE company
(
    company_id INT PRIMARY KEY NOT NULL AUTO INCREMENT,
    company_city VARCHAR(100) NOT NULL,
    company_street VARCHAR(255) NOT NULL,
    company_nr VARCHAR(5) NOT NULL,
    company_post_code VARCHAR(6) NOT NULL CHECK(char_length(company_post_code)=6)
)ENGINE=InnoDB;
```

Rysunek 4.

Tabela „company”:

Opis: Tabela przechowuje informacje o różnych oddziałach banku.

Pola:

- company_id – unikalny ID oddziału banku (INT, PRIMARY KEY – klucz główny, NOT NULL – wymagany, AUTO_INCREMENT – auto inkrementowany)
- company_city – miasto, gdzie znajduje się oddział (VARCHAR(100) – ograniczony do 100 znaków, NOT NULL – wymagany)
- company_street – ulica, na której znajduje się oddział (VARCHAR(255) – ograniczony do 255 znaków, NOT NULL – wymagany)
- company_nr – numer budynku oddziału (adres) (VARCHAR(5) – ograniczony do 5 znaków, NOT NULL – wymagany)
- company_post_code – kod pocztowy (VARCHAR(6) – ograniczony do 6 znaków, NOT NULL – wymagany, CHECK(char_length(company_post_code)=6) – sprawdza czy numer ma 6 znaków (wymagane 6 znaków))

```
CREATE TABLE positions
(
    position_name VARCHAR(100) UNIQUE PRIMARY KEY NOT NULL,
    position_salary DECIMAL(7,2) NOT NULL CHECK(position_salary >= 2200.0)
)ENGINE=InnoDB;
```

Rysunek 5.

Tabela „positions”:

Opis: Tabela przechowuje informacje o różnych stanowiskach w banku.

Pola:

- position_name – unikalna nazwa stanowiska (VARCHAR(100) – ograniczony do 100 znaków, PRIMARY KEY – klucz główny, NOT NULL – wymagany)

- position_salary – pensja (DECIMAL(7,2) – ograniczony do 7 cyfr, w tym 2 po przecinku, NOT NULL – wymagany, CHECK(position_salary >= 2200.0) – sprawdza czy pensja jest większa bądź równa 2200 (wymagane żeby pensja była większa bądź równa 2200)

```
CREATE TABLE employee
(
    employee_pesel VARCHAR(11) UNIQUE PRIMARY KEY NOT NULL CHECK(char_length(employee_pesel)=11),
    employee_email VARCHAR(100) UNIQUE NOT NULL,
    employee_fname VARCHAR(100) NOT NULL,
    employee_lname VARCHAR(100) NOT NULL,
    employee_position VARCHAR(100) NOT NULL DEFAULT 'Brak pozycji',
    employee_company INT NOT NULL,

    FOREIGN KEY (employee_position)
    REFERENCES positions(position_name)
    ON UPDATE CASCADE
    ON DELETE NO ACTION,

    FOREIGN KEY (employee_company)
    REFERENCES company(company_id)
    ON UPDATE CASCADE
    ON DELETE NO ACTION
) ENGINE=InnoDB;
```

Rysunek 6.

Tabela „employee”:

Opis: Tabela przechowuje informacje o pracownikach.

Pola:

- employee_pesel – unikalny PESEL pracownika (VARCHAR(11), PRIMARY KEY – klucz główny, NOT NULL – wymagany, CHECK(char_length(employee_pesel)=11) – sprawdza czy numer ma 11 znaków (wymagane 11 znaków))
- employee_email – email pracownika (VARCHAR(100) – ograniczony do 100 znaków, NOT NULL – wymagany)
- employee_fname – imię (VARCHAR(100) – ograniczony do 100 znaków, NOT NULL – wymagany)
- employee_lname – nazwisko (VARCHAR(100) – ograniczony do 100 znaków, NOT NULL – wymagany)
- employee_position – stanowisko pracownika (VARCHAR(100) – ograniczony do 100 znaków, NOT NULL – wymagany, DEFAULT 'Brak pozycji', FOREIGN KEY – klucz obcy do position_name z tabeli „positions”)
- employee_company – ID oddziału, w którym pracuje pracownik (INT, NOT NULL – wymagany, FOREIGN KEY – klucz obcy do company_id z tabeli „company”)

```

CREATE TABLE client_company
(
  client_nr VARCHAR(16),
  company_id INT,

  PRIMARY KEY (client_nr, company_id),

  FOREIGN KEY (client_nr)
  REFERENCES client (client_nr)
  ON UPDATE CASCADE
  ON DELETE NO ACTION,

  FOREIGN KEY (company_id)
  REFERENCES company (company_id)
  ON UPDATE CASCADE
  ON DELETE NO ACTION
)ENGINE=InnoDB;

```

Rysunek 7.

Tabela „client_company”:

Opis: Tabela łącząca przechowująca informacje o powiązaniach pomiędzy klientami a oddziałami banku.

Pola:

- client_nr – unikalny numer klienta (VARCHAR(16) – ograniczony do 16 znaków, PRIMARY KEY – klucz główny, FOREIGN KEY – klucz obcy do client_nr z tabeli „client”)
- company_id – unikalny ID oddziału banku (INT, PRIMARY KEY – klucz główny, FOREIGN KEY – klucz obcy do company_id z tabeli „company”)

3. Opis funkcji

```
DELIMITER $$
CREATE FUNCTION client_company_count(city VARCHAR(100))
RETURNS VARCHAR(10)
DETERMINISTIC
BEGIN
    DECLARE liczba VARCHAR(10);
    SELECT COUNT(*) INTO liczba FROM client_company_view WHERE company_city = city;
    RETURN liczba;
END$$
DELIMITER ;
```

Rysunek 8.

Funkcja "client_company_count" to funkcja licząca klientów w każdym mieście. Funkcja przyjmuje parametr "city" typu VARCHAR(100) i zlicza ilość klientów powiązanych z oddziałami banku w danym mieście poprzez wykonanie zapytania do widoku "client_company_view" i zwraca wynik jako wartość typu VARCHAR(10).

```
DELIMITER $$
CREATE FUNCTION city_salary(city VARCHAR(100))
RETURNS VARCHAR(10)
DETERMINISTIC
BEGIN
    DECLARE suma VARCHAR(10);
    SELECT SUM(position_salary) INTO suma FROM detailed_employee WHERE company_city = "Rzeszów";
    RETURN suma;
END$$
DELIMITER ;
```

Rysunek 9.

Funkcja "city_salary" to funkcja zwracająca sumę wypłat dla pracowników w poszczególnej placówce. Funkcja przyjmuje parametr "city" typu VARCHAR(100) i oblicza sumę wypłat dla pracowników związanych z daną placówką poprzez wykonanie zapytania do tabeli "detailed_employee" z warunkiem, że "company_city" jest równe wartości "Rzeszów". Następnie funkcja zwraca wynik jako wartość typu VARCHAR(10).

```

DELIMITER $$
CREATE FUNCTION log_in(email VARCHAR(100), passwd VARCHAR(100))
RETURNS VARCHAR(1)
DETERMINISTIC
BEGIN
    DECLARE temp VARCHAR(1);
    SELECT COUNT(*) INTO temp FROM employee WHERE employee_email = email AND employee_pesel = passwd;
    RETURN temp;
END$$
DELIMITER ;

```

Rysunek 10.

Funkcja "log_in" to funkcja logująca pracownika. Funkcja przyjmuje dwa parametry: "email" typu VARCHAR(100) oraz "passwd" typu VARCHAR(100) i sprawdza, czy istnieje pracownik o podanym emailu i hasle w tabeli "employee". Jeśli taki pracownik istnieje, funkcja zwraca wartość "1" jako potwierdzenie logowania, w przeciwnym razie zwraca wartość "0".

```

DELIMITER $$
CREATE FUNCTION brutto(balance DECIMAL(11,2), tax FLOAT)
RETURNS DECIMAL(11,2)
DETERMINISTIC
BEGIN
    DECLARE temp DECIMAL(7,2);
    set temp = balance + (balance * cast(tax as decimal(7,2)));
    RETURN temp;
END$$
DELIMITER ;

```

Rysunek 11.

Funkcja "brutto" to funkcja obliczająca wartość brutto. Funkcja przyjmuje dwa parametry: "balance" typu DECIMAL(11,2) oraz "tax" typu FLOAT i oblicza wartość brutto na podstawie podanego salda (balance) i stawki podatku (tax). Oblicza to poprzez dodanie do salda kwoty równowrotnej do podatku i zwraca wynik jako wartość DECIMAL(11,2).

```

DELIMITER $$
CREATE FUNCTION company_salary(city VARCHAR(100), street VARCHAR(100), nr VARCHAR(5), post_code VARCHAR(6))
RETURNS VARCHAR(10)
DETERMINISTIC
BEGIN
    DECLARE suma VARCHAR(10);
    SELECT SUM(position_salary) INTO suma
    FROM detailed_employee
    WHERE company_city = city AND company_street = street AND company_nr = nr AND company_post_code = post_code;
    RETURN suma;
END$$
DELIMITER ;

```

Rysunek 12.

Funkcja "company_salary" to funkcja obliczająca sumę wynagrodzeń w konkretnym oddziale. Funkcja przyjmuje cztery parametry: "city" typu VARCHAR(100), "street" typu VARCHAR(100), "nr" typu VARCHAR(5) oraz "post_code" typu VARCHAR(6) i oblicza sumę wynagrodzeń wszystkich pracowników znajdujących się w określonym oddziale na podstawie podanych danych (miasta, ulicy, numeru budynku i kodu pocztowego). Zwraca wynik jako wartość VARCHAR(10).

4. Opis procedur

```

DELIMITER $$
$$
CREATE PROCEDURE bankapp.employ(IN employy VARCHAR(20), OUT emp INT)
BEGIN
    SELECT COUNT(*) INTO emp FROM bankapp.employee
    WHERE employee_position = employy;
END$$
DELIMITER ;

```

Rysunek 13.

Procedura "employ" to procedura licząca pracowników na konkretnych stanowiskach. Procedura przyjmuje jeden wejściowy parametr "employy" typu VARCHAR(20) oraz jeden parametr wyjściowy "emp" typu INT i zlicza pracowników znajdujących się na konkretnych stanowiskach w tabeli "employee" na podstawie wartości przekazanej do parametru wejściowego "employy" i przypisuje wynik do parametru wyjściowego "emp".

```

DELIMITER $$
$$
CREATE PROCEDURE bankapp.client_overflow(IN cli CHAR(6), OUT emp INT)
BEGIN
    SELECT COUNT(*) INTO emp FROM detailed_overflow
    WHERE client_nr = cli;
END$$
DELIMITER ;

```

Rysunek 14.

Procedura "client_overflow" to procedura licząca ilość wykonanych przelewów przez danego klienta. Procedura przyjmuje jeden wejściowy parametr "cli" typu CHAR(6) oraz jeden parametr wyjściowy "emp" typu INT i zlicza ilość wykonanych przelewów przez danego klienta na podstawie wartości przekazanej do parametru wejściowego "cli" i przypisuje wynik do parametru wyjściowego "emp".

```

DELIMITER $$
$$
CREATE PROCEDURE bankapp.company_employee(IN city VARCHAR(20), OUT emp INT)
BEGIN
    SELECT COUNT(*) INTO emp FROM company
    JOIN employee
    ON employee.employee_company = company.company_id
    WHERE company_city = city;
END$$
DELIMITER ;

```

Rysunek 15.

Procedura "company_employee" to procedura licząca ilość pracowników w każdym mieście. Procedura przyjmuje jeden wejściowy parametr "city" typu VARCHAR(20) oraz jeden parametr wyjściowy "emp" typu INT i zlicza ilość pracowników w każdym mieście na podstawie wartości przekazanej do parametru wejściowego "city" i przypisuje wynik do parametru wyjściowego "emp".

```

DELIMITER $$
$$
CREATE PROCEDURE add_employee (IN pesel varchar(11), IN email varchar(50), IN fname varchar(50), IN lname varchar(50), IN posn varchar(50), IN company INT)
BEGIN
    insert into employee values (pesel, email, fname, lname, posn, company);
END$$
DELIMITER ;

```

Rysunek 16.

Procedura "add_employee" to procedura dodająca nowego pracownika. Procedura przyjmuje sześć wejściowych parametrów: "pesel" (typu varchar(11)), "email" (typu varchar(50)), "fname" (typu varchar(50)), "lname" (typu varchar(50)), "posn" (typu varchar(50)) i "company" (typu INT) i dodaje nowego pracownika do tabeli "employee" przy użyciu wartości przekazanych do parametrów wejściowych.

```

DELIMITER $$
$$
CREATE PROCEDURE add_client (IN nr varchar(6), IN passwd varchar(50), IN fname varchar(50), IN lname varchar(50))
BEGIN
    insert into client values (nr, passwd, fname, lname);
END$$
DELIMITER ;

```

Rysunek 17.

Procedura "add_client" to procedura dodająca nowego klienta. Procedura przyjmuje cztery wejściowe parametry: "nr" (typu varchar(6)), "passwd" (typu varchar(50)), "fname" (typu varchar(50)) i "lname" (typu varchar(50)) i dodaje nowego klienta do tabeli "client" przy użyciu wartości przekazanych do parametrów wejściowych.

```

DELIMITER $$
$$
CREATE PROCEDURE add_card (IN nr varchar(16), IN term_data DATE, IN cvc varchar(3), IN card_type ENUM('Debetowa', 'Kredytowa'), IN balance FLOAT, IN client VARCHAR(6))
BEGIN
    insert into card values (nr, term_data, cvc, card_type, balance, client);
END$$
DELIMITER ;

```

Rysunek 18.

Procedura "add_card" to procedura dodająca nową kartę. Procedura przyjmuje sześć wejściowych parametrów: "nr" (typu varchar(16)), "term_data" (typu DATE), "cvc" (typu varchar(3)), "card_type" (typu ENUM), "balance" (typu FLOAT) i "client" (typu VARCHAR(6)) i dodaje nową kartę do tabeli "card" przy użyciu wartości przekazanych do parametrów wejściowych.

```

DELIMITER $$
$$
CREATE PROCEDURE add_position (IN name varchar(16), IN salary FLOAT)
BEGIN
    insert into positions values (name, salary);
END$$
DELIMITER ;

```

Rysunek 19.

Procedura "add_position" to procedura dodająca nowe stanowisko. Procedura przyjmuje dwa wejściowe parametry: "name" (typu varchar(16)) i "salary" (typu FLOAT) i dodaje nowe stanowisko do tabeli "positions" przy użyciu wartości przekazanych do parametrów wejściowych.

```

DELIMITER $$
$$
CREATE PROCEDURE indebted(IN c_type ENUM('Debetowa', 'Kredytowa'), in balance DECIMAL(11,2))
BEGIN
    select client_nr, card_nr, client_fname, client_lname, card_type, card_balance from card_client cc
    where card_type = c_type and card_balance < balance
    order by card_balance;
END$$
DELIMITER ;

```

Rysunek 20.

Procedura "indebted" to procedura pokazująca karty w przedziale pieniężnym. Procedura przyjmuje dwa wejściowe parametry: "c_type" (typu ENUM) i "balance" (typu DECIMAL(11,2)) i wyświetla informacje o kartach klientów, których typ karty pasuje do wartości "c_type" i saldo karty jest mniejsze niż wartość "balance". Wyniki są sortowane według salda karty.


```

DELIMITER $$
$$
CREATE PROCEDURE send_overflows_by_client(IN client VARCHAR(6), in first_date DATE, in second_date DATE)
BEGIN
    select client_nr, overflow_send_number, card_type, card_balance, overflow_recipient_number, overflow_data, overflow_amount
    from send_overflows
    where client_nr = client and overflow_data >= first_date and overflow_data <= second_date
    order by overflow_data;
END$$
DELIMITER ;

```

Rysunek 21.

Procedura "send_overflows_by_client" to procedura pokazująca tabele przelewów klienta w danym okresie czasu. Procedura przyjmuje trzy wejściowe parametry: "client" (typu VARCHAR(6)), "first_date" (typu DATE) i "second_date" (typu DATE) i wyświetla informacje dotyczące przelewów klienta, których numer klienta odpowiada wartości "client" oraz które zostały wykonane w okresie między "first_date" a "second_date". Wyniki są sortowane według daty przelewu.

```

DELIMITER $$
$$
CREATE PROCEDURE position_salary_brutto(tax FLOAT)
BEGIN
    SELECT position_name, position_salary AS 'netto', brutto(position_salary, tax) AS 'brutto'
    FROM positions
    ORDER BY position_salary;
END$$
DELIMITER ;

```

Rysunek 22.

Procedura "position_salary_brutto" to procedura wyświetlająca zarobki pozycji w firmie netto i brutto. Procedura przyjmuje jeden wejściowy parametr "tax" (typu FLOAT) i wyświetla nazwy pozycji oraz odpowiadające im wynagrodzenia netto i brutto. Wynagrodzenie brutto jest obliczane na podstawie wynagrodzenia netto i wartości podatku przekazanej jako parametr. Wyniki są sortowane według wynagrodzenia netto.

```

DELIMITER $$
$$
CREATE PROCEDURE indebtedted_city(city VARCHAR(100), street VARCHAR(100), nr VARCHAR(5), post_code VARCHAR(6))
BEGIN
    SELECT client.client_nr, company_city, company_street, company_nr, company_post_code, card_nr, card_balance
    FROM client_company
    LEFT JOIN company USING(company_id)
    LEFT JOIN client USING(client_nr)
    LEFT JOIN card ON card.client_nr = client.client_nr
    WHERE card_balance < 0
    AND company_city = city
    AND company_street = street
    AND company_nr = nr
    AND company_post_code = post_code
    ORDER BY card_balance;
END$$
DELIMITER ;

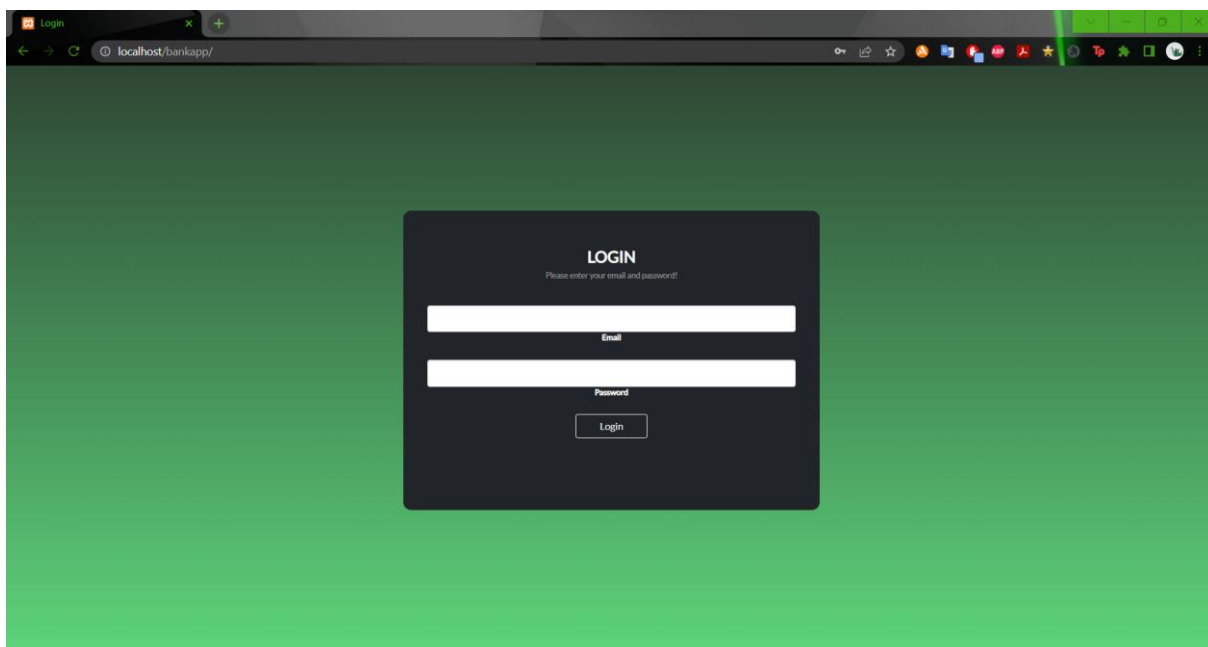
```

Rysunek 23.

Procedura "indebtedted_city" to procedura wyświetlająca zadłużonych klientów w konkretnej placówce. Procedura przyjmuje cztery wejściowe parametry: city (typ VARCHAR), street (typ VARCHAR), nr (typ VARCHAR) i post_code (typ VARCHAR) i wyświetla informacje o zadłużonych klientach znajdujących się w konkretnej placówce. Wyświetlane są numery klientów, informacje o miejscu pracy klienta (miasto, ulica, numer, kod pocztowy), numer karty oraz saldo zadłużenia. Wyniki są sortowane według salda zadłużenia.

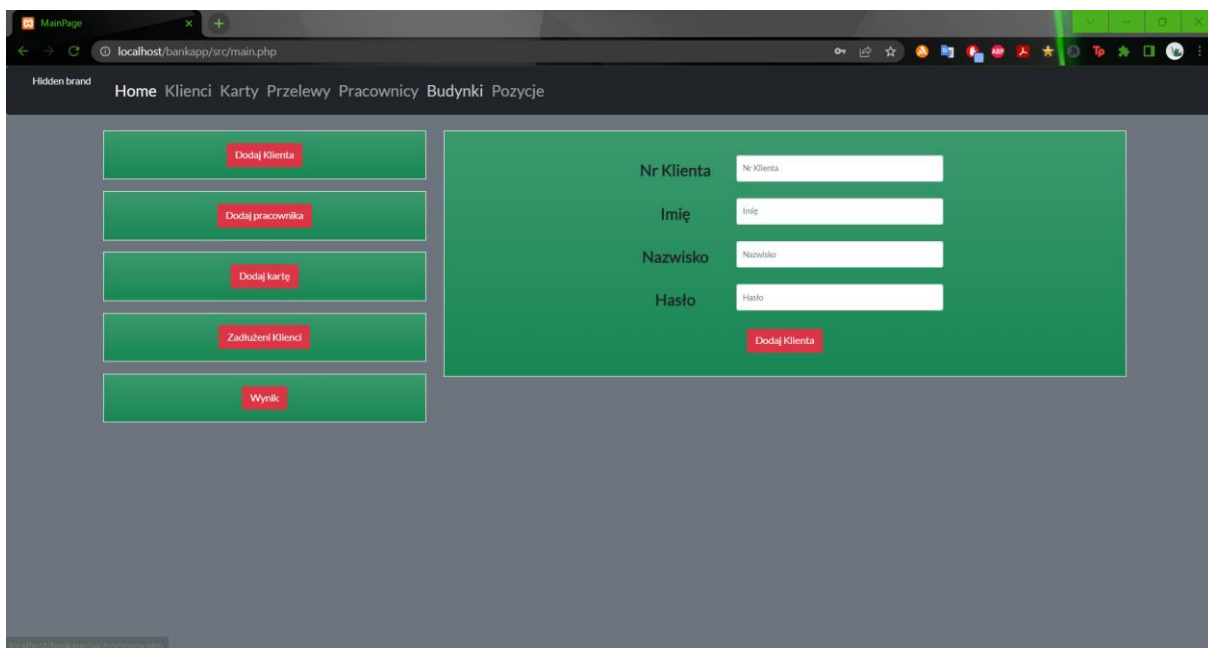
5. Prezentacja warstwy użytkowej projektu

Na potrzeby prezentacji funkcjonalności bazy, projekt zawiera również przykładowe GUI aplikacji napisane w języku PHP. Na rysunku 24 przedstawiono główne okno aplikacji, czyli ekran logowania (patrz Rysunek 24.), na którym pracownik banku może się zalogować.



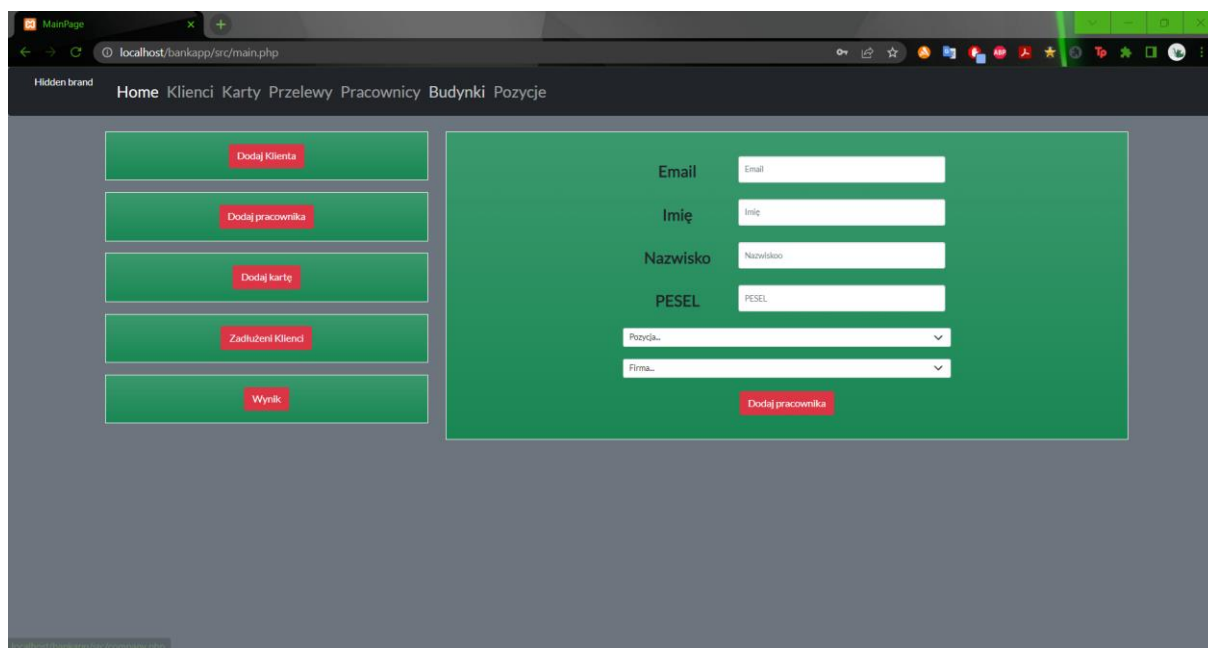
Rysunek 24.

Po zalogowaniu widzimy panel, gdzie możemy dodać klienta po wpisaniu odpowiednich danych (patrz Rysunek 25.).



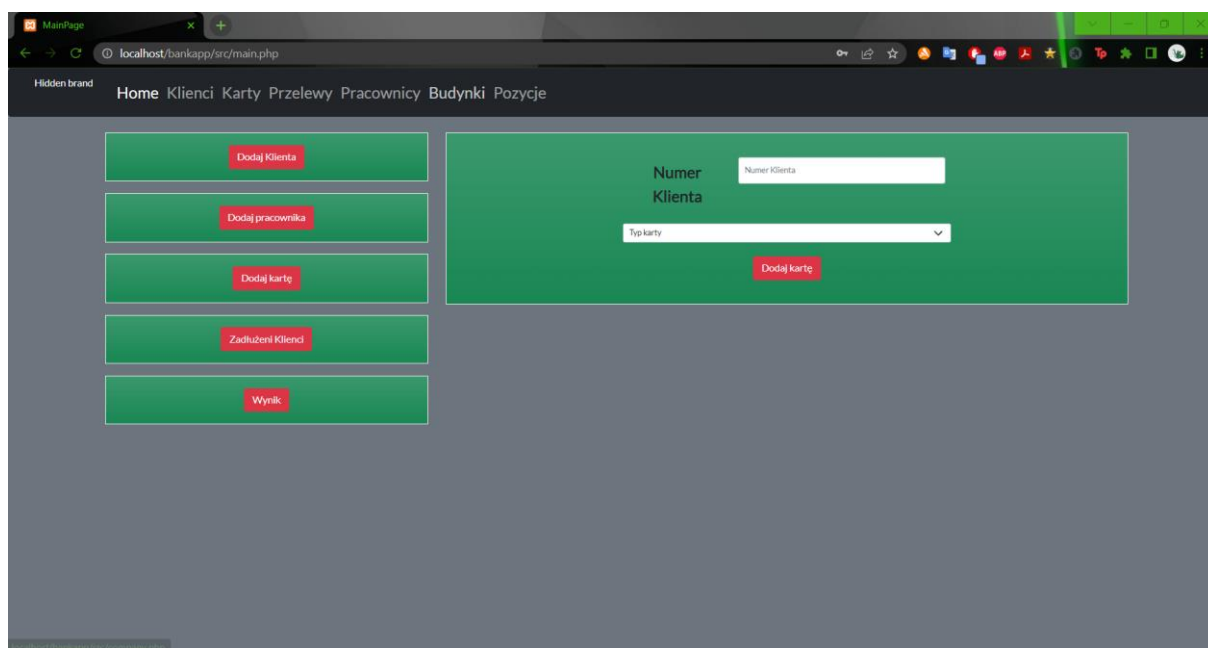
Rysunek 25.

Po lewej stronie widzimy przyciski, które przenoszą nas na odpowiednie panele (np. przycisk „Dodaj pracownika” przeniesie nas do panelu, gdzie możemy dodać pracownika (patrz Rysunek 26.)).



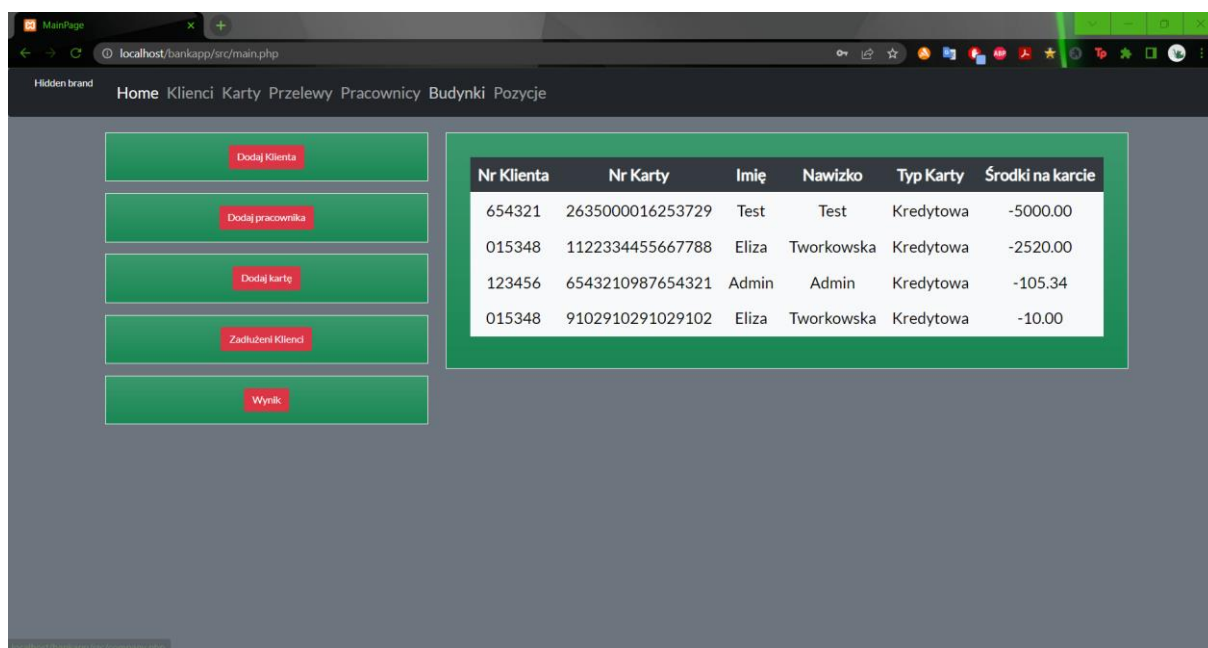
Rysunek 26.

Wszystkie panele są do siebie analogiczne pod względem funkcjonalności. Po kliknięciu w przycisk „Dodaj kartę” znajdziemy się na panelu, gdzie możemy dodać kartę (patrz Rysunek 27.).



Rysunek 27.

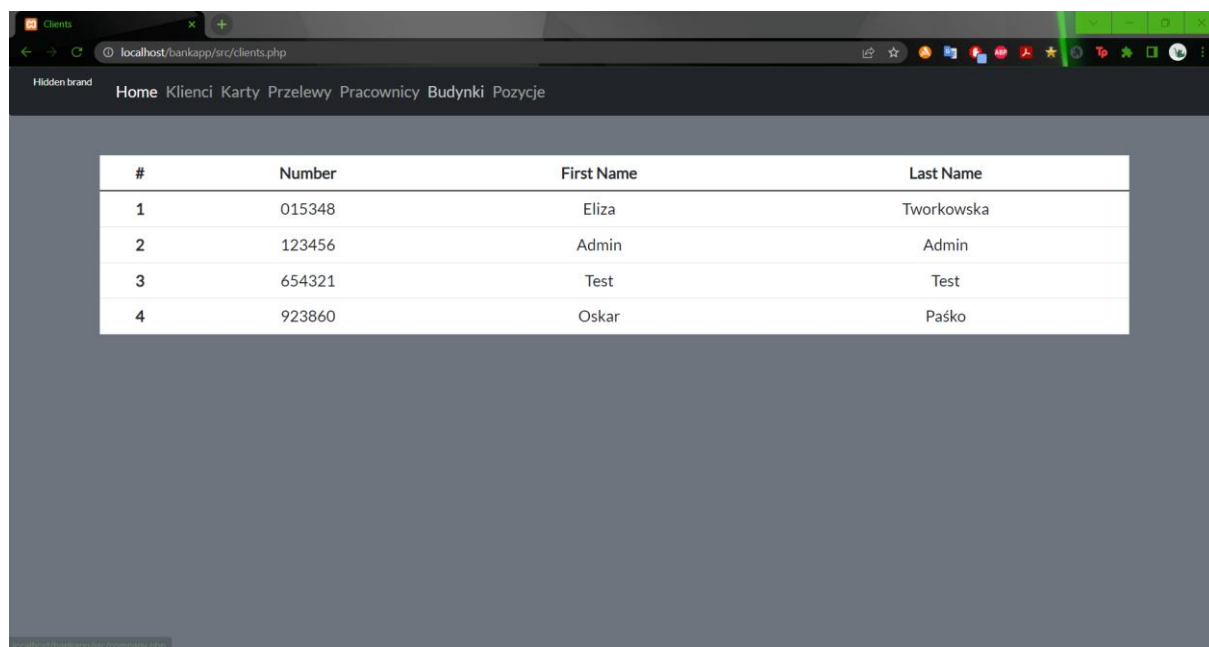
Po kliknięciu przycisku „Zadłużeni klienci” wyświetli nam się tabela z zadłużonymi klientami (patrz Rysunek 28.).



Rysunek 28.

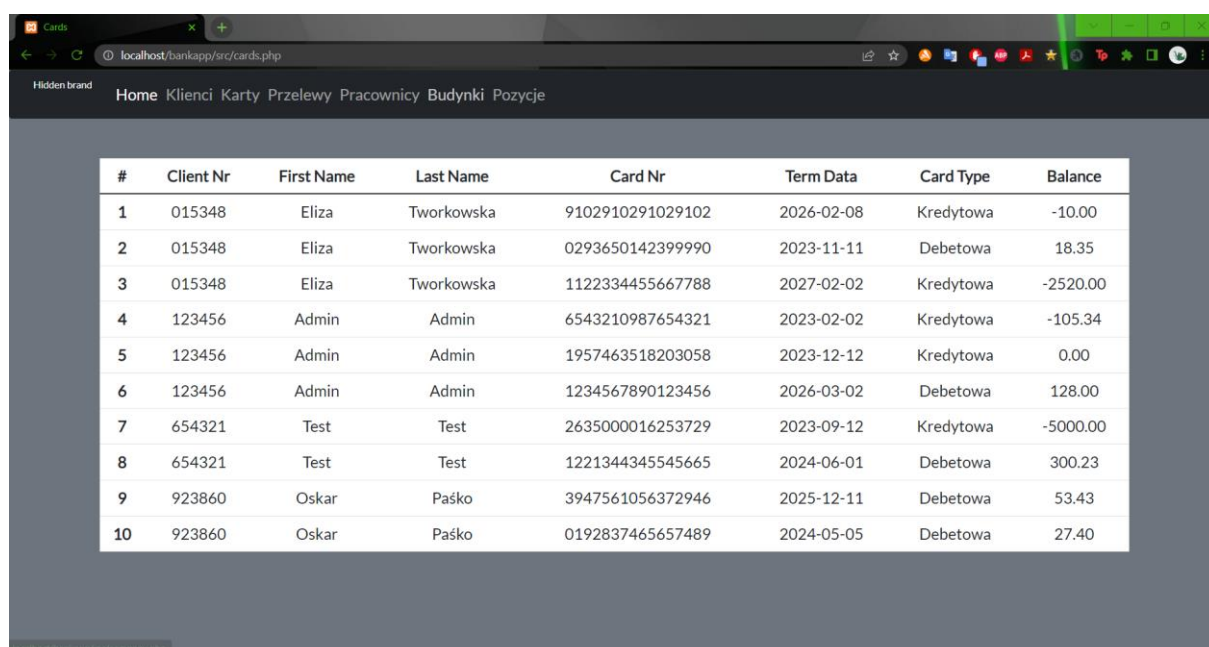
Możemy jeszcze zauważyć, że u góry posiadamy menu z możliwościami wyboru: „Home”, „Klienci”, „Karty”, „Przelewy”, „Pracownicy”, „Budynki”, „Pozycje”. Po kliknięciu danej

opcji wyświetli nam się tabela z danymi informacjami, np. po kliknięciu w „Klienci” wyświetli nam się tabela z klientami (patrz Rysunek 29.), po kliknięciu w „Karty” wyświetli nam się tabela z kartami (patrz Rysunek 30.). Wszystkie przyciski menu działają analogicznie. Przycisk „Home” wraca nas do strony głównej, czyli strony dodawania klienta (patrz Rysunek 25.)



#	Number	First Name	Last Name
1	015348	Eliza	Tworkowska
2	123456	Admin	Admin
3	654321	Test	Test
4	923860	Oskar	Paško

Rysunek 29.

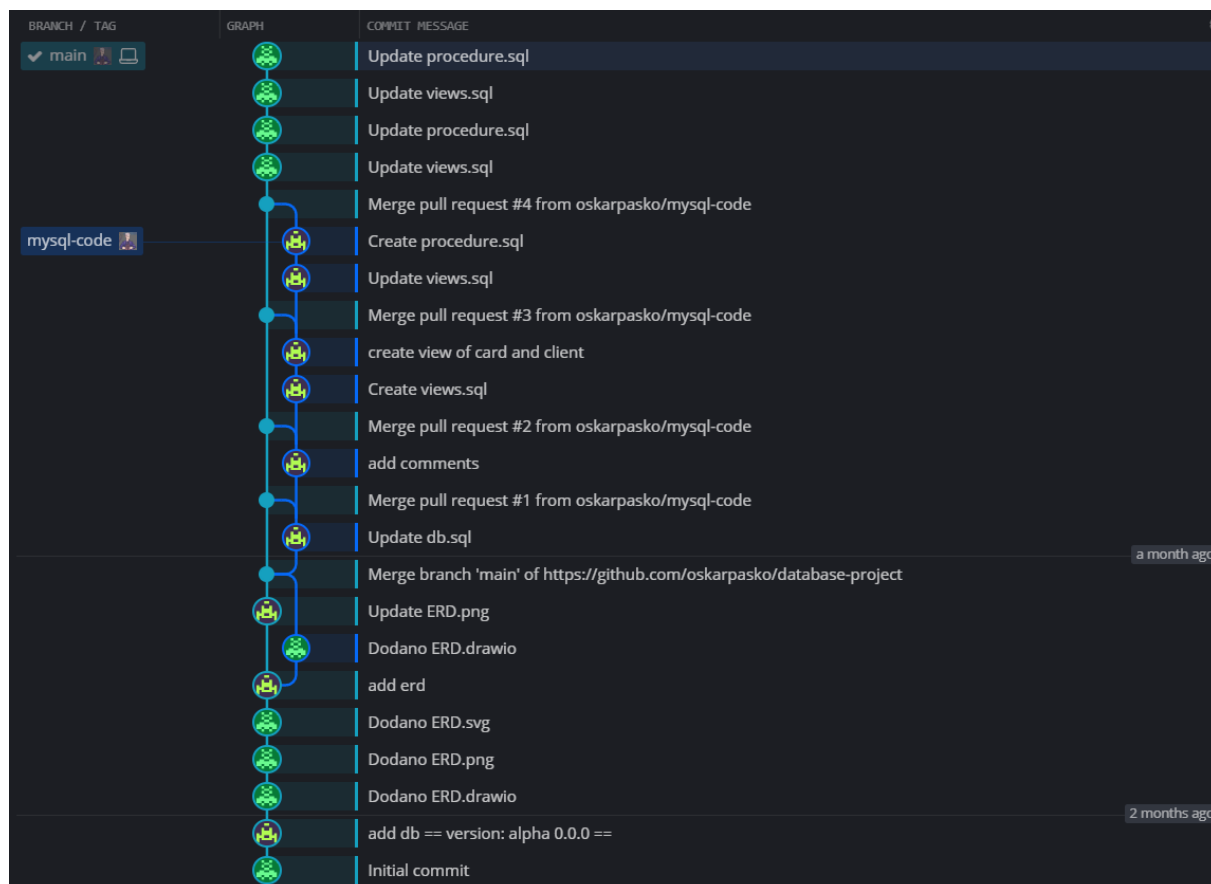


#	Client Nr	First Name	Last Name	Card Nr	Term Data	Card Type	Balance
1	015348	Eliza	Tworkowska	9102910291029102	2026-02-08	Kredytowa	-10.00
2	015348	Eliza	Tworkowska	0293650142399990	2023-11-11	Debetowa	18.35
3	015348	Eliza	Tworkowska	1122334455667788	2027-02-02	Kredytowa	-2520.00
4	123456	Admin	Admin	6543210987654321	2023-02-02	Kredytowa	-105.34
5	123456	Admin	Admin	1957463518203058	2023-12-12	Kredytowa	0.00
6	123456	Admin	Admin	1234567890123456	2026-03-02	Debetowa	128.00
7	654321	Test	Test	2635000016253729	2023-09-12	Kredytowa	-5000.00
8	654321	Test	Test	1221344345545665	2024-06-01	Debetowa	300.23
9	923860	Oskar	Paško	3947561056372946	2025-12-11	Debetowa	53.43
10	923860	Oskar	Paško	0192837465657489	2024-05-05	Debetowa	27.40

Rysunek 30.

6. System kontroli wersji

Projekt realizowany był z wykorzystaniem systemu kontroli wersji Git, a wszystkie pliki źródłowe projektu znajdują się pod adresem: <https://github.com/oskarpasko/database-project>. Na Rysunku 1. przedstawiono zrzut ekranu pokazujący część historii commit'ów.



Rysunek 31. Historia commit'ów

7. Odtworzenie bazy danych

7.1. Wymagania wstępne

- XAMPP 8.2.0
- Apache/2.4.54 (Win64)
- PHP/8.2.0
- MySQLnd 8.2.0

7.2. Importowanie bazy danych

Za pomocą komendy „git clone <https://github.com/oskarpasko/database-project.git>” zaimportować projekt z zdalnego repozytorium. Następnie w programie do obsługi bazy danych np. w DBeaver, MySQL Workbench itp. wklej oraz zainicjuj kolejno pliki: db.sql,

views.sql, procedure.sql, functions.sql lub tylko plik db_import.sql. (Pamiętaj aby z plików z procedurami i funkcjami nie wklejać wywołań.)

7.3. Otworzenie GUI

Po wykonaniu poprzednich kroków, wykonać następujące czynności:

- upewnij się, że jest zaimportowana baza danych
- włącz aplikację XAMPP i uruchom funkcję Apache oraz MySQL
- przejdź do Explorer -> htdocs i skopiuj tam folder 'bankapp'
- wpisz w przeglądarkę <http://localhost/bankapp/>
- powinna włączyć się główna strona aplikacji