Skład: Dawid Orłowski, Konrad Mech, Dawid Plewa

Wydział Inżynierii Elektrycznej i Komputerowej
Informatyka w Inżynierii Komputerowej

PROGRAMOWANIE W JĘZYKU JAVA

Sprawozdanie

„Tworzenie oprogramowania dla systemu informatycznego kasyno”

SPIS TREŚCI

1.1 Cel i zakres projektu

Zadaniem projektu jest stworzenie aplikacji kasyna, zapewniającej niezapomnianą rozrywkę i wspaniałe przeżycia.

Aplikacja będzie połączona z bazą danych w celu kolekcjonowania najważniejszych informacji.

Dla użytkowników będzie dostępny panel aplikacji wyświetlany w autorsko stworzonym interfejsie graficznym.

Zajmiemy się stworzeniem logiki i mechanizmu każdej z poszczególnych gier dostępnych w aplikacji.

Użytkownik otrzyma dostęp do wpłacania oraz wypłacania swoich środków w dowolnym momencie poprzez prosty dostęp do swojego portfela w menu głównym aplikacji.

Projekt będzie można w przyszłości poszerzać o dowolną ilość gier oraz dodatkowych funkcji.

1.2 Charakterystyka użytkowników

Nasza aplikacja będzie przeznaczona dla graczy do różnego sposobu relaksu jak i poczucia adrenaliny do uprawiania hazardu. Do wyboru będzie kilka gier, każdy znajdzie coś dla siebie.

Kasyno przeznaczone będzie dla osób pełnoletnich, chętnych na zapoznanie się z nowym sposobem rozrywki online, którą będziemy zapewniać.

1.3 Główne funkcje produktu

Możemy wyróżnić wiele funkcji aplikacji, natomiast głównymi będą:

-**Panel rejestracji i logowania** – bezpieczny mechanizm rejestracji konta graczy przy użyciu hasła. Zabezpieczone

systemy logowania oraz rejestracji wraz z autorską szatą graficzną.
-**Menu aplikacji** – główny panel naszej aplikacji wyświetlany w szacie graficznej. Zawierać będzie stronę główną, panel użytkownika, panel zarządzania bilansem konta oraz panel gier opisujący zasady dostępnych gier w naszym kasynie.
-**Portfel i transakcje** – system zarządzania portfelem gracza, wpłaty, wypłaty z konta przy użyciu nowoczesnych technologii oraz systemów płatności.

1.4 Wymagania funkcjonalne

-rejestracja,
-panel logowania,

Przykład użycia:

Użytkownikowi po uruchomieniu aplikacji ukazuje się okno logowania oraz rejestracji, w którym po wprowadzeniu danych w poprawny sposób może utworzyć własne konto w naszym systemie. Po rejestracji użytkownik przechodzi do logowania podanym wcześniej emailem oraz hasłem. Gdy logowanie skończy się poprawnie użytkownikowi ukaże się główne okno aplikacji.

-panel użytkownika,
-edycja danych klienta,

Przykład użycia:

Użytkownik po zalogowaniu w menu głównym ma dostęp do swojego panelu, gdzie wyświetlane są jego wszystkie dane. Posiada on możliwość edycji danych takich jak email, hasło, czy też swoją nazwę.

-przegląd gier,
-przegląd zasad gier,

Przykład użycia:

Użytkownik w menu głównym posiada dostęp do okna gier dostępnych w aplikacji, w którym opisane są wszystkie aktualnie dostępne gry wraz z poszczególnymi zasadami.

-wpłaty,
-wypłaty,

Przykład użycia:

Użytkownik w menu głównym posiada dostęp do swojego portfela, w którym na bieżąco może dysponować swoimi środkami, wpłacać oraz wypłacać swoje pieniądze. W planach jest stworzenie historii transakcji, aby każda osoba mogła w panelu aplikacji sprawdzać swoje wpłaty oraz wypłaty wykonane w przeszłości.

-zarządzanie bazą danych (administrator)

Przykład użycia:

Administrator posiada dostęp do bazy danych postawionej na systemie localhost phpMyAdmin. Na systemie ma on możliwość usuwania, czy też edycji danych zarejestrowanych użytkowników. Posiada również możliwość rozszerzania bazy o nowe tabele w zależności od aktualnych potrzeb aplikacji.

1.5 Wymagania niefunkcjonalne

-Bezpieczeństwo,

Zapewnienie ochrony systemu przed nieautoryzowanym dostępem.

-Wydajność,

Optymalizacja systemu do szybkiego przetwarzania transakcji oraz zapytań.

-Dostępność,

Zapewnienie ciągłego, nieprzerwanego dostępu do systemu dla użytkowników.

-Zarządzanie bazą danych,

Efektywne zarządzanie bazą danych, w tym bezpieczne przechowywanie danych.

-Skalowalność,

Zdolność system do obsługi rosnącej liczby użytkowników oraz rozszerzającej się oferty gier.

-Losowość gier – „Fair Play”,

Zapewnienie, że wyniki gier są generowane w sposób całkowicie losowy i sprawiedliwy.

-Elastyczność systemu,

Możliwość dostosowywania i modyfikowania systemu w reakcji na zmieniające się potrzeby biznesowe.

-Przyjazny dla oka interfejs,

Projekt interfejsu użytkownika, który jest estetycznie atrakcyjny oraz intuicyjny w obsłudze.

-Szyfrowanie danych,

Implementacja silnych metod szyfrowania danych w celu zwiększenia bezpieczeństwa danych użytkownika.

-Zabezpieczenie kont,

Stosowanie środków bezpieczeństwa, takich jak uwierzytelnianie dwuskładnikowe, w celu większej ochrony kont użytkowników.

1.9 Problem, Założenia projektowe

Branża hazardowa, szczególnie kasyna zmaga się z wyzwaniami związanymi z efektywnym zarządzaniem operacjami, personalizacją doświadczeń oraz utrzymanie konkurencyjności na rynku.

Głównymi problemami są:

1. **Brak personalizacji –** istniejące rozwiązania nie są wystarczalne dla graczy potrzebujących nowych doświadczeń, co może prowadzić do utraty lojalności klientów.
2. **Bezpieczeństwo danych –** zagadnienia związane z bezpieczeństwem danych i transakcji są narażone na krytyczne i liczne ataki cybernetyczne. Trzeba użyć dużej ilości różnorodnych zabezpieczeń.

Założenia projektowe mają na celu adresowanie identyfikowanych problemów w branży hazardowej, jednocześnie umożliwiając rozwój innowacyjnych rozwiązań.

1. **Wieloplatformowość –** oprogramowanie powinno być dostępne na różnych platformach w tym komputerach stacjonarnych, tabletach oraz smartfonach, aby umożliwić elastyczne korzystanie dla graczy.

2. Responsywność interfejsu – interfejs użytkownika powinien być responsywny, dostosowujący się do różnych rozmiarów ekranów.

3. Personalizacja doświadczeń graczy – system powinien umożliwiać częściową lub nawet pełną personalizacje doświadczeń graczy na podstawie ich preferencji, historii gier.

4. Integracja z nowoczesnymi technologiami – oprogramowanie powinno być zdolne do integracji z nowoczesnymi technologiami takimi jak sztuczna inteligencja czy też blockchain, aby wprowadzić innowacyjne funkcje i usługi.

5. Współpraca z dostawcami – system powinien umożliwiać łatwą integrację z różnymi dostawcami gier, usług płatniczych lub innych partnerów.

6. Zgodność z regulacjami branżowymi – oprogramowanie musi być zgodne z obowiązującymi przepisami i regulacjami dotyczącymi hazardu online.

2. Diagram przypadków użycia

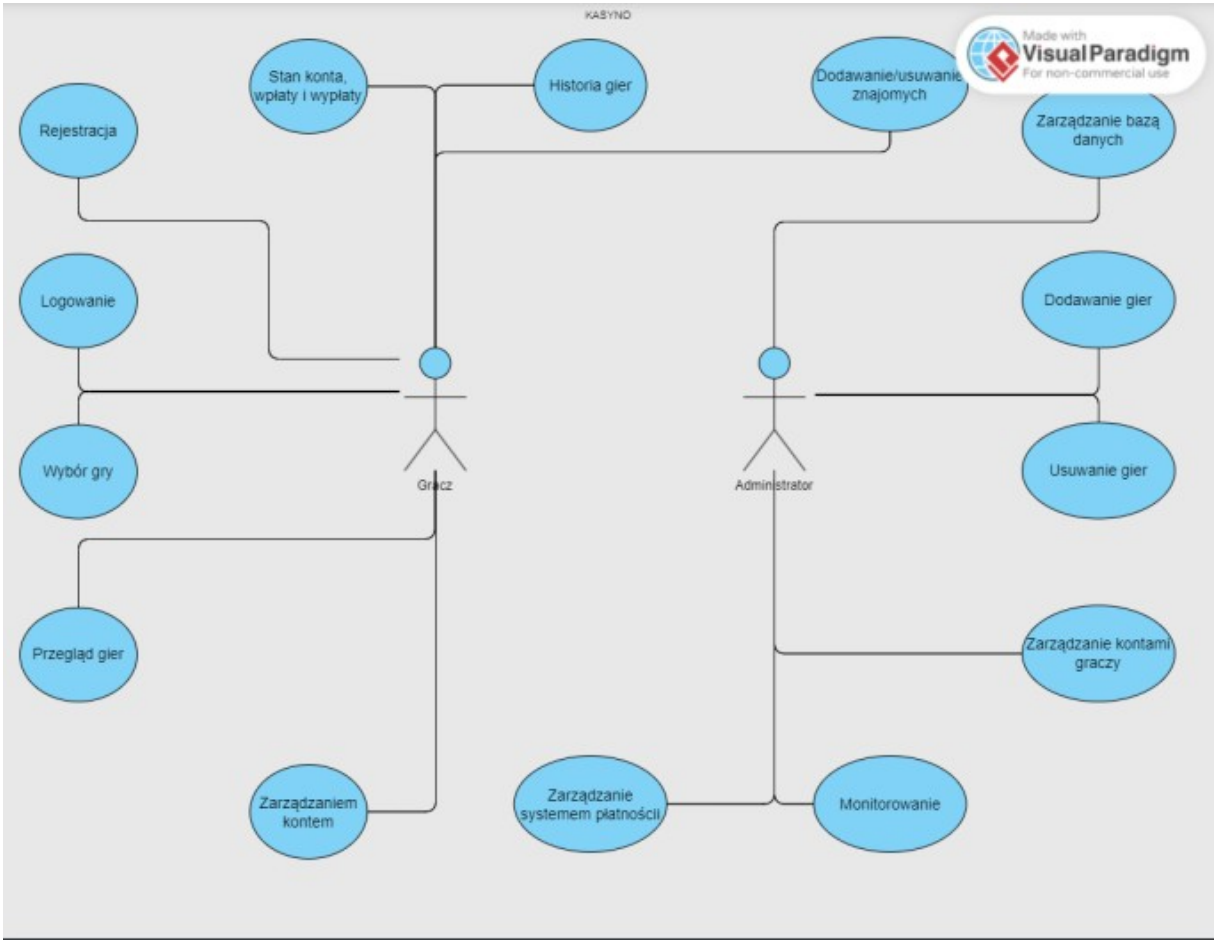


Diagram przypadków użycia, który widzimy, reprezentuje różne scenariusze interakcji użytkowników z aplikacją kasyna napisaną w języku Java. Oto krótki opis każdego przypadku użycia:

Rejestracja: Użytkownik może utworzyć nowe konto w kasynie.

Logowanie: Użytkownik może zalogować się na swoje konto.

Wybór gry: Po zalogowaniu użytkownik ma możliwość wybrania gry, w którą chce grać.

Przegląd gier: Użytkownik może przeglądać dostępne gry w kasynie.

Stan konta, wpłaty i wypłaty: Użytkownik może sprawdzić stan swojego konta i dokonać wpłat lub wypłat środków.

Historia gier: Użytkownik ma dostęp do historii swoich gier, aby śledzić swoje aktywności w kasynie. **in progress*

Zarządzanie kontem: Użytkownik może zarządzać swoim kontem, np. zmieniać ustawienia lub aktualizować dane osobowe. **in progress*

Zarządzanie systemem płatności: Użytkownik może zarządzać swoimi metodami płatności, dodawać nowe metody lub aktualizować istniejące.

Dla administratora aplikacji kasyna istnieją następujące przypadki użycia:

Dodawanie/Usuwanie użytkowników: Administrator może dodawać lub usuwać użytkowników.

Zarządzanie bazą danych: Administrator dba o utrzymanie i zarządzanie bazą danych, w której przechowywane są wszystkie dane użytkowników i gier.

Dodawanie gier: Administrator może wprowadzać nowe gry do oferty kasyna.

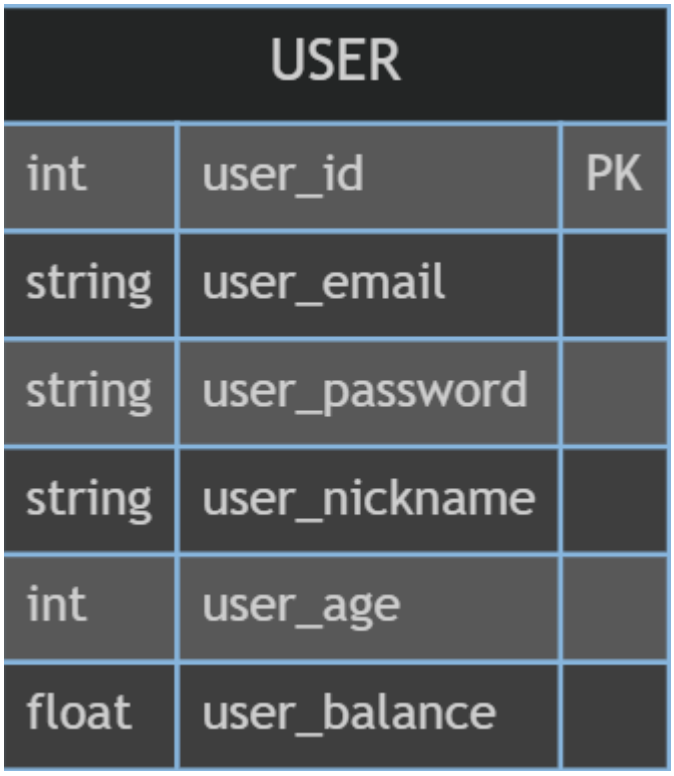
Usuwanie gier: Administrator może usuwać gry, które nie są już dostępne lub popularne.

Zarządzanie kontami graczy: Administrator ma możliwość zarządzania kontami graczy, w tym blokowania lub dezaktywowania kont.

Monitorowanie: Administrator ma możliwość monitorowania działalności w kasynie, w tym gier i transakcji finansowych.

Cały system jest zaprojektowany tak, aby umożliwić płynną interakcję między użytkownikami a aplikacją, a także umożliwić administratorowi zarządzanie kluczowymi aspektami aplikacji kasyna.

3. Diagram ERD



Obrazek przedstawia diagram ERD (Entity Relationship Diagram) dla tabeli o nazwie "**USER**".

Diagram składa się z sześciu atrybutów:

user_id (int) - Jest to klucz główny tabeli (PK), który jest unikatowym identyfikatorem dla każdego rekordu użytkownika. Typ danych to liczba całkowita.

user_email (string) - Ten atrybut przechowuje adres e-mail użytkownika. Typ danych to ciąg znaków.

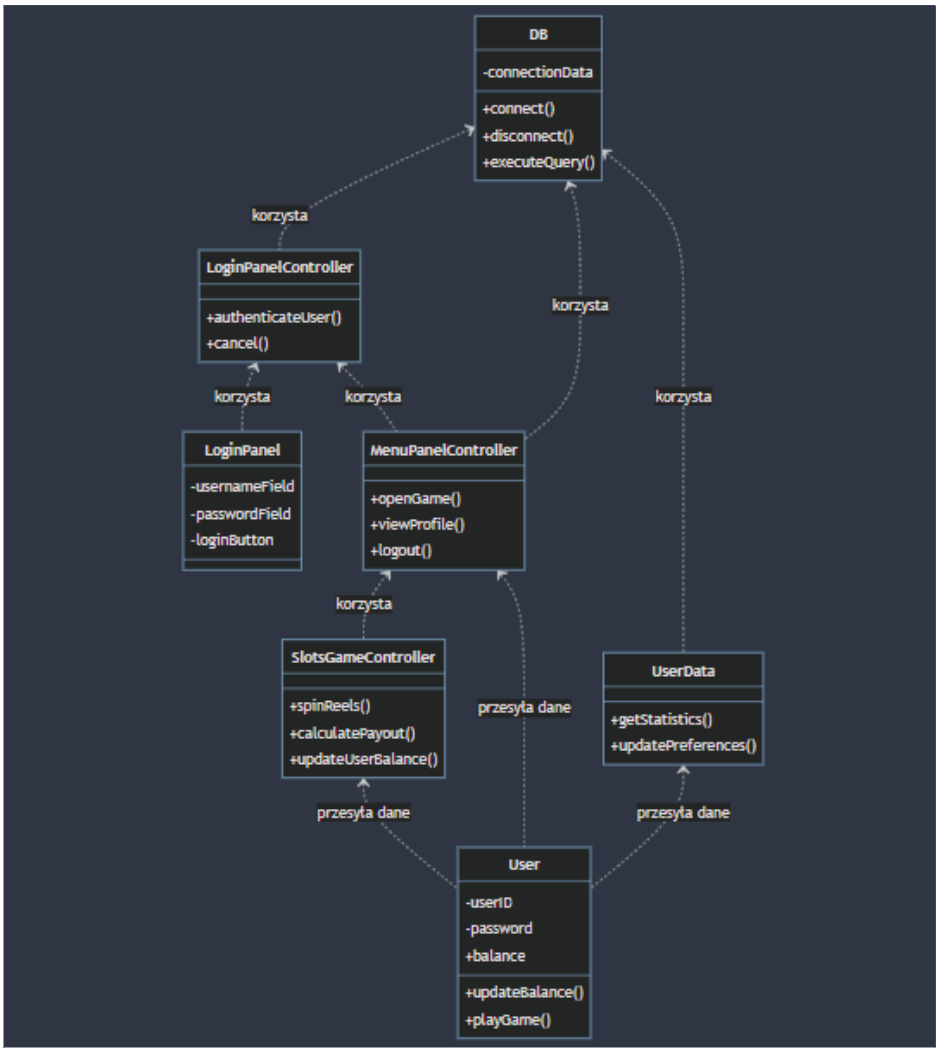
user_password (string) - Zawiera hasło użytkownika. Typ danych to ciąg znaków, który najprawdopodobniej jest zaszyfrowany dla bezpieczeństwa.

user_nickname (string) - To pseudonim użytkownika. Typ danych to ciąg znaków.

user_age (int) - Przedstawia wiek użytkownika jako liczbę całkowitą.

user_balance (float) - To pole wskazuje saldo użytkownika, prawdopodobnie na jakimś koncie lub w aplikacji. Typ danych to liczba zmiennoprzecinkowa, co pozwala na reprezentowanie wartości pieniężnych włączając grosze.

4. Diagram Klas



Klasa DB: Ta klasa jest odpowiedzialna za zarządzanie połączeniem z bazą danych. Zawiera metody *connect*(), *disconnect*(), *executeQuery*() oraz prywatne pola

przechowujące dane połączenia. Jest używana przez inne klasy kontrolerów do interakcji z bazą danych.

Klasa LoginPanel: Reprezentuje interfejs graficzny panelu logowania. Zawiera prywatne komponenty GUI, takie jak *usernameField*, *passwordField* i *loginButton*. Jest to klasa czysto wizualna, bez bezpośrednich zależności od innych klas.

Klasa LoginPanelController: Zarządza logiką autentykacji użytkownika. Zawiera metody authenticateUser() oraz cancel(). Ta klasa używa klasy DB do weryfikacji danych logowania, co jest zaznaczone jako zależność (strzałka przerywana).

Klasa MenuPanelController: Odpowiada za interakcję z głównym menu aplikacji. Zawiera metody takie jak openGame(), viewProfile(), logout(). Podobnie jak LoginPanelController, ma zależność od klasy DB.

Klasa SlotsGameController: Zarządza mechaniką gry w sloty. Zawiera metody: spinReels(), calculatePayout(), updateUserBalance(). Ta klasa używa obiektów z klasy User do aktualizacji salda oraz ma zależność od klasy DB.

Klasa User: Reprezentuje gracza. Zawiera pola takie jak userID, password, balance oraz metody updateBalance(), playGame(). Ma asocjację z klasą UserData, co wskazuje na to, że każdy użytkownik ma powiązane dane profilu.

Klasa UserData: Przechowuje dane profilu użytkownika, takie jak statystyki gier i ustawienia. Zawiera metody getStatistics(), updatePreferences().

5. Implementacja kodu oraz przykłady użycia.

Połączenie klient – serwer

Klasa ClientHandlerCallable

```
private Socket clientSocket;

1 usage  new *
public ClientHandlerCallable(Socket socket) { this.clientSocket = socket; }

new *
@Override
public ServerResponse call() {
    try (ObjectInputStream objectInputStream = new ObjectInputStream(clientSocket.getInputStream());
         ObjectOutputStream objectOutputStream = new ObjectOutputStream(clientSocket.getOutputStream())) {
        ClientRequest clientRequest = (ClientRequest) objectInputStream.readObject();
        ServerResponse serverResponse = new ServerResponse();
        if(clientRequest.getAction().equals("login")) {
            LoginData loginData = (LoginData) clientRequest.getData();
            User user = UserData.getAuthenticatedUser(loginData.getEm(), loginData.getPas());
            serverResponse.setData(user);
        } else if (clientRequest.getAction().equals("register")) {

        }
        objectOutputStream.writeObject(serverResponse);
        return serverResponse;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    } finally {
        try {
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Klasa ClientRequest

```java
    private String action; // getUserBasicData
    3 usages
    private Serializable data;
    2 usages
    private Integer privateToken;

    new *
    public ClientRequest() {
    }

    1 usage  new *
    public ClientRequest(String action, Serializable data) {
        this.action = action;
        this.data = data;
    }

    no usages  new *
    public ClientRequest(String action, Serializable data, Integer privateToken) {
        this.action = action;
        this.data = data;
        this.privateToken = privateToken;
    }

    2 usages  new *
    public String getAction() { return action; }

    1 usage  new *
    public Serializable getData() { return data; }

    no usages  new *
    public Integer getPrivateToken() { return privateToken; }
}
```

Klasa LoginData

```
new *
public class LoginData implements Serializable {
    3 usages
    String em;

    1 usage  new *
    public String getEm() {
        return em;
    }

    no usages  new *
    public void setEm(String em) {
        this.em = em;
    }

    1 usage  new *
    public String getPas() {
        return pas;
    }

    no usages  new *
    public void setPas(String pas) {
        this.pas = pas;
    }

    1 usage  new *
    public LoginData(String em, String pas) {
        this.em = em;
        this.pas = pas;
    }
```

Klasa RequestSender

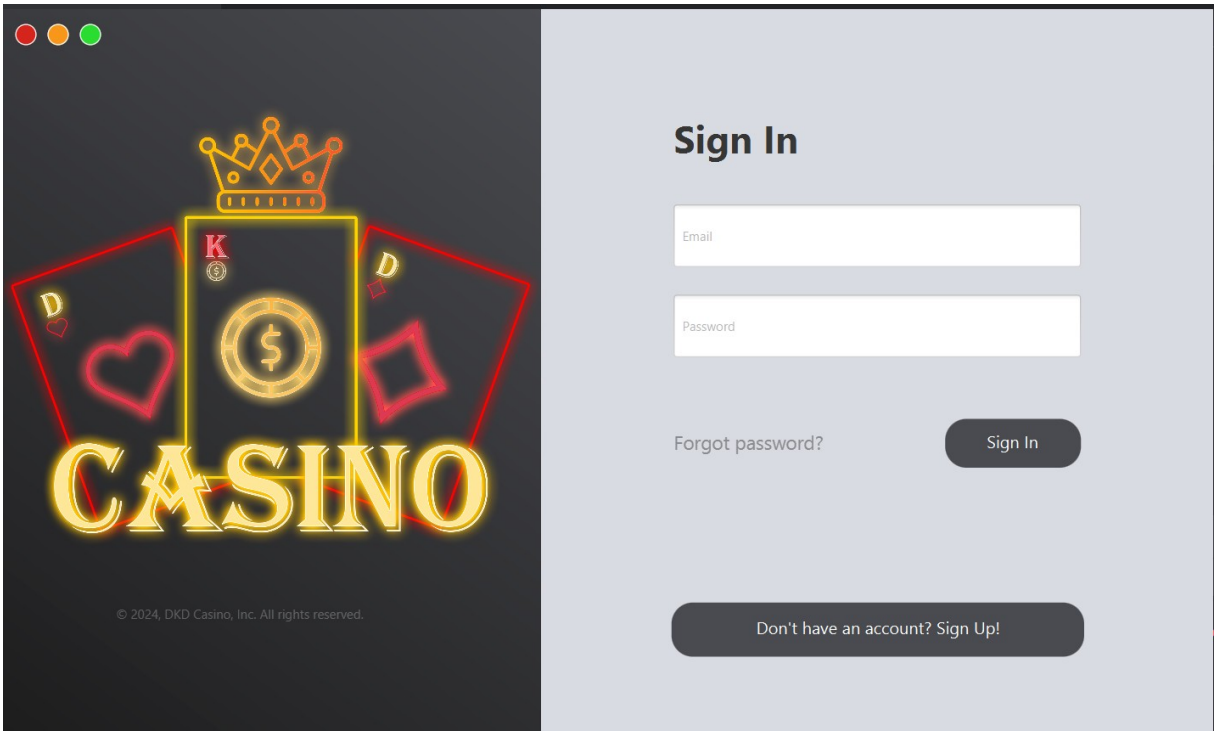
```java
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

new *
public class RequestSender {
    new *
    public RequestSender() {
    }

    1 usage  new *
    public static ServerResponse sendRequest(ClientRequest request) {
        try (Socket socket = new Socket( host: "127.0.0.1",  port: 12345);
             ObjectOutputStream outputStream = new ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream inputStream = new ObjectInputStream(socket.getInputStream())) {

            outputStream.writeObject(request);
            return (ServerResponse) inputStream.readObject();
        } catch (ClassNotFoundException | IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Klasa Server

```java
new *
public class Server {
    new *
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        ExecutorService executorService = Executors.newFixedThreadPool( nThreads: 10);
        try {
            serverSocket = new ServerSocket( port: 12345);
            while (true) {
                Socket clientSocket = serverSocket.accept();
                Callable<ServerResponse> clientTask = new ClientHandlerCallable(clientSocket);
                executorService.submit(clientTask);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (serverSocket != null && !serverSocket.isClosed()) {
                try {
                    serverSocket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            executorService.shutdown();
        }
    }
```

Klasa ServerResponse

```java
public class ServerResponse implements Serializable {
    no usages
    @Serial
    private static final long serialVersionUID = 1L;

    2 usages
    private Serializable data;

    new *
    public ServerResponse() {
    }

    1 usage  new *
    public Serializable getData() {
        return data;
    }

    1 usage  new *
    public void setData(Serializable data) {
        this.data = data;
    }
}
```



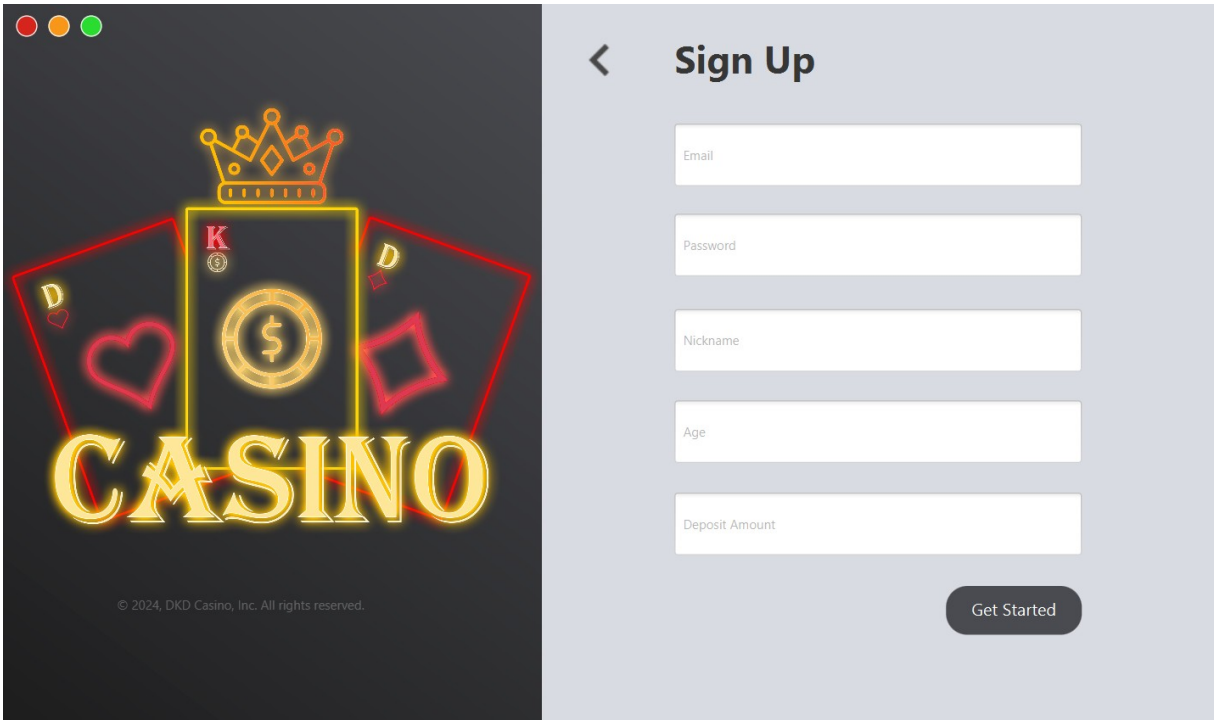
Pierwsze okno aplikacji – system logowania.

Poniższe zdjęcie implementuje połączenie aplikacji z bazą danych.

```
1   package com.main.casinoapp;
2
3   import ...
6
    new *
7   public class db {
        4 usages  new *
8       public static Connection mycon() {
9
10          Connection con = null;
11
12          try {
13              Class.forName( className: "com.mysql.jdbc.Driver");
14
15              con = DriverManager.getConnection( url: "jdbc:mysql://127.0.0.1:3306/casio", user: "root", password: "");
16          } catch (ClassNotFoundException | SQLException e) {
17              System.out.println(e);
18          }
19          return con;
20      }
21  }
22
23
```

```
if (event.getSource().equals(btnRegister)) {
    String nickname = tfNickname.getText();
    String email = tfEmailReg.getText();
    String pass = tfPassReg.getText();
    String age = tfAge.getText();
    String bal = tfBalance.getText();

    if(nickname != null && email != null && pass != null && age != null && bal != null) {
        try {
            java.sql.Statement s = db.mycon().createStatement();
            s.executeUpdate( sql: " INSERT INTO user (user_email,user_password,user_nickname,user_age,user_balance)
                    + "VALUES ('" + email + "','" + pass + "','" + nickname + "','" + age + "','" + bal + "') ");

            tfNickname.clear();
            tfEmailReg.clear();
            tfPassReg.clear();
            tfAge.clear();
            tfBalance.clear();
            pnlSignIn.toFront();
        } catch (Exception e) {
            System.out.println(e);
        }
    } else {

    }
```

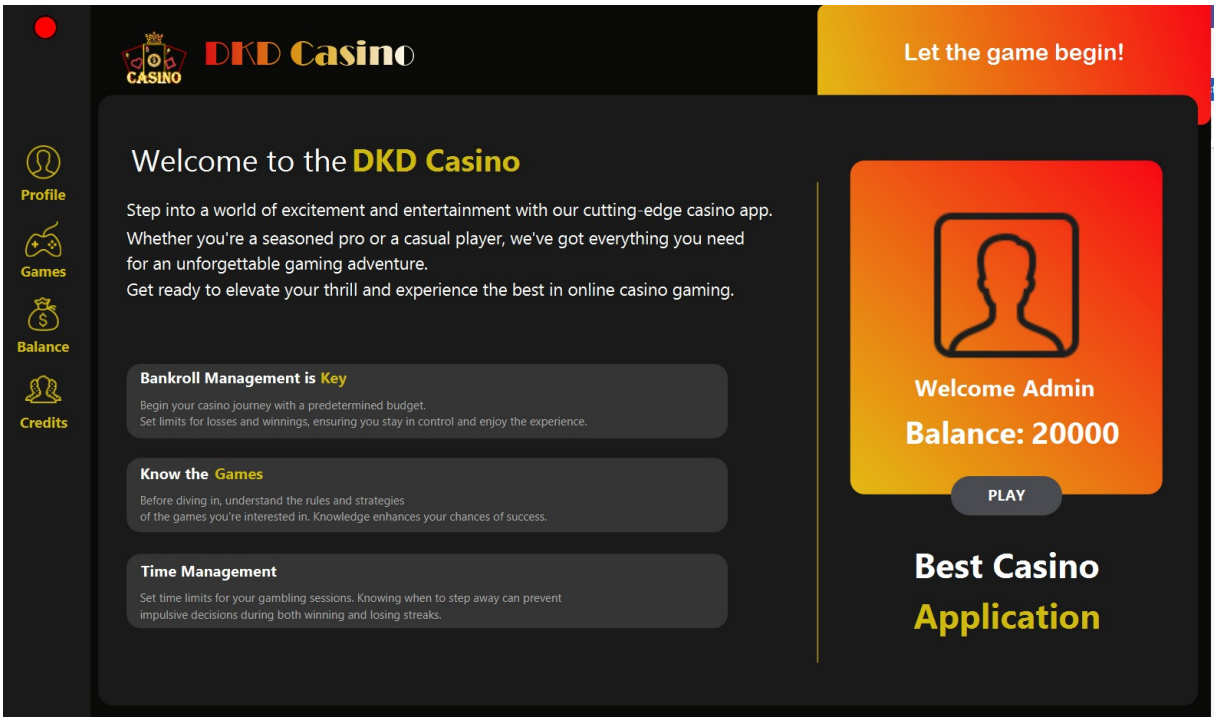
Funkcja rejestracji i połączenia z bazą danych.



Panel rejestracji. Oba panele połączone z bazą danych.

```
1 usage  new *
public static User getAuthenticatedUser(String email, String password, Connection connection) {
    try {
        String sql = "SELECT * FROM user WHERE user_email=? AND user_password=?";
        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString( parameterIndex: 1, email);
        preparedStatement.setString( parameterIndex: 2, password);

        ResultSet resultSet = preparedStatement.executeQuery();

        if (resultSet.next()) {
            int userId = resultSet.getInt( columnLabel: "user_id");
            int userBalance = resultSet.getInt( columnLabel: "user_balance");
            String userNickname = resultSet.getString( columnLabel: "user_nickname");
            int userAge = resultSet.getInt( columnLabel: "user_age");
            User authenticatedUser = new User(userId, email, password);
            authenticatedUser.setBalance(userBalance);
            authenticatedUser.setNickname(userNickname);
            authenticatedUser.setAge(userAge);
            return authenticatedUser;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
 private void handleButtonAction(ActionEvent event) {
     if (event.getSource().equals(btnSignUp)) {
         pnlSignUp.toFront();
     }

     if (event.getSource().equals(btnSignIn)) {

         String em = tfEmail.getText();
         String ps = tfPass.getText();

         try {
             User authenticatedUser = UserData.getAuthenticatedUser(em, ps, con);

             if (authenticatedUser != null) {
                 User.setCurrentUser(authenticatedUser);

                 closeCurrentStage();
                 openMenuView();
             } else {
                 System.exit( status: 0);
             }
         } catch (IOException e) {
             e.printStackTrace();
         }

     }
```

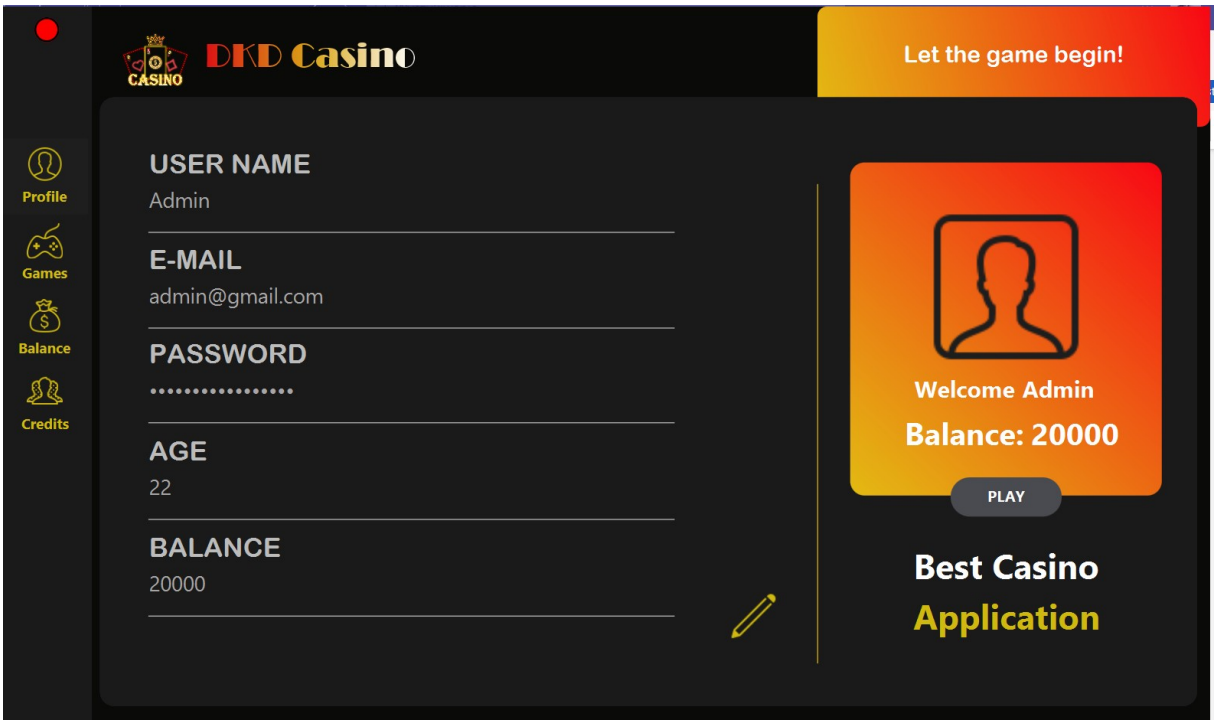
Funkcja logowania oraz pobierania danych o zalogowanym użytkowniku.



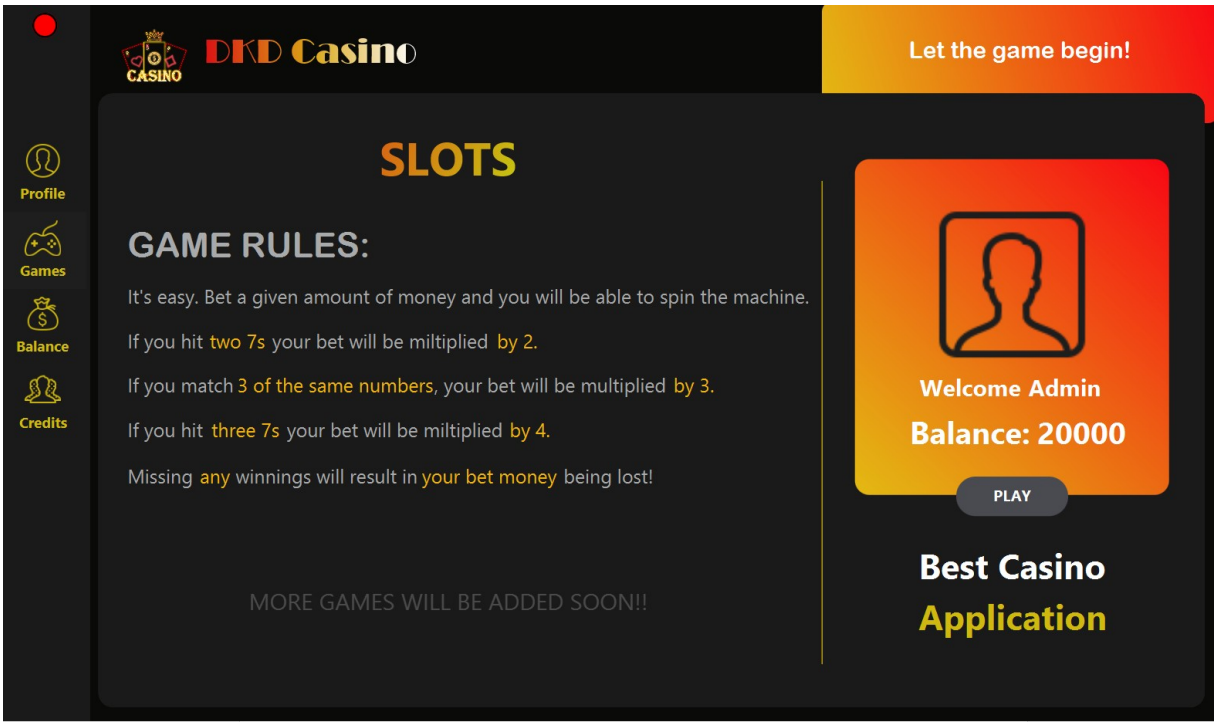
Główne menu aplikacji.

```
public User(int userId, String email, String password) {
    this.userId = userId;
    this.email = email;
    this.password = password;
    this.balance = balance;
    this.nickname = nickname;
    this.age = age;
}
3 usages  new *
public int getBalance() { return balance; }

6 usages  new *
public String getNickname() { return nickname; }

1 usage  new *
public int getAge() { return age; }

1 usage  new *
public void setAge(int age) {
    this.age = age;
}
1 usage  new *
public void setNickname(String nickname) { this.nickname = nickname; }

2 usages  new *
public void setBalance(int balance) { this.balance = balance; }

1 usage  new *
public static void setCurrentUser(User user) { currentUser = user; }

14 usages  new *
public static User getCurrentUser() { return currentUser; }
```

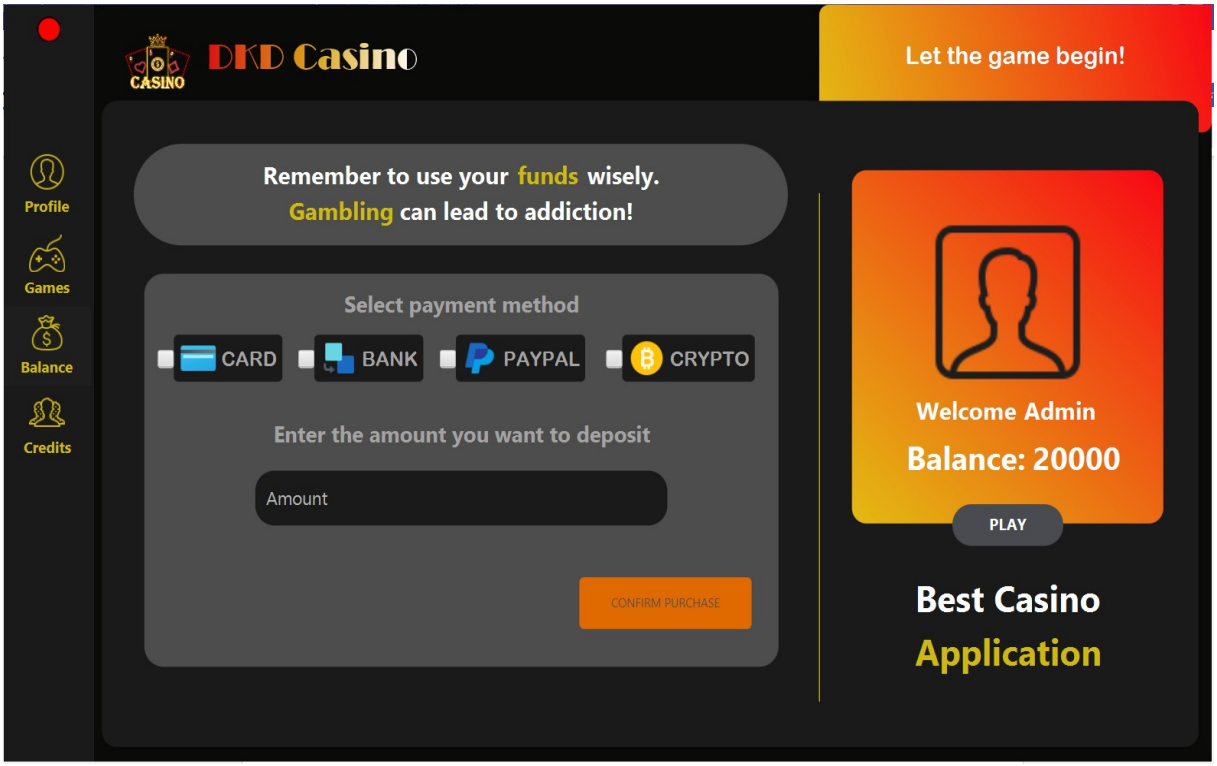
Klasa zwracająca aktualnie zalogowanego użytkownika.



Okno profilowe użytkownika.



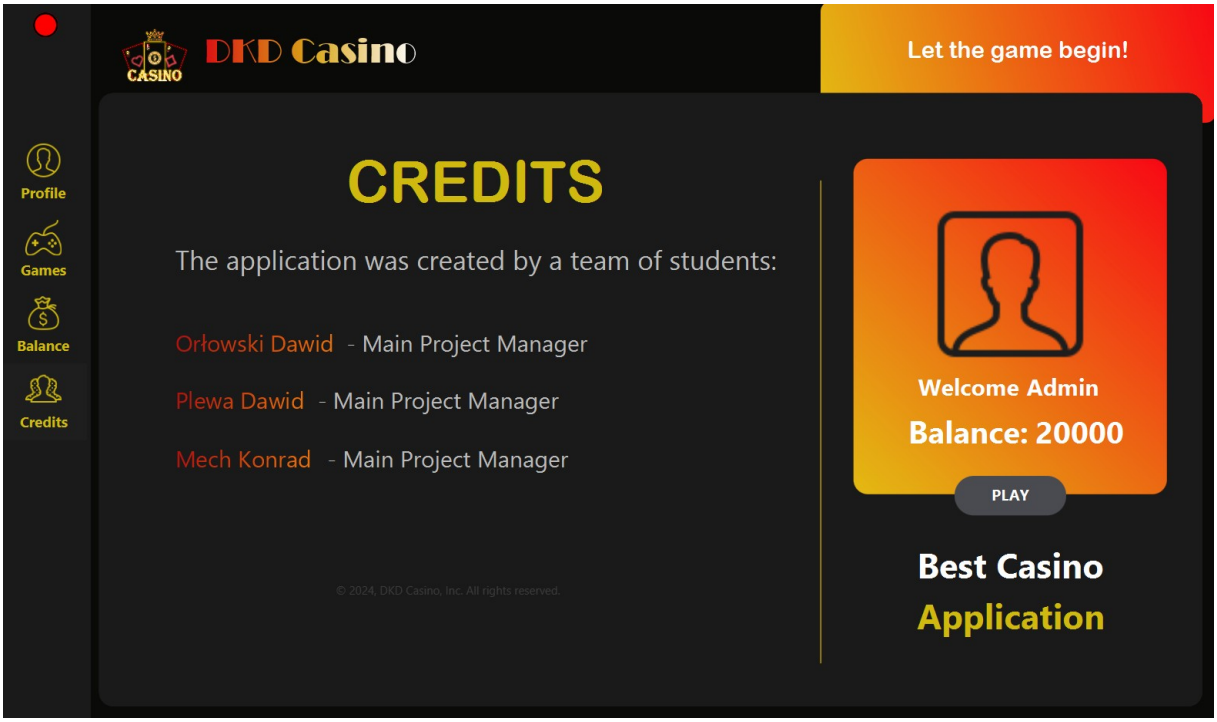
Okno opisu gier.



Portfel użytkownika.

```
2 usages  new *
public static void updateBalanceInDatabase(int userId, int newBalance) {
    String sql = "UPDATE user SET user_balance = ? WHERE user_id = ?";
    try (PreparedStatement preparedStatement = db.mycon().prepareStatement(sql)) {
        preparedStatement.setInt( parameterIndex: 1, newBalance);
        preparedStatement.setInt( parameterIndex: 2, userId);

        int affectedRows = preparedStatement.executeUpdate();
        if (affectedRows == 0) {
            throw new SQLException("Aktualizacja salda nie powiodła się, żaden wiersz nie został zmieniony.");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

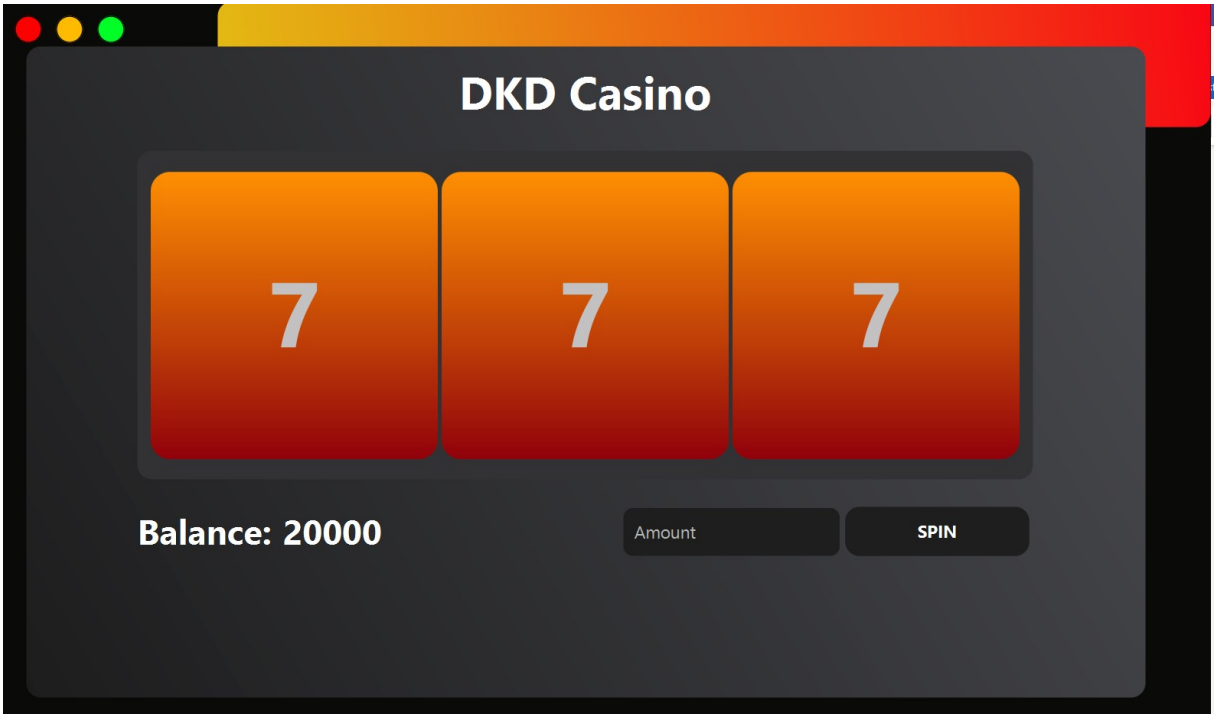
Funkcja uaktualniania bilansu użytkownika.



Okno twórców aplikacji.

```
private void handleSpinEvent(ActionEvent evt) {
    int amount = Integer.valueOf(tfAmount.getText());

    if((tfAmount.getText() != null) && (amount != 0)) {
        int slot1 = (int)(Math.random() * 9) + 1;
        int slot2 = (int)(Math.random() * 9) + 1;
        int slot3 = (int)(Math.random() * 9) + 1;

        lbl1.setText(String.valueOf(slot1));
        lbl2.setText(String.valueOf(slot2));
        lbl3.setText(String.valueOf(slot3));

        if (slot1 == 7 && slot2 == 7 && slot3 == 7){
            System.out.println("Trafiles trzy 7!!!");
            coins += 4 * amount;
        }

        else if (slot1 == 7 && slot2 == 7 || slot1 == 7 && slot3 == 7 || slot2 == 7 && slot3 == 7 ){
            System.out.println("Trafiles dwie 7!");
            coins += 2 * amount;
        }

        else if (slot1 == slot2 && slot1 == slot3){
            System.out.println("Trafiles 3 takie same liczby!");
            coins += 3 * amount;
        }

        else {
            System.out.println("Nic nie trafiles!!! :(");
            coins += 0;
        }
    }
```

Funkcja gry Slots



System gry – podajemy ilość pieniędzy, za jaką chcemy zakręcić maszyną.

Po zakręceniu nasze saldo maleje w momencie, gdy nic nie wygramy.

Funkcja uaktualniania bilansu użytkownika.

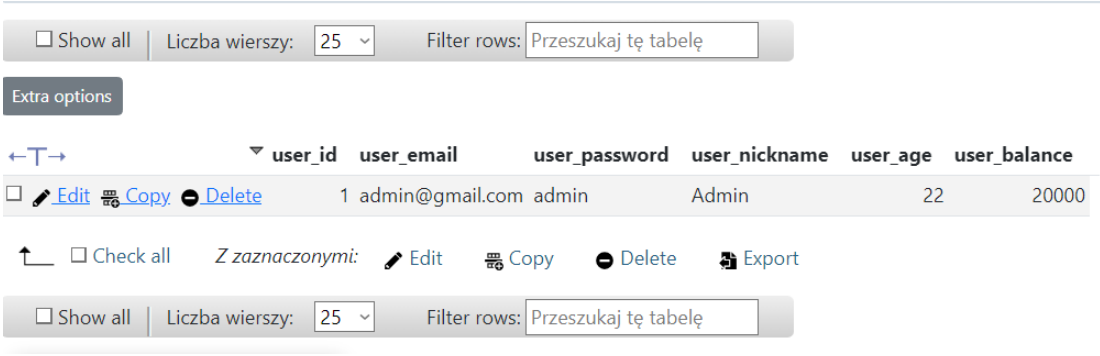


Tabela użytkownika zawarta w bazie danych.

6. WNIOSKI

Projekt "Tworzenie oprogramowania dla systemu informatycznego kasyno" obejmuje tworzenie aplikacji kasyna online. Cechuje się ona innowacyjnym interfejsem, zaawansowanymi funkcjami bezpieczeństwa, i elastyczną architekturą.

Kluczowe aspekty projektu to:

Funkcjonalność: Aplikacja zawiera różne funkcje, w tym system zarządzania portfelem, panel użytkownika, i różne gry kasynowe.

Bezpieczeństwo: Wdrożone zostały zaawansowane technologie szyfrowania i autentykacji, zapewniające bezpieczeństwo transakcji i danych użytkowników.

Wieloplatformowość: Oprogramowanie jest dostępne na różnych urządzeniach, co zapewnia elastyczność w korzystaniu.

Skalowalność: System jest przygotowany do obsługi rosnącej liczby użytkowników i rozszerzania oferty gier.