**Universitatea Tehnică ‚,Gheorghe Asachi” Iași Facultatea de Automatică și Calculatoare Iași Specializarea:** Calculatoare și Tehnologia Informației

**Disciplina:** Ingineria Programării

**Casino Extravaganza App**

**Coordonator,**

**Prof. Florin Leon**

**Studenți,**

**Fărcăș Cosmin Cătălin, Grupa 1308B**

**Gălbează Ciprian, Grupa 1308B**

**Ohriniuc Claudiu Constantin, Grupa 1308B**

**2023**

|  |  |
| --- | --- |
| **Cuprins** |  |
| **1.SRS** | **3** |
| **I. Introducere** | **3** |
| A.Motivul |  |
| B.Scopul |  |
| C.Definiț ii, acronime și abrevieri |  |
| D.Referințe |  |
| **II. Descrierea** | **4** |
| A.Perspectiva aplicației |  |
| B.Funcțiile aplicației |  |
| C.Clasa utilizator |  |
| D.Constrângeri generale |  |
| E.Documentația utilizatorului |  |
| **III. Cerințe specifice** | **5** |
| A.Interfața cu utilizatorul |  |
| B.Componente hardware |  |
| C.Componente software |  |
| **2.Modul de utilizare a programului** | **6** |
| **3.Diagrame UML** | **10** |
| **4.Exemple de execuție ale aplicației** | **15** |
| **Anexa 1: Metode pentru realizarea conexiunii la baza de date** | **15** |
| **Anexa 2: Etape de testare** | **19** |

2

**Documentul specificațiilor cerințelor software**

**1.Introducere**

**1.1 Motivul**

Motivația din spatele acestui proiect este crearea unei aplicații care permite jucătorilor să participe la diferite jocuri de cazinou, cum ar fi Blackjack. Aplicația le oferă utilizatorilor o modalitate convenabilă și distractivă de a experimenta atmosfera unui cazinou fără a părăsi comfortul casei lor.

**1.2 Scopul**

Scopul acestui SRS este de a descrie funcționalitatea aplicației de cazinou. Funcționalitatea principală a aplicației constă în oferirea de jocuri de cazinou, permițând jucătorilor să se angajeze în jocuri, să facă pariuri și să urmărească rezultatele acestora.

**1.3 Definiții, acronime și abrevieri**

➢ Șablonul Proxy: acest model de proiectare implică utilizarea unui server proxy ca intermediar între jucătorii interni și jocurile externe la care aceștia doresc să participe.

➢ Șablonul Factory: acest model de proiectare este o modalitate de a crea obiecte, cum ar fi jucători sau jocuri, fără a expune logica de creare specifică în cadrul clasei client.

➢ Șablonul State: acest model de proiectare permite unui obiect să își modifice comportamentul atunci când starea sa internă se schimbă. În acest caz, obiectul va apărea ca și cum și-ar schimba clasa. Acest model este utilizat în mod obișnuit pentru a implementa mașini de stare în cadrul obiectelor.

➢ Joc de cazinou: un tip de joc în care jucătorii pariază pe rezultate diverse, de obicei în speranța de a câștiga bani.

➢ Jucător: o persoană care participă la un joc în cadrul aplicației.

➢ Pariu: suma de bani pe care un jucător o riscă în speranța de a câștiga mai mult.

**1.4 Referințe**

* + <https://www.stirilekanald.ro/regulile-jocului-de-barbut-20098553>
  + <https://www.cazinoonline.com/blackjack/reguli/>
  + <https://www.telegraph.co.uk/betting/casino-guides/poker/how-to-play-video-poker-slots/>
  + https://www.casinowow.com/guides/online-3-reel-classic-slots-basics

1. **Descrierea**

**2.1 Perspectiva aplicației**

Proiectul oferă jucătorului o experiență unică și modernă de a se bucura de jocuri de cazino. Acesta nu este un concept nou, dar este o implementare originala avand doar ca baza structura deja existentă a altor aplicații. Se pune accent pe diverse jocuri, inclusiv poker slots, sloturi, jocuri cu zaruri, jocuri "mai mare sau mai mic", "arunca cu banul" și blackjack. Include și posibilitatea de autentificare pentru o experiență personalizată.

**2.2 Funcțiile aplicației**

Funcțiile aplicației constau în înregistrarea unui utilizator, logarea acestuia, adaugarea de bani in cont si multiplicarea sau pierderea lor prin jocurile existente.

**2.3 Clasa utilizator**

Utilizatorii din această aplicație sunt jucători ai unui cazinou virtual. Informațiile lor, inclusiv numele de utilizator, parola criptată și balanța jocului, sunt stocate într-o bază de date SQLite. Există constrângeri asupra unicității numelui de utilizator și a lungimii parolei pentru a asigura securitatea. Parolele sunt criptate folosind o metodă de hash sigură pentru a proteja datele sensibile ale utilizatorului.

**2.4 Constrângeri generale**

Fiind o aplicație de dimensiuni reduse, aceasta nu are limitări de memorie RAM sau ROM. Limitările de care are nevoie aplicația să funcționeze sunt sistemul de operare care trebuie să fie Windows, iar în partea de periferice este nevoie de un mouse și o tastatură.

**2.5 Documentația utilizatorului**

În interiorul interfeței este un buton special creat pentru utilizator care ii deschide o fereastră unde ii este prezentată funcționalitatea aplicației, impreuna cu toate regulile jocurilor.

**3. Cerințe specifice**

**3.1 Interfața cu utilizatorul**

Aplicația oferă o interfață grafică atractivă și intuitivă, realizată prin intermediul platformei Windows Forms App. Aceasta include diferite ecrane pentru fiecare joc de noroc disponibil în cadrul aplicației, precum și un ecran de autentificare și unul de înregistrare pentru noii utilizatori. Interfața permite utilizatorilor să navigheze cu ușurință între diferitele jocuri, să-și vizualizeze soldul actual și să efectueze operațiuni precum plasarea de pariuri sau retragerea câștigurilor.

**3.2 Componente hardware**

Pentru dezvoltarea și testarea aplicației au fost utilizate 3 laptopuri cu specificații medii, care rulează sistemul de operare Windows. Aceasta nu exclude posibilitatea utilizării aplicației pe alte sisteme de operare sau dispozitive, atâta timp cât acestea suportă .NET Framework.

**3.3 Componente software**

Pe partea de back-end, am folosit limbajul de programare C#, cunoscut pentru puternicele sale funcționalități orientate pe obiect, care ne-au permis să abordăm problema într-un mod modular și ușor de înțeles. Pentru gestionarea datelor utilizatorilor și a tranzacțiilor, am folosit SQLite, un sistem de gestiune a bazelor de date relaționale, care oferă un grad ridicat de fiabilitate și performanță.

Aplicația a fost dezvoltată și testată pe sistemul de operare Windows, folosind mediul de dezvoltare Microsoft Visual Studio 2022 Community. Acesta ne-a oferit o gamă largă de instrumente și facilități, precum un editor de cod avansat, facilități de debugging și un set complet de instrumente pentru proiectarea interfeței cu utilizatorul.

Aplicația include următoarele jocuri de noroc: Coin Flip, în care utilizatorii pariază pe rezultatul unei aruncări de monedă; Blackjack, un joc popular de cărți în care scopul este să obții un scor cât mai aproape de 21, fără a-l depăși; Slots, un joc de noroc care simulează funcționarea unei mașini cu sloturi; HigherLower , unde trebuie calculate daca urmatoarea carte are sanse sa fie mai mica sau mai mare; Poker Slots , in care jucatorul incearca sa aiba o mana cat mai buna impotriva casei; si Dices in care jucatorul arunca zaruri si incearca sa castige , avand mai multe modalitati de a juca.

Fiecare joc are propriul său set de reguli și interfață grafică, însă toate împărtășesc o infrastructură comună, care include sistemul de autentificare a utilizatorilor, gestionarea tranzacțiilor și interfața de navigare principală a aplicației.

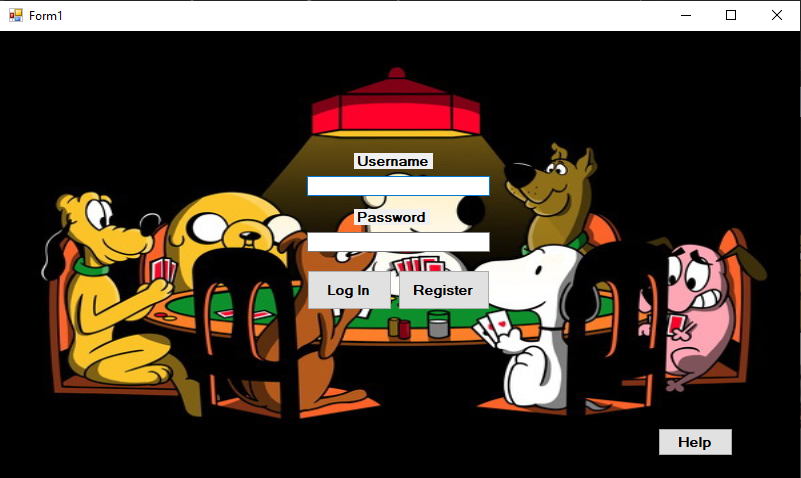
**Modul de utilizare a programului**

Aplicația își propune să ofere o experiență captivantă și intuitivă utilizatorilor săi, printr-un mediu de joc variat și distractiv. Meniul simplu și clar facilitează navigarea rapidă prin diferitele jocuri de noroc disponibile, inclusiv coin flip, slot machines, poker și multe altele. În plus, exista un buton de help ce contine informatii relevante pentru a ajuta utilizatorii să înțeleagă mai bine regulile și șansele lor de câștig.

**Pagina Start**

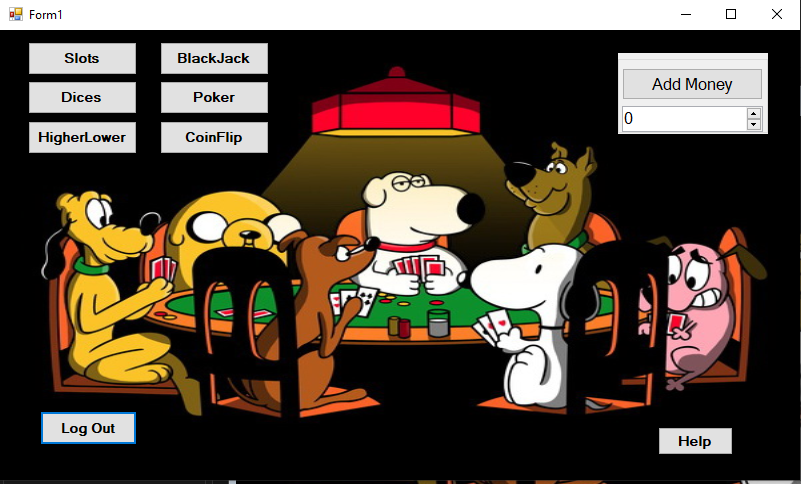
Cand utilizatorul deschide aplicatia este intampinat de pagina de start in care are optiunea de a se autentifica sau de a crea un nou cont pentru a putea avea acces la jocurile disponibile.

Pentru a se autentifica utilizatorul trebuie sa aleaga un nume unic si o parola , iar cu aceste date va urma sa dea log in.

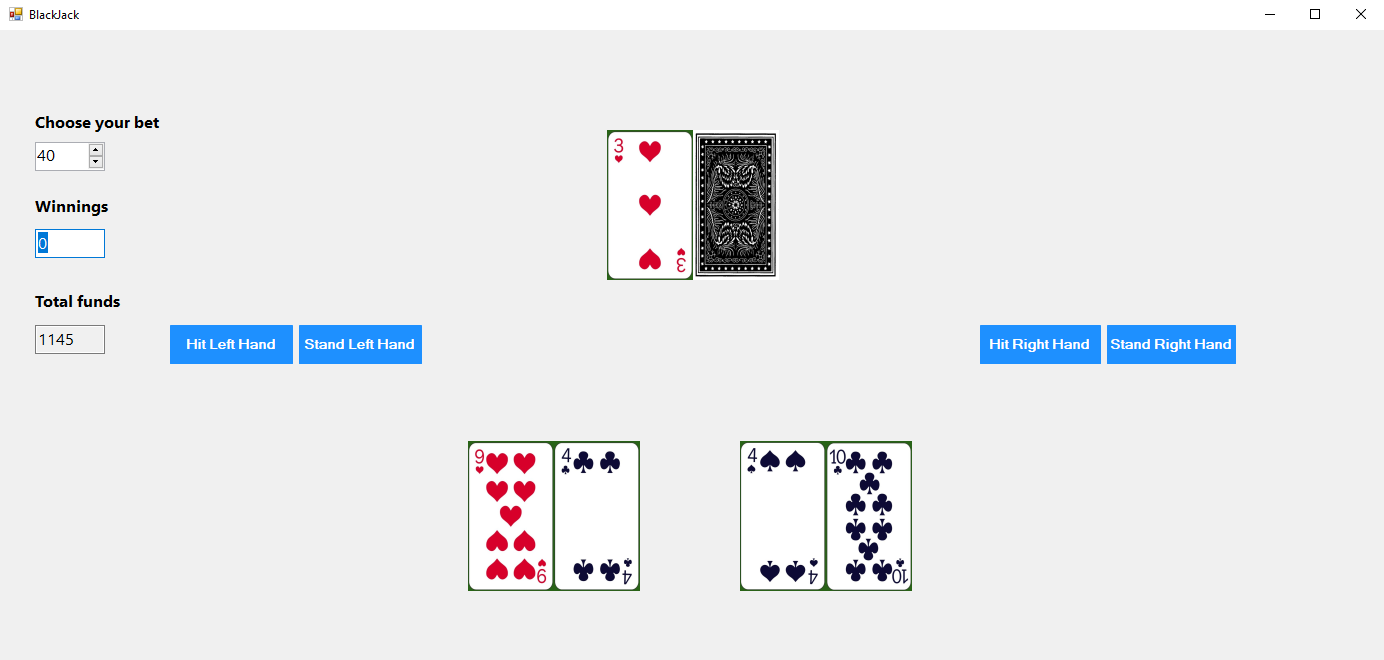


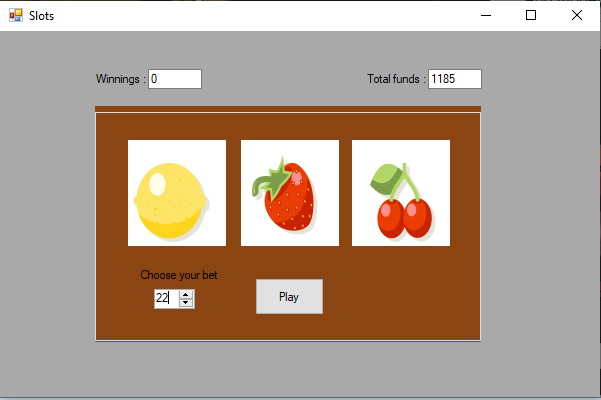
**Pagina principală**

In pagina principala utilizatorul va avea acces la jocuri de noroc diverse precum slots, blackjack , poker, iar acestea nu sunt singurile optiuni. Clientul va avea de asemenea acces sa adauge bani in cont ,dar si sa isi dea log out.

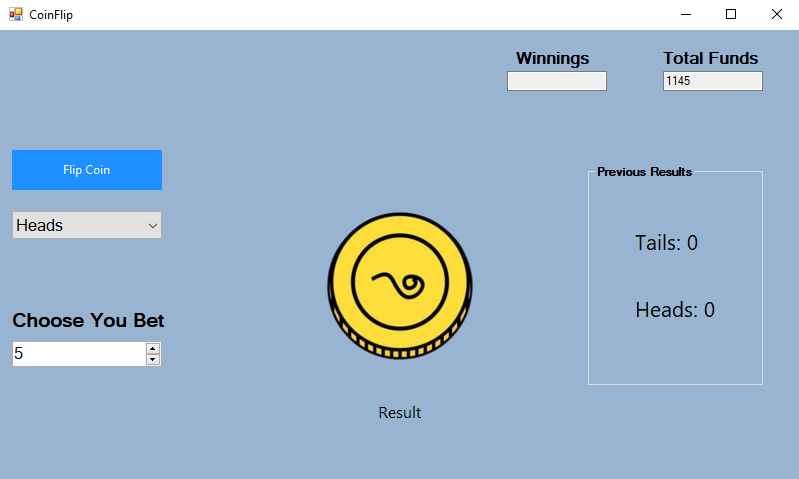


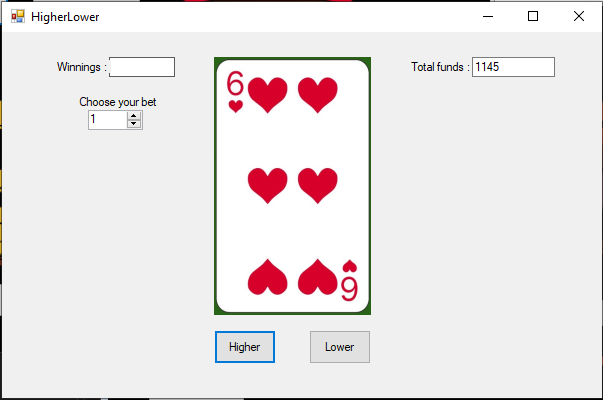
**Pagini din jocuri**

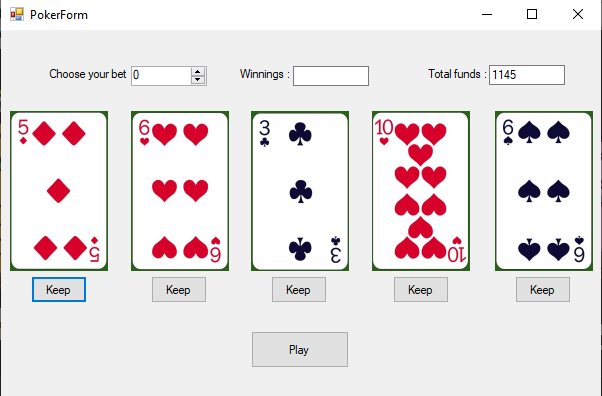






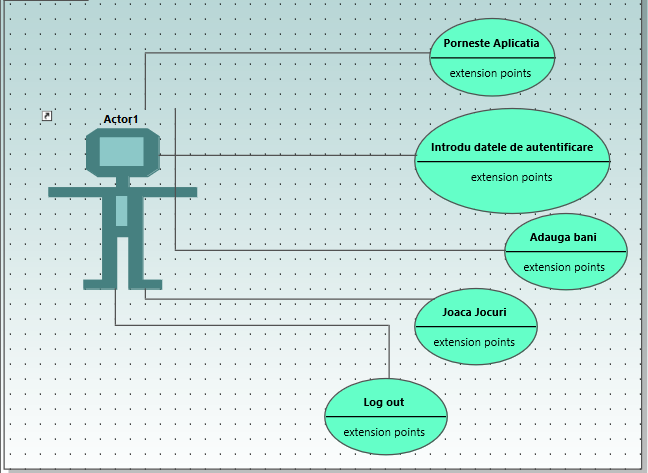




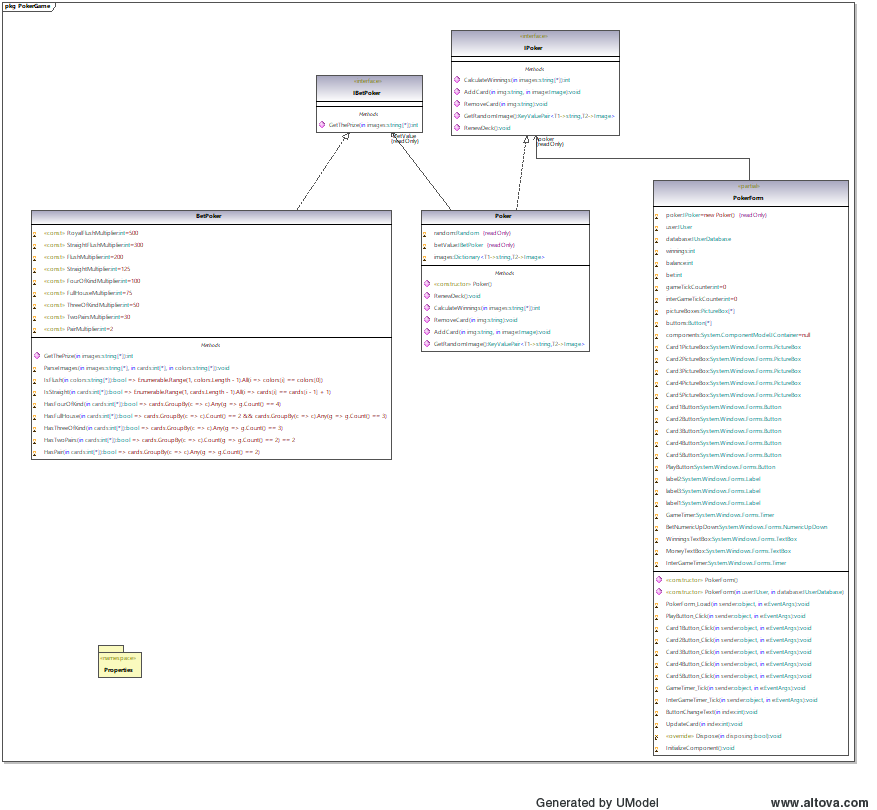


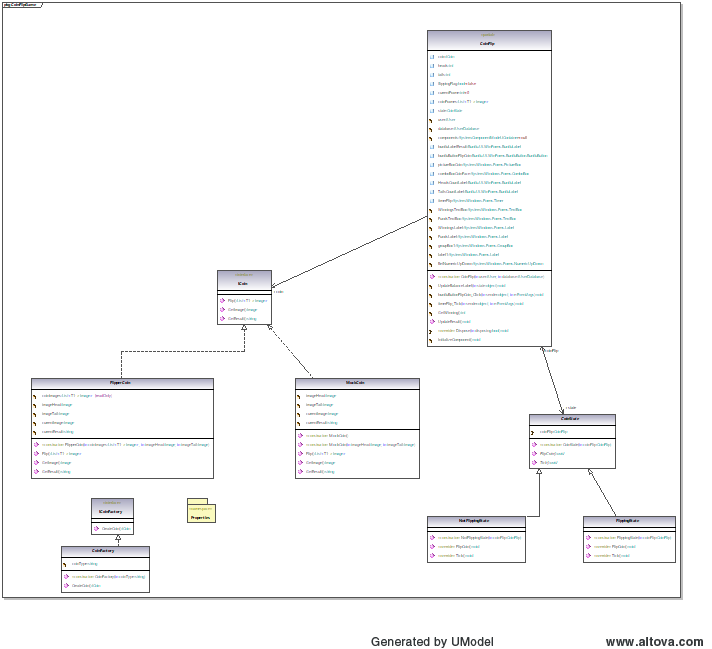
**Diagrame UML**

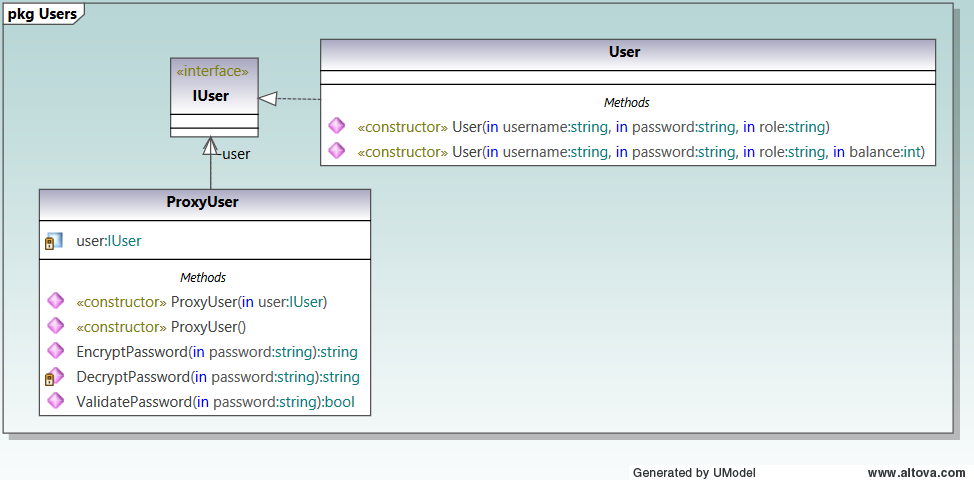
**Diagrama de cazuri de utilizare**

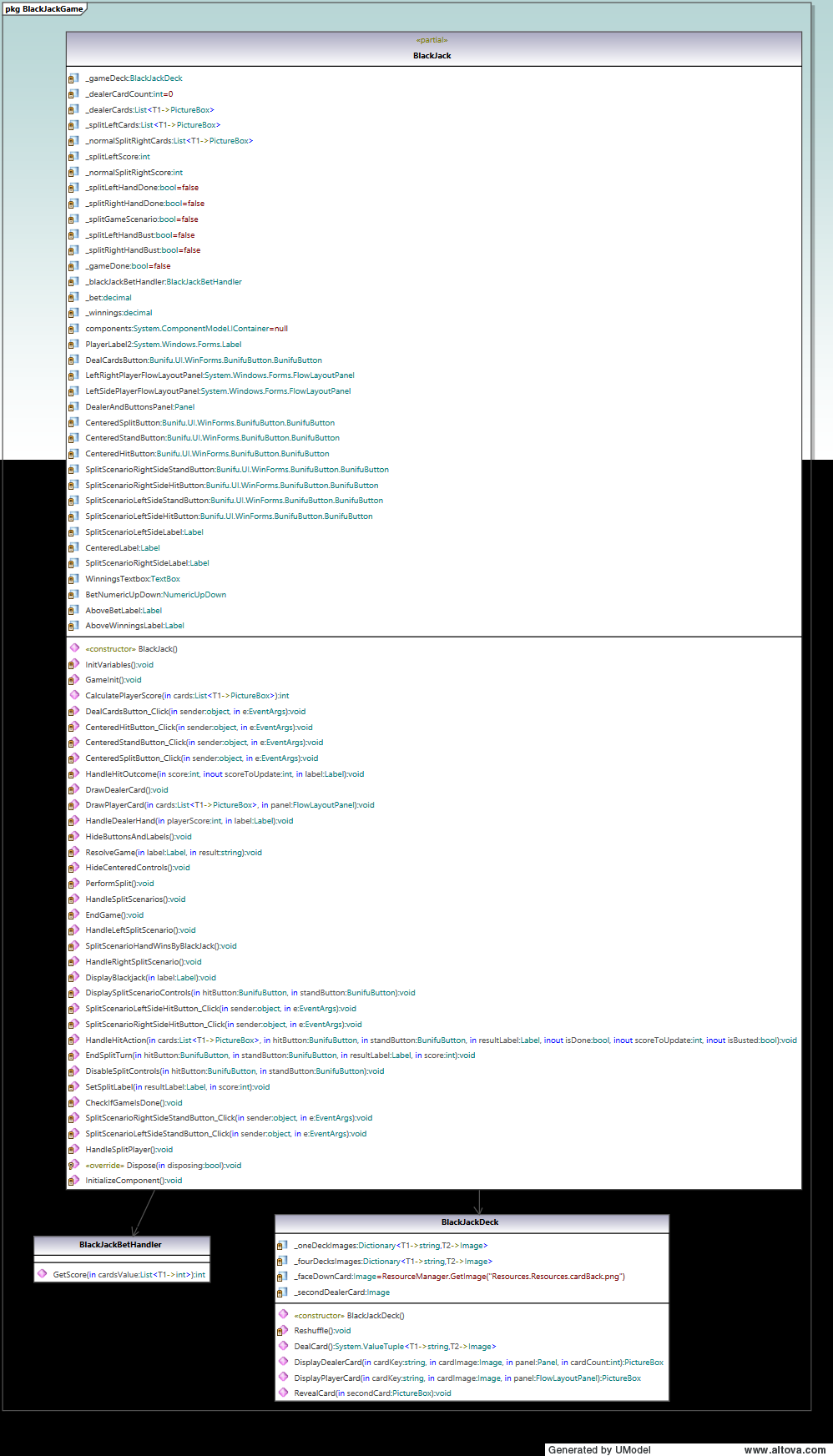


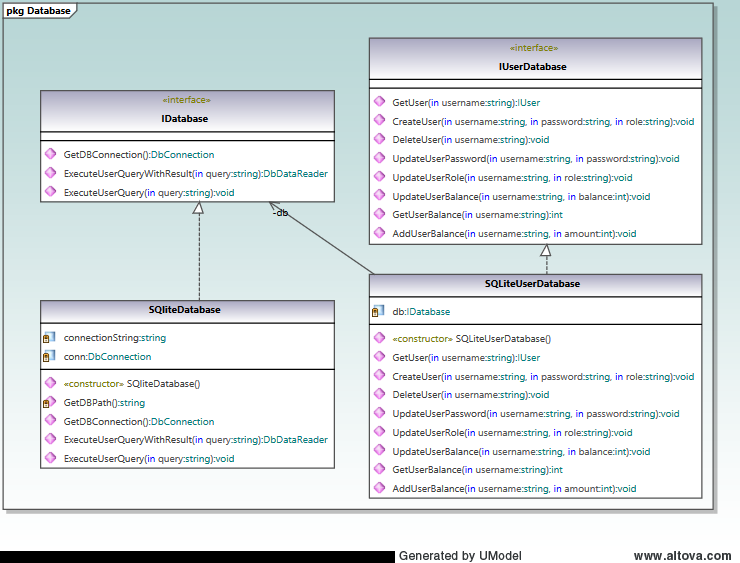
**Diagrama de clase**

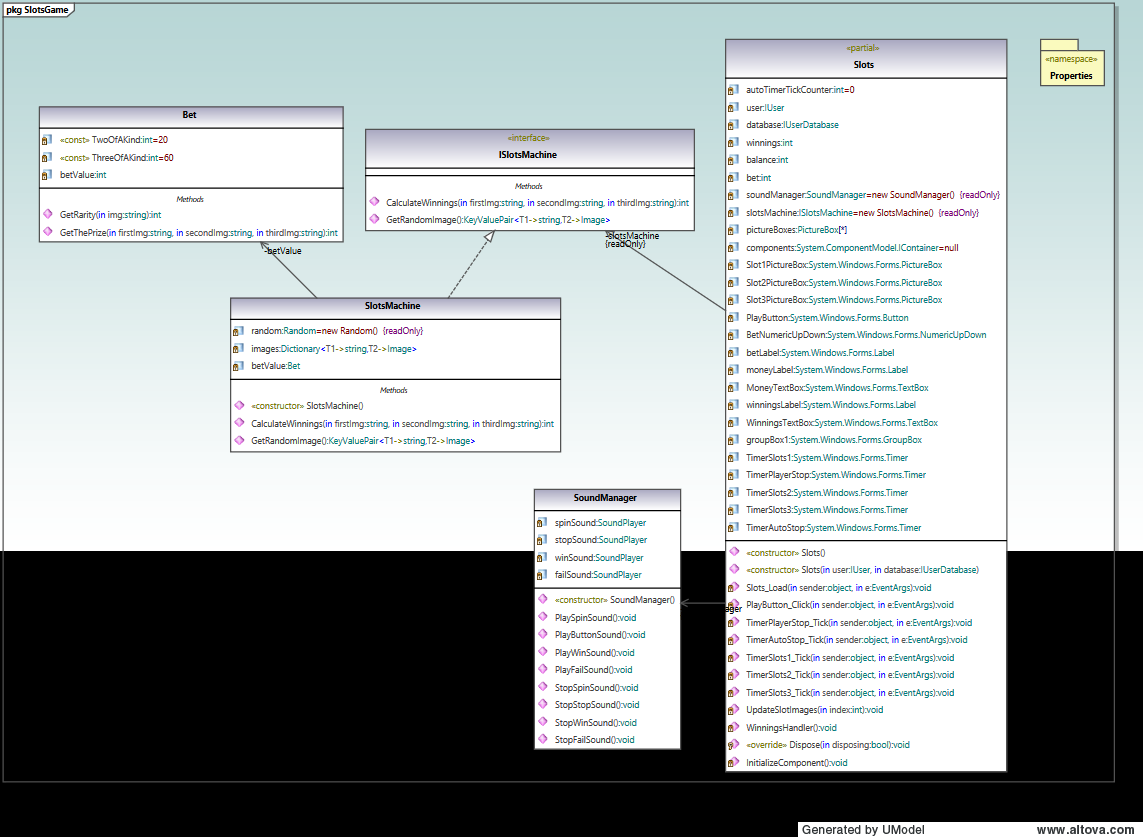




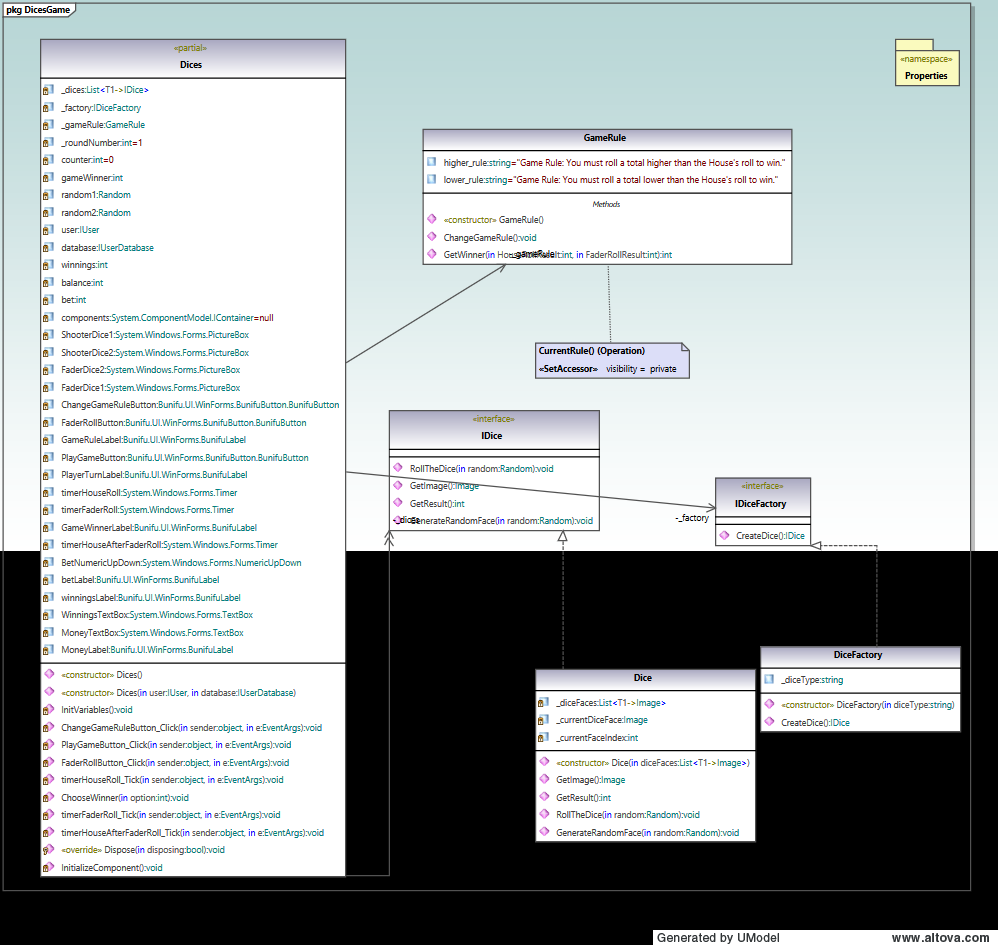


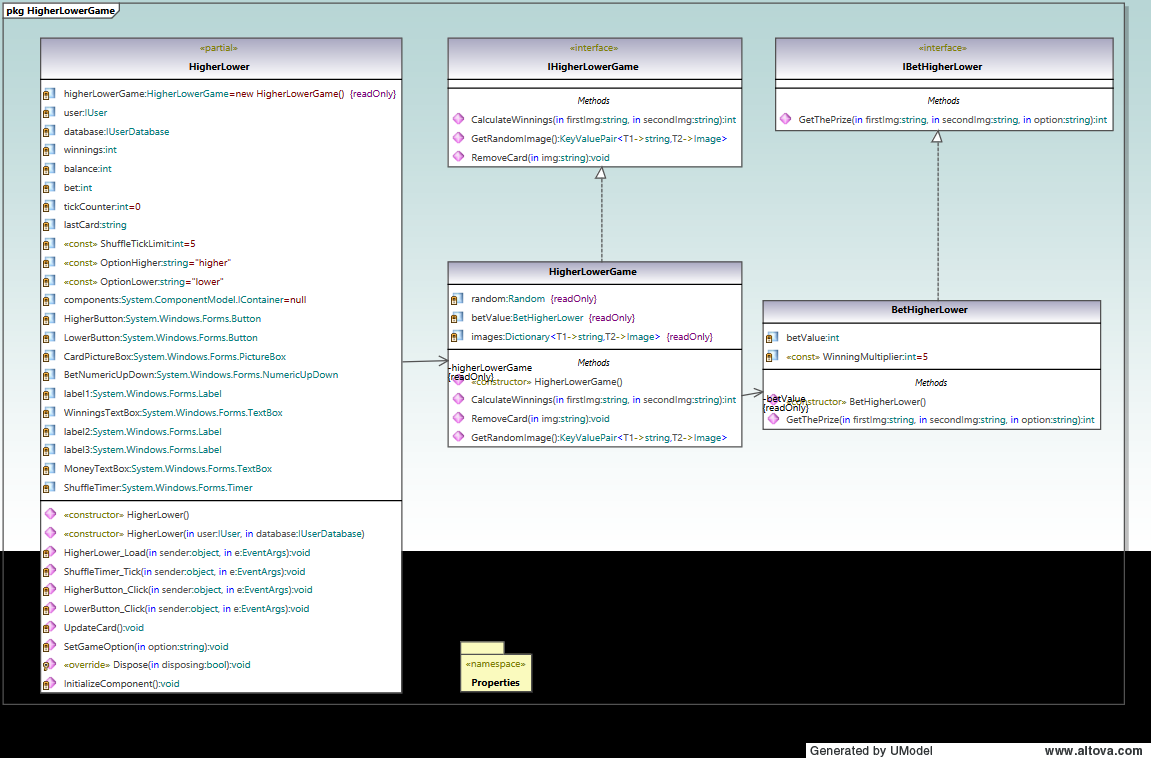






12





13

**Anexa 1: Exemple ale codului sursă asociat**

**A. Metode pentru realizarea conexiunii la baza de date**

string GetDBPath()

{

// Method to generate the connection string for SQLite database

string baseDirectory = AppDomain.CurrentDomain.BaseDirectory;

DirectoryInfo directoryInfo = new DirectoryInfo(baseDirectory);

directoryInfo = directoryInfo.Parent.Parent.Parent;

string databasePath = Path.Combine(directoryInfo.FullName, "Database", "Resources", "Users.db");

return $"Data Source={databasePath}";

}

public DbConnection GetDBConnection()

{

// Establish a connection to SQLite database

DbConnection conn = new SQLiteConnection(connectionString);

for (int i = 0; i < 3; i++)

{

try

{

conn.Open();

break;

}

catch (Exception exception)

{

throw new Exception($"The connection with the database could not be established\nException: {exception}");

}

}

return conn;

}

public IUser GetUser(string username)

{

DbDataReader reader = db.ExecuteUserQueryWithResult($"SELECT \* FROM Users WHERE Username = '{username}';");

// Check if the user exists in the database

if (reader.HasRows)

{

reader.Read();

// Create and return User object

return new User

(

reader.GetString(0), //username

reader.GetString(1), //password

reader.GetString(3), //role

reader.GetInt32(2) //balance

);

}

else

{

// Return null if user doesn't exist

return null;

}

}

public void CreateUser(string username, string password, string role)

{

db.ExecuteUserQuery($"INSERT INTO Users (Username, Password, Role) VALUES ('{username}', '{password}', '{role}');");

}

public void DeleteUser(string username)

{

db.ExecuteUserQuery($"DELETE FROM Users WHERE Username = '{username}';");

}

public void UpdateUserPassword(string username, string password)

{

db.ExecuteUserQuery($"UPDATE Users SET Password = '{password}' WHERE Username = '{username}';");

}

public void UpdateUserRole(string username, string role)

{

db.ExecuteUserQuery($"UPDATE Users SET Role = '{role}' WHERE Username = '{username}';");

}

public void UpdateUserBalance(string username, int balance)

{

db.ExecuteUserQuery($"UPDATE Users SET Balance = '{balance}' WHERE Username = '{username}';");

}

public int GetUserBalance(string username)

{

DbDataReader reader = db.ExecuteUserQueryWithResult($"SELECT Balance FROM Users WHERE Username = '{username}';");

reader.Read();

int balance = reader.GetInt32(0);

reader.Close();

return balance;

}

B. **Metode pentru realizarea de Game Logic**

1. Calcularea premiului la poker in functie de numere si culori

public int GetThePrize(string[] images)

{

// Arrays to store the parsed card data

int[] cards = new int[5];

string[] colors = new string[5];

// Parse the image strings into card and color arrays

ParseImages(images, cards, colors);

// Check for various winning hand types and calculate prize accordingly

// The ordering is important here, checking for the most valuable hands first

if (IsFlush(colors) && IsStraight(cards))

{

// Check for royal flush

return cards[4] == 15 ? BetValue \* RoyalFlushMultiplier : BetValue \* StraightFlushMultiplier;

}

else if (IsFlush(colors))

{

return BetValue \* FlushMultiplier;

}

else if (IsStraight(cards))

{

// Check for straight flush

return cards[4] == 15 ? BetValue \* StraightFlushMultiplier : BetValue \* StraightMultiplier;

}

else if (HasFourOfKind(cards))

{

return BetValue \* FourOfKindMultiplier;

}

else if (HasFullHouse(cards))

{

return BetValue \* FullHouseMultiplier;

}

else if (HasThreeOfKind(cards))

{

return BetValue \* ThreeOfKindMultiplier;

}

else if (HasTwoPairs(cards))

{

return BetValue \* TwoPairsMultiplier;

}

else if (HasPair(cards))

{

return BetValue \* PairMultiplier;

}

// No winning hand

return 0;

}

// Check if all cards are of the same color

private bool IsFlush(string[] colors) =>

Enumerable.Range(1, colors.Length - 1).All(i => colors[i] == colors[0]);

// Check if the cards form a continuous sequence

private bool IsStraight(int[] cards) =>

Enumerable.Range(1, cards.Length - 1).All(i => cards[i] == cards[i - 1] + 1);

// Check for four cards of the same value

private bool HasFourOfKind(int[] cards) =>

cards.GroupBy(c => c).Any(g => g.Count() == 4);

// Check for three cards of the same value and two cards of another value

private bool HasFullHouse(int[] cards) =>

cards.GroupBy(c => c).Count() == 2 && cards.GroupBy(c => c).Any(g => g.Count() == 3);

// Check for three cards of the same value

private bool HasThreeOfKind(int[] cards) =>

cards.GroupBy(c => c).Any(g => g.Count() == 3);

// Check for two pairs

private bool HasTwoPairs(int[] cards) =>

cards.GroupBy(c => c).Count(g => g.Count() == 2) == 2;

// Check for a pair

private bool HasPair(int[] cards) =>

cards.GroupBy(c => c).Any(g => g.Count() == 2);

1. Implementarea de state machine

public abstract class CoinState

{

// CoinFlip game instance

protected CoinFlip coinFlip;

// Constructor sets the CoinFlip game instance

public CoinState(CoinFlip coinFlip)

{

this.coinFlip = coinFlip;

}

// Abstract method to flip the coin - must be implemented by subclasses

public abstract void FlipCoin();

// Abstract method to perform a tick action - must be implemented by subclasses

public abstract void Tick();

}

public class FlippingState : CoinState

{

// Constructor: Initializes the FlippingState with a CoinFlip object

public FlippingState(CoinFlip coinFlip) : base(coinFlip) { }

// Ignored during the flipping state

public override void FlipCoin()

{

// No operation

}

// Handles animation updates and completion

public override void Tick()

{

if (coinFlip.currentFrame < coinFlip.coinFrames.Count)

{

// If the animation is not finished, update the coin's image

coinFlip.pictureBoxCoin.Image = coinFlip.coinFrames[coinFlip.currentFrame++];

}

else

{

// If the animation is finished, stop the timer, update the result, and reset variables

coinFlip.currentFrame = 0;

coinFlip.flippingFlag = false;

coinFlip.pictureBoxCoin.Image = coinFlip.coin.GetImage();

coinFlip.timerFlip.Stop();

coinFlip.UpdateResult();

}

}

}

public class NotFlippingState : CoinState

{

public NotFlippingState(CoinFlip coinFlip) : base(coinFlip) { }

public override void FlipCoin()

{

// Check if the user has selected a coin face

string selectedCoinFace = coinFlip.comboBoxCoinFace.SelectedItem?.ToString();

if (string.IsNullOrEmpty(selectedCoinFace))

{

coinFlip.bunifuLabelResult.Text = "You have to choose the coin face, Heads or Tails";

return;

}

// Flip the coin

coinFlip.coinFrames = coinFlip.coin.Flip();

coinFlip.flippingFlag = true;

coinFlip.bunifuLabelResult.Text = "";

coinFlip.timerFlip.Start(); // Start the Timer to handle the animation

coinFlip.state = new FlippingState(coinFlip); // State changed to FlippingState when coin starts flipping

}

public override void Tick()

{

// No operation

}

}

**ANEXA 2: Etapa de testare**

**Unit Tests pentru jocul de Poker**

[TestMethod]

public void TestAddCardToDeck()

{

var cardToAdd = "\_02trefla";

var cardImage = poker.Images[cardToAdd];

poker.RemoveCard(cardToAdd);

var initialCount = poker.Images.Count;

poker.AddCard(cardToAdd, cardImage);

var finalCount = poker.Images.Count;

Assert.AreEqual(initialCount + 1, finalCount);

}

[TestMethod]

public void TestGetRandomImageReturnsImage()

{

var image = poker.GetRandomImage();

Assert.IsNotNull(image.Value);

Assert.IsTrue(poker.Images.ContainsKey(image.Key));

}

[TestMethod]

public void TestCalculateWinningsForOnePair()

{

string[] highCardImages = { "\_02trefla", "\_02romb", "\_06frunza", "\_08inima", "\_10trefla" };

int expectedWinnings = 2;

poker.BetValue = 1;

int actualWinnings = poker.CalculateWinnings(highCardImages);

Assert.AreEqual(expectedWinnings, actualWinnings);

}

**Unit Tests pentru jocul de BlackJack**

[TestInitialize]

public void Initializa()

{

panel = new FlowLayoutPanel();

form = new BlackJack();

test = ResourceManager.GetImage("Resources.Resources.cardBack.png");

deck = new BlackJackDeck();

pictureBoxes = new List<PictureBox>{

deck.DisplayPlayerCard("1#trefla\_02",test ,panel),

deck.DisplayPlayerCard("3#trefla\_11",test ,panel),

deck.DisplayPlayerCard("4#frunza\_11",test ,panel),

deck.DisplayPlayerCard("2#inima\_14",test ,panel),

};

bet = new BlackJackBetHandler();

}

//testing if the picture box list i update whenever i display a new card gives me the wanted value

[TestMethod]

public void TestCalculatePlayerScore()

{

Assert.AreEqual(form.CalculatePlayerScore(pictureBoxes),14);

}

[TestMethod]

public void TestGetScore()

{

Assert.AreEqual(bet.GetScore(new List<int> { 11, 3, 4 }), 18);

Assert.AreEqual(bet.GetScore(new List<int> { 11, 3, 4 }), 18);

Assert.AreEqual(bet.GetScore(new List<int> { 11, 3, 4 }), 18);

Assert.AreEqual(bet.GetScore(new List<int> { 11, 11, 14 }), 12);

Assert.AreEqual(bet.GetScore(new List<int> { 11, 11, 11, 10, 11 }), 14);

Assert.AreEqual(bet.GetScore(new List<int> { 11, 13, 14 }), 21);

Assert.AreEqual(bet.GetScore(new List<int> { 11, 11, 10, 14, 13 }), 32);

}

**Unit Tests pentru baza de date**

[TestClass]

public class DatabaseTests

{

SQliteDatabase sqliteDatabase;

SQLiteUserDatabase userDatabase;

[TestInitialize]

public void Initialize()

{

sqliteDatabase = new SQliteDatabase();

userDatabase = new SQLiteUserDatabase(sqliteDatabase);

}

[TestMethod]

public void TestDBConnection()

{

// Expecting the connection to be open

Assert.AreEqual(System.Data.ConnectionState.Open, sqliteDatabase.GetDBConnection().State);

}

[TestMethod]

public void TestExecuteQueryWithResult()

{

// This query is expected to return at least one user

var reader = sqliteDatabase.ExecuteUserQueryWithResult("SELECT \* FROM Users");

// Expecting to have at least one row

Assert.IsTrue(reader.HasRows);

}

[TestMethod]

public void TestExecuteQuery()

{

// This query is expected not to throw any exceptions

sqliteDatabase.ExecuteUserQuery("SELECT \* FROM Users");

// If the test reaches this point, then the query execution is successful, and we pass the test

Assert.IsTrue(true);

}

[TestMethod]

public void TestCreateExtractAndDeleteUser()

{

userDatabase.CreateUser("TestUser", "TestPassword", "Admin");

IUser user = userDatabase.GetUser("TestUser");

Assert.IsNotNull(user);

Assert.AreEqual("TestUser", user.Username);

Assert.AreEqual("TestPassword", user.Password);

Assert.AreEqual("Admin", user.Role);

Assert.AreEqual(0, user.Balance);

userDatabase.DeleteUser("TestUser");

user = userDatabase.GetUser("TestUser");

Assert.IsNull(user);

}

[TestMethod]

public void TestAddMoney()

{

userDatabase.CreateUser("TestUser", "TestPassword", "Admin");

Assert.AreEqual(0, userDatabase.GetUserBalance("TestUser"));

userDatabase.AddUserBalance("TestUser", 100);

Assert.AreEqual(100, userDatabase.GetUserBalance("TestUser"));

userDatabase.AddUserBalance("TestUser", 200);

Assert.AreEqual(300, userDatabase.GetUserBalance("TestUser"));

userDatabase.AddUserBalance("TestUser", -100);

Assert.AreEqual(200, userDatabase.GetUserBalance("TestUser"));

userDatabase.DeleteUser("TestUser");

Assert.IsNull(userDatabase.GetUser("TestUser"));

}

}