using System;

using System.Collections;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using DataAccessLayer;

using DataLayer;

namespace Lucas\_Said\_6.\_1A\_HomeAssignment

{

public class Presentation

{

static Players players = new Players();

static Games game = new Games();

static Data data = new Data();

static DataAccess dal = new DataAccess();

static Games games = new Games();

//Boolean Variables

static bool userQuit = false;

static bool login\_done = false;

public static string gameTitle;

static string usernameOne;

static string usernameTwo;

// Create an instance of GameScreen

static GameScreen gameScreen = new GameScreen(GameScreen.LoadCells());

static void Main(string[] args)

{

do

{

if (login\_done == false)

{

Console.WriteLine("-----------------------");

Console.WriteLine("1. Add Player Details");

Console.WriteLine("2. Quit");

Console.WriteLine("-----------------------");

}

else

{

Console.WriteLine("-----------------------");

Console.WriteLine("1. Configure Ships");

Console.WriteLine("2. Launch Attack");

Console.WriteLine("3. Quit");

Console.WriteLine("-----------------------");

}

Console.WriteLine("Choose your option");

var choice = Console.ReadLine();

if (int.TryParse(choice, out int userChoice))

{

if (login\_done == false)

{

switch (userChoice)

{

case 1:

userDetails();

Console.WriteLine("");

break;

case 2:

Console.WriteLine("");

userQuit = true;

break;

default:

Console.Clear();

Console.WriteLine("Input a choice from the menu");

break;

}

}

else if(userQuit==false)

{

switch (userChoice)

{

case 1:

configureShips();

Console.WriteLine("");

break;

case 2:

attack();

Console.WriteLine("");

break;

case 3:

Console.WriteLine("");

userQuit = true;

break;

default:

Console.Clear();

Console.WriteLine("Input a choice from the menu");

break;

}

}

}

else

{

Console.Clear();

Console.WriteLine("Input a choice from the menu using numbers");

}

} while (!userQuit);

}

private static void userDetails()

{

// Connection to Database

SqlConnection myconnection = dal.openDB();

for (int i = 1; i < 3; i++)

{

string playerUsername;

string newpassword;

do

{

Console.WriteLine("Enter Username (Player " + i + ")");

playerUsername = Console.ReadLine();

if (string.IsNullOrWhiteSpace(playerUsername))

{

Console.WriteLine("Username cannot be empty. Please try again.");

}

} while (string.IsNullOrWhiteSpace(playerUsername));

do

{

Console.WriteLine("Create your Password");

newpassword = MaskUserInput();

if (string.IsNullOrWhiteSpace(newpassword))

{

Console.WriteLine("Password cannot be empty. Please try again.");

}

} while (string.IsNullOrWhiteSpace(newpassword));

players.Username = playerUsername;

players.Password = newpassword;

data.addPlayer(players, myconnection);

if (i == 1)

{

usernameOne = playerUsername;

players.PlayerOne = playerUsername;

}

else

{

usernameTwo = playerUsername;

players.PlayerTwo = playerUsername;

}

}

string gameTitle;

do

{

Console.WriteLine("Enter a Title for your Game");

gameTitle = Console.ReadLine();

if (string.IsNullOrWhiteSpace(gameTitle))

{

Console.WriteLine("Game title cannot be empty. Please try again.");

}

} while (string.IsNullOrWhiteSpace(gameTitle));

int user1\_id = data.getPlayerId(usernameOne, myconnection);

int user2\_id = data.getPlayerId(usernameTwo, myconnection);

games.Title = gameTitle;

games.CreatorFK = user1\_id.ToString();

games.OpponentFK = user2\_id.ToString();

data.addGame(games, myconnection);

login\_done = true;

}

private static void configureShips()

{

DataAccess dal = new DataAccess();

Data data = new Data();

SqlConnection myconnection = dal.openDB();

GameShipConfiguration shipConfig = new GameShipConfiguration();

List<string> validCoordinates = new List<string>{"A1", "A2", "A3", "A4", "A5","B1", "B2", "B3", "B4", "B5","C1", "C2", "C3", "C4", "C5","D1", "D2", "D3", "D4", "D5","E1", "E2", "E3", "E4", "E5"};

for (int i = 1; i <= 2; i++)

{

Console.WriteLine("Player" + i);

for (int s = 1; s <= 5; s++)

{

string coordinate;

do

{

Console.Write("Enter Ship Coordinate: ");

coordinate = Console.ReadLine().ToUpper();

if (!validCoordinates.Contains(coordinate))

{

Console.WriteLine("Invalid coordinate. Please enter a valid coordinate.");

}

} while (!validCoordinates.Contains(coordinate));

if (i == 1)

{

shipConfig.PlayerFK = players.PlayerOne;

}

else

{

shipConfig.PlayerFK = players.PlayerTwo;

}

shipConfig.GameFK = data.getGameId(gameTitle, myconnection);

shipConfig.Coordinate = coordinate;

try

{

bool overlap = data.shipOverlap(shipConfig, myconnection);

while (overlap)

{

Console.WriteLine("Ship Overlapping. Please enter a different coordinate.");

coordinate = Console.ReadLine().ToUpper();

if (i == 1)

{

shipConfig.PlayerFK = players.PlayerOne;

}

else

{

shipConfig.PlayerFK = players.PlayerTwo;

}

shipConfig.GameFK = data.getGameId(gameTitle, myconnection);

shipConfig.Coordinate = coordinate;

overlap = data.shipOverlap(shipConfig, myconnection);

}

data.addShipConfiguration(shipConfig, myconnection);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.WriteLine("Wrong coordinate");

// Reset position if overlapping

s--;

}

}

Console.Clear();

}

dal.closeDB();

}

private static void attack()

{

gameScreen.LoadGrid(players);

do

{

Console.WriteLine("Player 1");

Console.WriteLine("Enter attack coordinates: ");

string attackCoordinatesPlayerOne = Console.ReadLine().ToUpper();

gameScreen.AttackMode(players,players.PlayerOne,attackCoordinatesPlayerOne);

gameScreen.PrintGrid(players, players.PlayerOne, attackCoordinatesPlayerOne);

if (game.Complete)

{

break;

}

Console.WriteLine("Player 2");

Console.WriteLine("Enter attack coordinates: ");

string attackCoordinatesPlayerTwo = Console.ReadLine().ToUpper();

gameScreen.AttackMode(players, players.PlayerTwo, attackCoordinatesPlayerTwo);

gameScreen.PrintGrid(players, players.PlayerTwo, attackCoordinatesPlayerTwo);

} while (!game.Complete);

}

static string MaskUserInput()

{

string userInput = "";

ConsoleKeyInfo key;

do

{

//This prevents the user input from getting displayed

key = Console.ReadKey(intercept: true);

//If statement to check what character was inputted

if (char.IsLetterOrDigit(key.KeyChar) || char.IsPunctuation(key.KeyChar))

{

//Add the character to the user input

userInput += key.KeyChar;

//Output an asterisk in the console

Console.Write('\*');

}

//Check if the user pressed backspace and move the cursor one back

else if (key.Key == ConsoleKey.Backspace && userInput.Length > 0)

{

Console.SetCursorPosition(Console.CursorLeft - 1, Console.CursorTop);

Console.Write(' ');

Console.SetCursorPosition(Console.CursorLeft - 1, Console.CursorTop);

userInput = userInput.Substring(0, userInput.Length - 1);

}

//Loop until the user pressed enter

} while (key.Key != ConsoleKey.Enter);

Console.WriteLine();

return userInput;

}

}

}

using DataAccessLayer;

using DataLayer;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lucas\_Said\_6.\_1A\_HomeAssignment

{

public class GameScreen

{

private List<Cell> cells;

List<string> shipCoordinatesPlayerOne = new List<string>();

List<string> shipCoordinatesPlayerTwo = new List<string>();

//Instatiate objects

Data data = new Data();

static DataAccess dal = new DataAccess();

SqlConnection myconnection = dal.openDB();

int game\_id = 0;

public GameScreen(List<Cell> cells)

{

this.cells = cells;

}

public void LoadGrid(Players playerDetail)

{

game\_id = data.getGameId(Presentation.gameTitle, myconnection);

shipCoordinatesPlayerOne = data.getShipCoordinate(game\_id, playerDetail.PlayerOne, myconnection);

shipCoordinatesPlayerTwo = data.getShipCoordinate(game\_id, playerDetail.PlayerTwo, myconnection);

}

public void PrintGrid(Players playerDetail, string player, string attacks)

{

StringBuilder grid = new StringBuilder();

int rowSize = 5;

grid.Append(" 1 2 3 4 5\n");

grid.Append(" +---+---+---+---+---+\n");

for (int y = 0; y < rowSize; y++)

{

char note = (char)('A' + y);

grid.Append(note + "| ");

for (int x = 1; x <= rowSize; x++)

{

string cellCoordinate = note.ToString() + x;

// Check if the current player is Player 1

if (player == playerDetail.PlayerOne)

{

// For Player 1, check if the current cell coordinate is in the list of ship coordinates for Player 1

if (shipCoordinatesPlayerOne.Contains(cellCoordinate))

{

// Cell has a ship

grid.Append("S ");

}

else

{

// Cell does not have a ship

// Check if the current cell coordinate is in the list of attack coordinates for Player 1

bool isAttackCoordinate = shipCoordinatesPlayerTwo.Contains(cellCoordinate);

// Iterate through the cells list to find and print the appropriate cell type

foreach (var cell in cells)

{

if (cell.Coordinate == cellCoordinate)

{

// If it's an AttackCell, use the specific PrintCell method

if (cell is AttackCell)

{

((AttackCell)cell).PrintCell(isAttackCoordinate);

}

else

{

cell.PrintCell();

}

break;

}

}

}

}

else

{

// For Player 2, check if the current cell coordinate is in the list of ship coordinates for Player 2

if (shipCoordinatesPlayerTwo.Contains(cellCoordinate))

{

// Cell has a ship

grid.Append("S ");

}

else

{

// Cell does not have a ship

// Check if the current cell coordinate is in the list of attack coordinates for Player 2

bool isAttackCoordinate = shipCoordinatesPlayerOne.Contains(cellCoordinate);

// Iterate through the cells list to find and print the appropriate cell type

foreach (var cell in cells)

{

if (cell.Coordinate == cellCoordinate)

{

// If it's an AttackCell, use the specific PrintCell method

if (cell is AttackCell)

{

((AttackCell)cell).PrintCell(isAttackCoordinate);

}

else

{

cell.PrintCell();

}

break;

}

}

}

}

if (x != rowSize)

{

grid.Append("| ");

}

}

grid.AppendLine("\n +---+---+---+---+---+");

}

Console.WriteLine(grid.ToString());

}

public void AttackMode(Players playerDetail, string player, string attacks)

{

if (player == playerDetail.PlayerOne)

{

if (shipCoordinatesPlayerTwo.Count == 0)

{

Console.WriteLine("Player 1 Wins!");

data.gameOver(game\_id, myconnection);

Environment.Exit(0);

}

if (shipCoordinatesPlayerTwo.Contains(attacks))

{

Console.WriteLine("Hit");

shipCoordinatesPlayerTwo.Remove(attacks);

}

else

{

Console.WriteLine("You missed");

}

}

else

{

if (shipCoordinatesPlayerOne.Count == 0)

{

Console.WriteLine("Player 2 Wins!");

data.gameOver(game\_id, myconnection);

Environment.Exit(0);

}

if (shipCoordinatesPlayerOne.Contains(attacks))

{

Console.WriteLine("Hit");

shipCoordinatesPlayerOne.Remove(attacks);

}

else

{

Console.WriteLine("You missed");

}

}

Console.WriteLine("Player 1 ships remaining: "+shipCoordinatesPlayerOne.Count());

Console.WriteLine("Player 2 ships remaining:"+shipCoordinatesPlayerTwo.Count());

}

public static List<Cell> LoadCells()

{

List<Cell> cells = new List<Cell>();

// Example: Add ShipCells to the list

for (int i = 0; i < 8; i++)

{

cells.Add(new ShipCell());

}

// Example: Add AttackCells to the list

for (int i = 0; i < 8; i++)

{

cells.Add(new AttackCell("X"));

}

return cells;

}

}

public abstract class Cell

{

// Common properties for both a cell showing a ship or a successful/missed attack

// This has to be implemented differently in the next two classes shown here

// for a cell representing a ship or a cell representing a Ship or a Cell representing a successful attack or a missed one

// If it's a ship, print S

// If it's missed, print M

// If it's successful, print X

public abstract void PrintCell();

// Add a property to store the coordinate of the cell

public string Coordinate { get; set; }

}

public class ShipCell : Cell

{

public override void PrintCell()

{

Console.Write("S");

}

}

public class AttackCell : Cell

{

private string status;

public AttackCell(string status)

{

this.status = status;

}

public override void PrintCell()

{

Console.Write(status + " ");

}

public void PrintCell(bool isAttackCoordinate)

{

if (isAttackCoordinate)

{

Console.Write("X ");

}

else

{

Console.Write("- ");

}

}

}

}

using DataAccessLayer;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Security.Cryptography;

namespace DataLayer

{

public class Data

{

public static void Main()

{

}

DataAccess dal = new DataAccess();

public Exception addPlayer(Players players, SqlConnection sqlConnection)

{

return dal.addPlayer(players, sqlConnection);

}

public int selectPlayer(Players players, SqlConnection sqlConnection)

{

return dal.selectPlayer(players, sqlConnection);

}

public Exception addGame(Games games, SqlConnection sqlConnection)

{

return dal.addGame(games, sqlConnection);

}

public Exception addShipConfiguration(GameShipConfiguration shipConfig, SqlConnection sqlConnection)

{

return dal.addShipConfiguration(shipConfig, sqlConnection);

}

public bool shipOverlap(GameShipConfiguration shipConfig, SqlConnection sqlConnection)

{

return dal.shipOverlap(shipConfig, sqlConnection);

}

public int getPlayerId(string username, SqlConnection sqlConnection)

{

return dal.getPlayerId(username, sqlConnection);

}

public int getGameId(string title, SqlConnection sqlConnection)

{

return dal.getGameId(title, sqlConnection);

}

public List<string> getShipCoordinate(int gameId,string username, SqlConnection sqlConnection)

{

return dal.getShipCoordinate(gameId,username, sqlConnection);

}

public Exception gameOver(int gameId, SqlConnection sqlConnection)

{

return dal.gameOver(gameId, sqlConnection);

}

}

}

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Diagnostics;

using System.Linq;

using System.Net.Configuration;

using System.Text;

using System.Threading.Tasks;

using ConnectionLayer;

using System.Runtime.Remoting.Contexts;

namespace DataAccessLayer

{

public class DataAccess

{

public static void Main()

{

}

// Creating an instance of the Connection class

Connection connection = new Connection();

public SqlConnection openDB()

{

// Calling ConnectionToDB() method

return connection.ConnectionToDB();

}

public void closeDB()

{

// Calling CloseDB() method

connection.CloseDB();

}

public Exception addPlayer(Players players, SqlConnection sqlConnection)

{

try

{

string addPlayerQuery = "INSERT INTO [dbo].[Players] ([Username], [Password]) " +

"VALUES ('"+ players.Username + "', '"+ players.Password + "')";

SqlCommand addPlayersCMD = new SqlCommand(addPlayerQuery, sqlConnection);

// Execute the command

addPlayersCMD.ExecuteNonQuery();

//To see if the players is added

Console.WriteLine("Player Added");

return null;

}

catch (Exception ex)

{

Console.WriteLine("Error adding Player");

return ex;

}

}

public int selectPlayer(Players players, SqlConnection sqlConnection)

{

try

{

string checkPlayerQuery = "SELECT COUNT(\*) FROM [dbo].[Players] WHERE Username LIKE '%" + players.Username + "%'";

SqlCommand checkPlayerCmd = new SqlCommand(checkPlayerQuery, sqlConnection);

int playerCount = (int)checkPlayerCmd.ExecuteScalar();

return playerCount;

}

catch (Exception ex)

{

Console.WriteLine("Error checking for Players");

return 0;

}

}

public Exception addGame(Games games, SqlConnection sqlConnection)

{

try

{

//SQL string that represnts adding the Player

string addPlayerQuery = "INSERT INTO [dbo].[Game] ([Title], [CreatorFK], [OpponentFK], [Complete]) " +

"VALUES ('" + games.Title + "', " + games.CreatorFK + ", " + games.OpponentFK + ", 0)";

//Adding the player object into the database

SqlCommand addPlayersCMD = new SqlCommand(addPlayerQuery, sqlConnection);

// Execute the commands

addPlayersCMD.ExecuteNonQuery();

return null;

}

catch (Exception ex)

{

Console.WriteLine("Error adding Game");

return ex;

}

}

public Exception addShipConfiguration(GameShipConfiguration shipConfig, SqlConnection sqlConnection)

{

try

{

string addShipConfigQuery = "INSERT INTO [dbo].[GameShipConfiguration] ([PlayerFK], [GameFK], [Coordinate]) " +

"VALUES ('"+shipConfig.PlayerFK+"', "+shipConfig.GameFK+", '"+shipConfig.Coordinate+"')";

SqlCommand command = new SqlCommand(addShipConfigQuery, sqlConnection);

command.ExecuteNonQuery();

Console.WriteLine("Ship Configuration Added");

return null;

}

catch (Exception ex)

{

Console.WriteLine("Error adding Configuration");

return ex;

}

}

public bool shipOverlap(GameShipConfiguration shipConfig, SqlConnection sqlConnection)

{

try

{

string checkOverlapQuery = "SELECT COUNT(\*) FROM [dbo].[GameShipConfiguration] " +

"WHERE [Coordinate] = '"+shipConfig.Coordinate+"' AND [PlayerFK] = '"+shipConfig.PlayerFK+"'";

SqlCommand command = new SqlCommand(checkOverlapQuery, sqlConnection);

int overlapCount = (int)command.ExecuteScalar();

if (overlapCount>0)

{

return true;

}else

{

return false;

}

}

catch (Exception ex)

{

Console.WriteLine("Error checking for overlap");

return false;

}

}

public int getPlayerId(string username, SqlConnection sqlConnection)

{

try

{

string playerId\_query = "SELECT ID FROM [dbo].[Players] WHERE Username = '" + username + "'";

SqlCommand idGetter = new SqlCommand(playerId\_query, sqlConnection);

int playerId = (int)idGetter.ExecuteScalar();

return playerId;

}

catch (Exception ex)

{

Console.WriteLine("Error getting Player Id");

return 0;

}

}

public int getGameId(string title, SqlConnection sqlConnection)

{

try

{

string gameIdQuery = "SELECT ID FROM [dbo].[Game] WHERE Title = '" + title + "'";

SqlCommand gameIDgetter = new SqlCommand(gameIdQuery, sqlConnection);

int gameId = (int)gameIDgetter.ExecuteScalar();

return gameId;

}

catch (Exception ex)

{

Console.WriteLine("Error getting Game Id");

return 0;

}

}

public List<string> getShipCoordinate(int gameId,string username, SqlConnection sqlConnection)

{

try

{

List<string> coordinates = new List<string>();

string query = "SELECT Coordinate FROM [dbo].[GameShipConfiguration] WHERE GameFK = " + gameId+ " AND PlayerFK = '"+username+"'";

SqlCommand command = new SqlCommand(query, sqlConnection);

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

string coordinate = reader["Coordinate"].ToString();

coordinates.Add(coordinate);

}

reader.Close();

return coordinates;

}

catch (Exception ex)

{

Console.WriteLine("Error with the Ship Coordinate");

return null;

}

}

public Exception gameOver(int gameId, SqlConnection sqlConnection)

{

try

{

string gameIdQuery = "UPDATE [dbo].[Game] SET Complete WHERE ID ="+gameId;

SqlCommand updateCmd = new SqlCommand(gameIdQuery, sqlConnection);

updateCmd.ExecuteNonQuery();

return null;

}

catch (Exception ex)

{

Console.WriteLine("Error with the game");

return ex;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.SqlClient;

namespace ConnectionLayer

{

public class Connection

{

public static void Main()

{

}

private SqlConnection conn;

public Connection()

{

//Connection string to connect to the DB

string connectionString = new SqlConnectionStringBuilder()

{

DataSource = ".\\SQLEXPRESS",

InitialCatalog = "oopHome",

IntegratedSecurity = true,

UserID = "LAPTOP-LUCAS\\lucas",

}.ConnectionString;

//This is to populate the Sqlconnection

conn = new SqlConnection(connectionString);

}

public SqlConnection ConnectionToDB()

{

try

{

// Openening the connection

conn.Open();

// Checking if the program can connect

if (conn.State == System.Data.ConnectionState.Open)

{

Console.WriteLine(" ");

} else

{

Console.WriteLine("Error Connecting to DB");

}

return conn;

}

catch (Exception ex)

{

Console.WriteLine($"Connection unsuccessful. Error: {ex.Message}");

return null;

}

}

public void CloseDB()

{

// Closing the connection

if (conn.State == System.Data.ConnectionState.Open)

{

conn.Close();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer

{

internal class Ships

{

public int ID { get; set; }

public string Title { get; set; }

public int Size { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer

{

public class GameShipConfiguration

{

public int ID { get; set; }

public string PlayerFK { get; set; }

public int GameFK { get; set; }

public string Coordinate { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer

{

public class Players

{

public int ID { get; set; }

public string Username { get; set; }

public string PlayerOne { get; set; }

public string PlayerTwo { get; set; }

public string Password { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer

{

public class Games

{

public int ID { get; set; }

public string Title { get; set; }

public string CreatorFK { get; set; }

public string OpponentFK { get; set; }

public bool Complete { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DataAccessLayer

{

public class Attacks

{

public int ID { get; set; }

public string Coordinate { get; set; }

public bool Hit { get; set; }

public int GameFK { get; set; }

}

}