# DELL

# Session 2023 - 2027

**Submitted by:**

Abdul Rehman 2023-CS-73

### Supervised by:

Muhammad Laeeq uz Zaman Khan Niazi

### Course:

CSC-103L Object Oriented Programming B

Department of Computer Science

**University of Engineering and Technology Lahore Pakistan**

**Table of content**

* [Introduction](#_bookmark0)
* Users of Application
* [Functionalities](#_bookmark1)
* [Wire Frames](#_bookmark2)
* [Uniqueness……………………………………………………….](#_TOC_250002).........................
* [C](#_TOC_250001)ode………..…………………………………………………………………….
* [CRC Diagram](#_TOC_250000)

# Introduction:

This semester, we're creating a system to help companies manage their employees and products better. Our aim is to make it easy for organizations to keep track of who's doing what and what products they have. We're focusing on making it simple to use and giving it useful features. With our system, users can easily see what tasks their employees are working on and keep an eye on how products are moving through the system. Our goal is to help companies work more efficiently and smoothly, making everyone's job easier.

I use 3 tire model i.e. (BL, DL, UI) in my business application. Using database for storage of data.

* I have done Windows Form Application with Database Also, On Entity that is User is done with both file handling and Database.
* On the other hand, I have done my Console project of one entity that is User with both File handing and Database.

### Users of Application:

* Admin

Admin: He can manage Products employees and customers.

* Sales Person

Sales Person: He can place orders for customers.

* Technician

Technician: He can Update the product.

* Customer

Customer: He can buy the product.

### Functionalities :

O Following are the functionalities of user i.e, What user can do?

***User ADMIN Functions So that I can Story ID***

|  |  |  |
| --- | --- | --- |
| Admin | view all data | See all the data that are added. |
| Manage sales person | Manage sale person. |
| Manage technician. | Manage Technician. |
| Manage Customer. | Manage customer. |
| View orders | View orders |
| Manage deactivate user | View users who have logged out |
| Manage products | View all products |

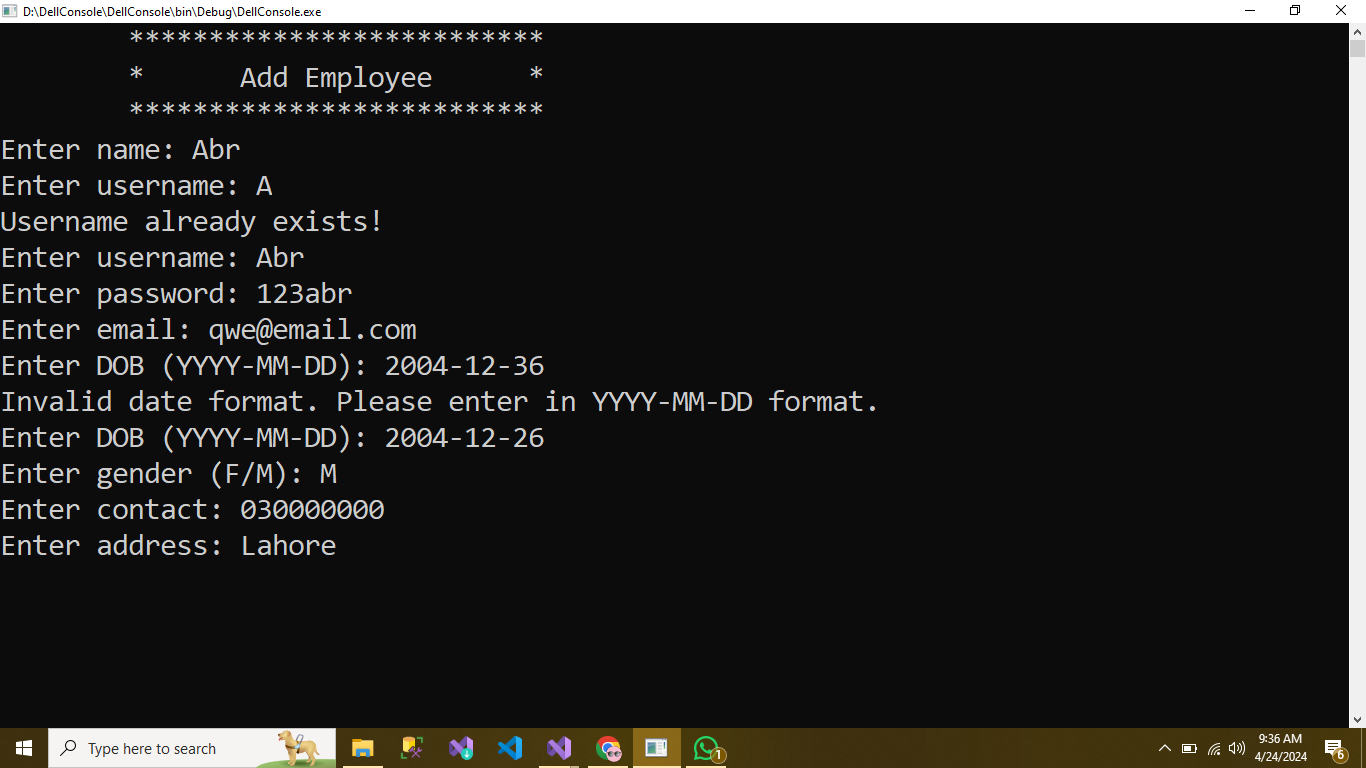
***User Customer Functions So that I can Story ID***

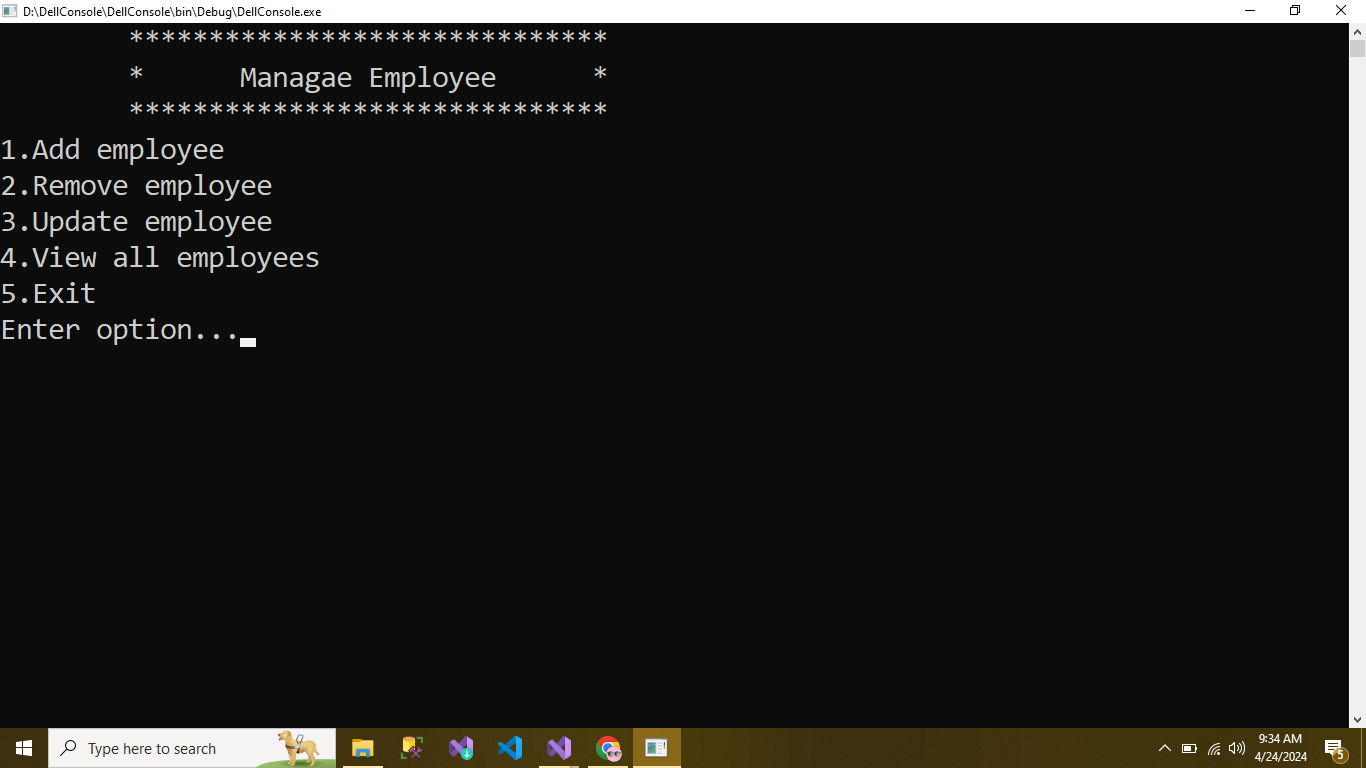
|  |  |  |
| --- | --- | --- |
| Customer | Buy | See all the data that are added. |
| View cart | Manage sale person. |
| Place order | Manage Technician. |
| Confirm order | Manage customer. |
| Update account | View orders |

### Wire frames:

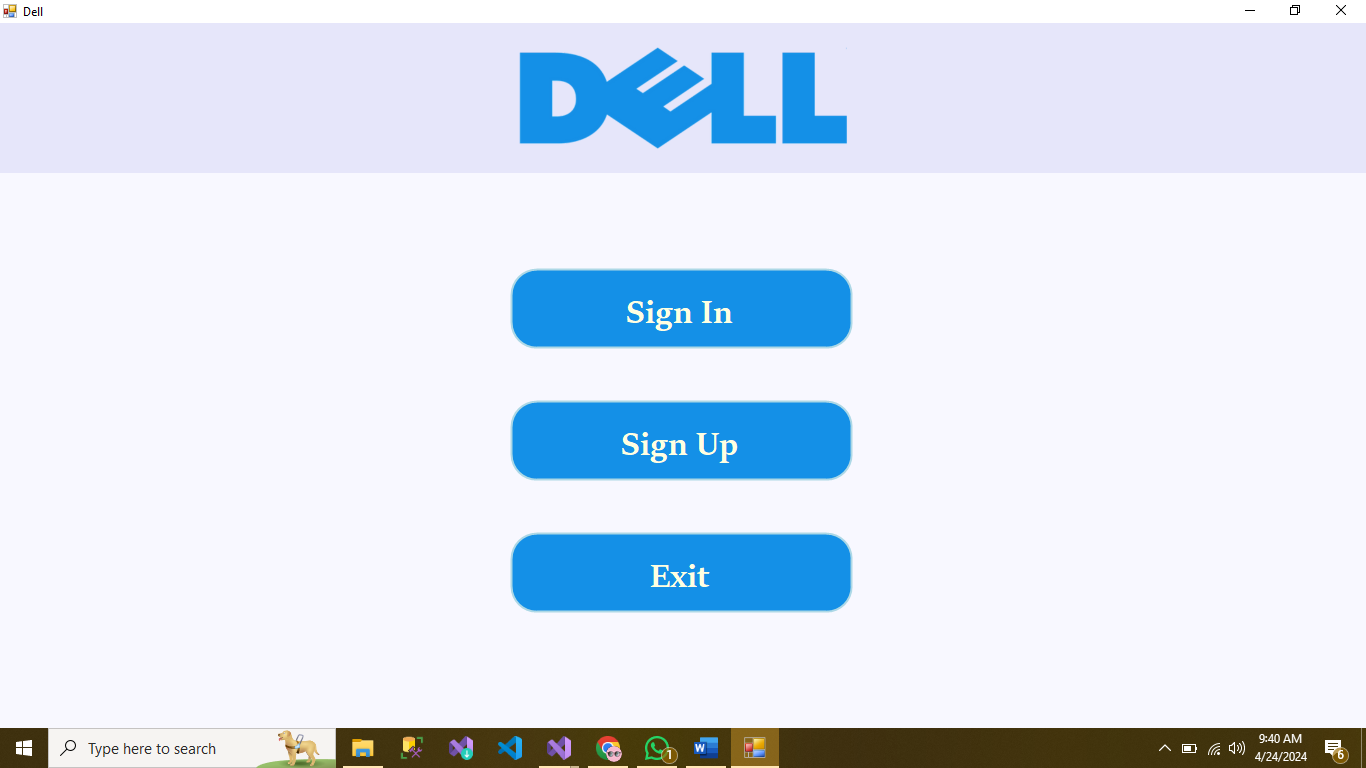
Majorly Four of interfaces in application i.e,

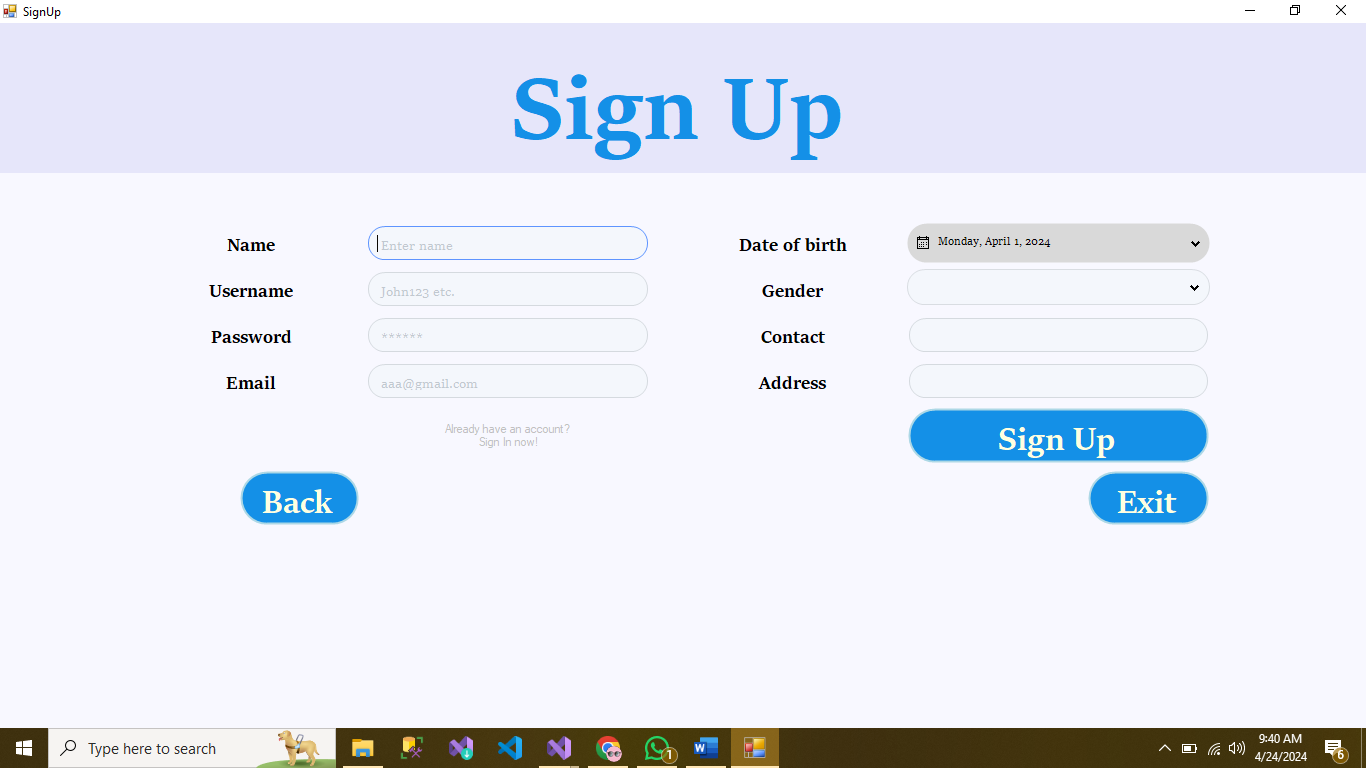
* **Login Interface.**
* **HomePage Interface.**
* **UserAccountPanel Interface.**
* **Console Wire frames:**

****

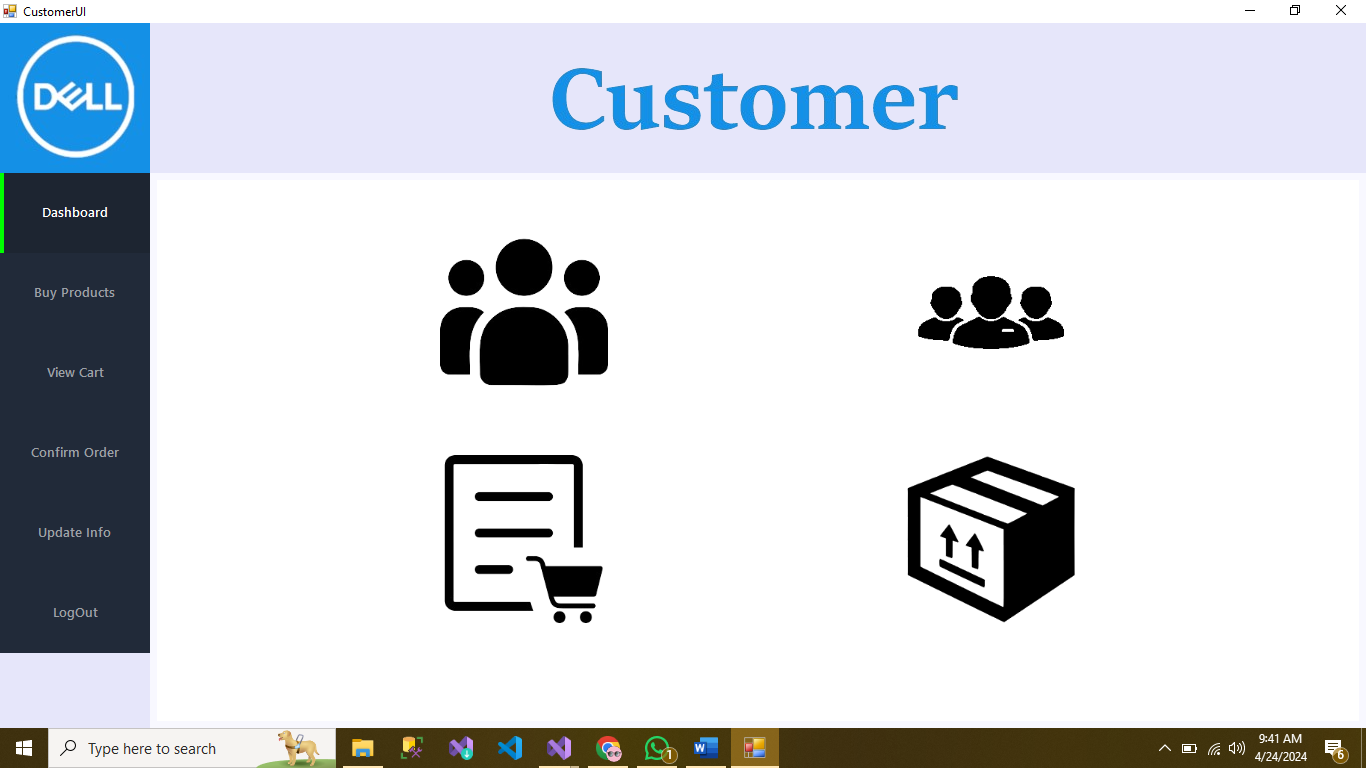


**Windows Form Wire frames:**









### Uniqueness:

* Unique thing I think in this application is that as you enter to make your account you have to give your gmail which is compulosory without gmail you can not create an account but as you give you mail a verification code is automatically send to the provided gmail which you have to enter as verification step with only correct code you can make account and use application.

### Code:

### BL Employee

using System;

namespace DellLibrary.BL

{

public class EmployeeBL : UserBL

{

private string designation; // Employee's job title

readonly private DateTime hireDate; // Date of employment

private DateTime resignationDate; // Date of resignation

// Full constructor with resignation date

public EmployeeBL(string name, string username, string password, string email, DateTime dob, string address, string contact, string gender, string status, string designation, DateTime hireDate, DateTime resignationDate) : base(name, username, password, email, dob, address, contact, gender, status)

{

this.designation = designation;

this.hireDate = hireDate;

this.resignationDate = resignationDate;

}

// Constructor without resignation date

public EmployeeBL(string name, string username, string password, string email, DateTime dob, string address, string contact, string gender, string status, string designation, DateTime hireDate) : base(name, username, password, email, dob, address, contact, gender, status)

{

this.designation = designation;

this.hireDate = hireDate;

}

// Constructor with basic details

public EmployeeBL(string name, string username, string password, string email, DateTime dob, string address, string contact, string gender) : base(name, username, password, email, dob, address, contact, gender)

{

}

public EmployeeBL() { } // Default constructor

public EmployeeBL(string username, string password) : base(username, password) { } // Constructor with only username and password

public string GetDesignation() { return designation; } // Get employee designation

public void SetDesignation(string value) { designation = value; } // Set employee designation

public DateTime GetHireDate() { return hireDate; } // Get hire date

public DateTime GetResignationDate() { return resignationDate; } // Get resignation date

public void SetResignationDate(DateTime value) { resignationDate = value; } // Set resignation date

}

}

# DL Layer:

using DellLibrary.BL;

using DellLibrary.DL\_Interfaces;

using DellLibrary.Utility;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Net.NetworkInformation;

namespace DellLibrary.DL.DB

{

public class EmployeeDLDB : IUserDL, IEmployeeDL

{

public string AddEmployee(EmployeeBL user) // adds employee to DB

{

string message = Validations.IsValidNewUser(user); // checks if user is valid or not

// if the user is valid

if (message == "True")

{

// query to add employee

string query = "INSERT INTO Employees (Name, Username, Password, Email, DOB, Address, Contact, Gender, Status, Designation, HireDate) " +

"VALUES (@Name, @Username, @Password, @Email, @DOB, @Address, @Contact, @Gender, @Status, @Designation, @HireDate)";

// connection to the database

using (SqlConnection con = Configuration.GetConnection())

{

try

{

con.Open();

SqlCommand command = new SqlCommand(query, con);

// Add parameters

command.Parameters.AddWithValue("@Name", user.GetName());

command.Parameters.AddWithValue("@Username", user.GetUsername());

command.Parameters.AddWithValue("@Password", user.GetPassword());

command.Parameters.AddWithValue("@Email", user.GetEmail());

command.Parameters.AddWithValue("@DOB", user.GetDob());

command.Parameters.AddWithValue("@Address", user.GetAddress());

command.Parameters.AddWithValue("@Contact", user.GetContact());

command.Parameters.AddWithValue("@Gender", user.GetGender());

command.Parameters.AddWithValue("@Status", user.GetStatus());

command.Parameters.AddWithValue("@Designation", user.GetDesignation());

command.Parameters.AddWithValue("@HireDate", user.GetHireDate());

// execute command

int rowsAffected = command.ExecuteNonQuery();

if (rowsAffected > 0) // if the employee was added

{

message = "True";

}

}

catch (Exception e) // if error occurs

{

message = e.Message;

}

finally // Close the database connection at end

{

con.Close();

}

}

}

return message; // return the result message

}

public string UpdateEmployee(EmployeeBL user, string username, string email) // updates employee data

{

string message;

// Determine if the user is a CEO

bool isCEO = user.GetDesignation() == "CEO";

// Validate user information

message = Validations.IsValidUpdatedUser(user, username, email, isCEO);

// If the user is valid

if (message == "True")

{

// Query to update employee

string query = "UPDATE Employees SET Name=@Name, Username=@Username, Password=@Password, Email=@Email, DOB=@DOB, Address=@Address, Contact=@Contact, Gender=@Gender WHERE Username=@user";

// Connection to the database

using (SqlConnection con = Configuration.GetConnection())

{

try

{

con.Open();

SqlCommand command = new SqlCommand(query, con);

// Add parameters

command.Parameters.AddWithValue("@Name", user.GetName());

command.Parameters.AddWithValue("@Username", user.GetUsername());

command.Parameters.AddWithValue("@user", username);

command.Parameters.AddWithValue("@Password", user.GetPassword());

command.Parameters.AddWithValue("@Email", user.GetEmail());

command.Parameters.AddWithValue("@DOB", user.GetDob());

command.Parameters.AddWithValue("@Address", user.GetAddress());

command.Parameters.AddWithValue("@Contact", user.GetContact());

command.Parameters.AddWithValue("@Gender", user.GetGender());

// Execute command

int rowsAffected = command.ExecuteNonQuery();

if (rowsAffected > 0) // If the employee was updated

{

message = "True";

}

}

catch (Exception e) // If an error occurs

{

message = e.Message;

}

finally // Close the database connection at the end

{

con.Close();

}

}

}

return message; // Return the result message

}

public string RemoveEmployee(string username) // removes employee

{

string message = "";

// makes connection with DB to remove employee

using (SqlConnection con = Configuration.GetConnection())

{

// first try to execute delete command

string query = $"DELETE Employees where Username=@Username;";

try

{

con.Open(); // opens Database Connection

SqlCommand command = new SqlCommand(query, con); // command to execute the query

// Add parameters

command.Parameters.AddWithValue("@Username", username);

SqlDataReader sqlDataReader = command.ExecuteReader(); // Execute the query

int rowAffected = sqlDataReader.RecordsAffected;

if (rowAffected>0)

{

message="True";

}

}

// if any exception returns the exception message

catch (Exception e)

{

message = e.Message;

}

finally // closes the database connection at the end

{

con.Close();

}

}

// returns the message

return message;

}

public string DeactivateEmployeeAccount(string username) // deactivates employee account

{

string message="";

// Query to update employee

string query = "UPDATE Employees SET Status='Deactivated' WHERE Username=@username;";

// Connection to the database

using (SqlConnection con = Configuration.GetConnection())

{

try

{

con.Open();

SqlCommand command = new SqlCommand(query, con);

// Add parameters

command.Parameters.AddWithValue("@username", username);

// Execute command

int rowsAffected = command.ExecuteNonQuery();

if (rowsAffected > 0) // If the employee was updated

{

message = "True";

}

}

catch (Exception e) // If an error occurs

{

message = e.Message;

}

finally // Close the database connection at the end

{

con.Close();

}

}

return message; // Return the result message

}

public string ActivateEmployeeAccount(string username) // activates employee account

{

string message = "";

// Query to update employee

string query = "UPDATE Employees SET Status='Active' WHERE Username=@username;";

// Connection to the database

using (SqlConnection con = Configuration.GetConnection())

{

try

{

con.Open();

SqlCommand command = new SqlCommand(query, con);

// Add parameters

command.Parameters.AddWithValue("@username", username);

// Execute command

int rowsAffected = command.ExecuteNonQuery();

if (rowsAffected > 0) // If the employee was updated

{

message = "True";

}

}

catch (Exception e) // If an error occurs

{

message = e.Message;

}

finally // Close the database connection at the end

{

con.Close();

}

}

return message; // Return the result message

}

public List<EmployeeBL> GetAllEmployees() // returns all employees list

{

List<EmployeeBL> Employees = new List<EmployeeBL>();

// makes connection with DB to get employees

using (SqlConnection con = Configuration.GetConnection())

{

string query = $"Select \* from Employees where Designation<>'CEO';";

// first try to execute retreive command

try

{

con.Open(); // opens Database Connection

SqlCommand command = new SqlCommand(query, con); // command to execute the query

SqlDataReader sqlDataReader = command.ExecuteReader(); // Execute the query

while (sqlDataReader.Read()) // if employees data found

{

if (sqlDataReader.IsDBNull(11)) // if resignation date is null

{

EmployeeBL employee = new EmployeeBL(sqlDataReader.GetString(0), sqlDataReader.GetString(1), sqlDataReader.GetString(2), sqlDataReader.GetString(3), sqlDataReader.GetDateTime(4), sqlDataReader.GetString(5), sqlDataReader.GetString(6), sqlDataReader.GetString(7), sqlDataReader.GetString(8), sqlDataReader.GetString(9), sqlDataReader.GetDateTime(10));

Employees.Add(employee);

}

else

{

EmployeeBL employee = new EmployeeBL(sqlDataReader.GetString(0), sqlDataReader.GetString(1), sqlDataReader.GetString(2), sqlDataReader.GetString(3), sqlDataReader.GetDateTime(4), sqlDataReader.GetString(5), sqlDataReader.GetString(6), sqlDataReader.GetString(7), sqlDataReader.GetString(8), sqlDataReader.GetString(9), sqlDataReader.GetDateTime(10), sqlDataReader.GetDateTime(11));

Employees.Add(employee);

}

}

}

catch (Exception e) // if any exception returns the exception message

{

throw (e);

}

finally // closes the database connection at the end

{

con.Close();

}

}

return Employees; // returns list

}

public EmployeeBL GetEmployeebyUsername(string username) // returns employee for a username

{

EmployeeBL employee = null;

// makes connection with DB to get employees

using (SqlConnection con = Configuration.GetConnection())

{

string query = $"Select \* from Employees where Username=@username and designation!='CEO';";

// first try to execute retreive command

try

{

con.Open(); // opens Database Connection

SqlCommand command = new SqlCommand(query, con); // command to execute the query

command.Parameters.AddWithValue("@Username", username);

SqlDataReader sqlDataReader = command.ExecuteReader(); // Execute the query

while (sqlDataReader.Read()) // if employees data found

{

if (sqlDataReader.IsDBNull(11)) // if resignation date is null

{

employee = new EmployeeBL(sqlDataReader.GetString(0), username, sqlDataReader.GetString(2), sqlDataReader.GetString(3), sqlDataReader.GetDateTime(4), sqlDataReader.GetString(5), sqlDataReader.GetString(6), sqlDataReader.GetString(7), sqlDataReader.GetString(8), sqlDataReader.GetString(9), sqlDataReader.GetDateTime(10));

}

else

{

employee = new EmployeeBL(sqlDataReader.GetString(0), username, sqlDataReader.GetString(2), sqlDataReader.GetString(3), sqlDataReader.GetDateTime(4), sqlDataReader.GetString(5), sqlDataReader.GetString(6), sqlDataReader.GetString(7), sqlDataReader.GetString(8), sqlDataReader.GetString(9), sqlDataReader.GetDateTime(10), sqlDataReader.GetDateTime(11));

}

}

}

catch (Exception e) // if any exception returns the exception message

{

throw (e);

}

finally // closes the database connection at the end

{

con.Close();

}

}

return employee; // returns employee

}

public List<EmployeeBL> GetAllEmployeesByStatus(string eStatus) // returns the list of employees acc to current status

{

List<EmployeeBL> Employees = new List<EmployeeBL>();

// makes connection with DB to get employees

using (SqlConnection con = Configuration.GetConnection())

{

string query = $"Select \* from Employees where Designation<>'CEO' and status = @estatus;";

// first try to execute retreive command

try

{

con.Open(); // opens Database Connection

SqlCommand command = new SqlCommand(query, con); // command to execute the query

command.Parameters.AddWithValue("@estatus", eStatus);

SqlDataReader sqlDataReader = command.ExecuteReader(); // Execute the query

while (sqlDataReader.Read()) // if employees data found

{

if (sqlDataReader.IsDBNull(11)) // if resignation date is null

{

EmployeeBL employee = new EmployeeBL(sqlDataReader.GetString(0), sqlDataReader.GetString(1), sqlDataReader.GetString(2), sqlDataReader.GetString(3), sqlDataReader.GetDateTime(4), sqlDataReader.GetString(5), sqlDataReader.GetString(6), sqlDataReader.GetString(7), sqlDataReader.GetString(8), sqlDataReader.GetString(9), sqlDataReader.GetDateTime(10));

Employees.Add(employee);

}

else

{

EmployeeBL employee = new EmployeeBL(sqlDataReader.GetString(0), sqlDataReader.GetString(1), sqlDataReader.GetString(2), sqlDataReader.GetString(3), sqlDataReader.GetDateTime(4), sqlDataReader.GetString(5), sqlDataReader.GetString(6), sqlDataReader.GetString(7), sqlDataReader.GetString(8), sqlDataReader.GetString(9), sqlDataReader.GetDateTime(10), sqlDataReader.GetDateTime(11));

Employees.Add(employee);

}

}

}

catch (Exception e) // if any exception returns the exception message

{

throw (e);

}

finally // closes the database connection at the end

{

con.Close();

}

}

return Employees; // returns list

}

public List<EmployeeBL> GetEmployeesByDesignation(string designation,string status) // returns the list of employees with specific designation and status

{

List<EmployeeBL> Employees = new List<EmployeeBL>();

// makes connection with DB to get employees

using (SqlConnection con = Configuration.GetConnection())

{

string query = $"Select \* from Employees where Designation=@designation and Status=@status;";

// first try to execute retreive command

try

{

con.Open(); // opens Database Connection

SqlCommand sqlCommand = new SqlCommand(query, con);

SqlCommand command = sqlCommand; // command to execute the query

// Add parameters

command.Parameters.AddWithValue("@designation", designation);

command.Parameters.AddWithValue("@status", status);

SqlDataReader sqlDataReader = command.ExecuteReader(); // Execute the query

while (sqlDataReader.Read())

{

string name = sqlDataReader.GetString(0);

string username = sqlDataReader.GetString(1);

string password = sqlDataReader.GetString(2);

string email = sqlDataReader.GetString(3);

DateTime birthDate = sqlDataReader.GetDateTime(4);

string address = sqlDataReader.GetString(5);

string contact = sqlDataReader.GetString(6);

string gender = sqlDataReader.GetString(7);

DateTime hireDate = sqlDataReader.GetDateTime(10);

EmployeeBL employee = new EmployeeBL(name, username,password,email,birthDate,address,contact,gender,status,designation,hireDate);

Employees.Add(employee);

}

}

catch (Exception e) // if any exception returns the exception message

{

throw (e);

}

finally // closes the database connection at the end

{

con.Close();

}

}

return Employees; // returns list

}

public bool UniqueAttributeCheck(string text, string attribute) // checks database for a unique attribute

{

// variable for checking attribute

bool check = false;

// query

string Query = $"Select \* from Employees where {attribute}='{text}';";

using (SqlConnection con = Configuration.GetConnection()) // connection to database

{

try

{

con.Open();

SqlCommand command = new SqlCommand(Query, con); // command to execute query

SqlDataReader sqlDataReader = command.ExecuteReader(); // datareader

if (sqlDataReader.Read()) // if attribute found

{

check = true;

}

}

catch (Exception)

{

check = true;

}

finally // Close the connection at end

{

con.Close();

}

}

// Return the result of the check

return check;

}

public UserBL UserSignIn(UserBL user) // checks user in database for signing in

{

EmployeeBL employee = null;

// query to find user in the database

string query = $"SELECT \* FROM Employees WHERE Username COLLATE Latin1\_General\_BIN = @Username AND Password COLLATE Latin1\_General\_BIN = @Password AND Status='Active';";

using (SqlConnection con = Configuration.GetConnection()) // Connection to the database

{

try

{

con.Open();

SqlCommand command = new SqlCommand(query, con); // command to execute the query

// Add parameters

command.Parameters.AddWithValue("@Username", user.GetUsername());

command.Parameters.AddWithValue("@Password", user.GetPassword());

SqlDataReader sqlDataReader = command.ExecuteReader(); // datareader

if (sqlDataReader.Read() && sqlDataReader.IsDBNull(11)) // if employee was found && resignation date is null

{

string designation = sqlDataReader.GetString(9);

employee = new EmployeeBL(sqlDataReader.GetString(0), sqlDataReader.GetString(1), sqlDataReader.GetString(2), sqlDataReader.GetString(3), sqlDataReader.GetDateTime(4), sqlDataReader.GetString(5), sqlDataReader.GetString(6), sqlDataReader.GetString(7), sqlDataReader.GetString(8), sqlDataReader.GetString(9), sqlDataReader.GetDateTime(10));

}

}

catch (Exception ex) // throw exception in case of errors

{

throw (ex);

}

finally

{

con.Close();

}

}

return employee; // return the result message

}

public int GetEmployeeCount() // returns count of total employees in database

{

int EmployeeCount = 0;

// makes connection with DB to get employees count

using (SqlConnection con = Configuration.GetConnection())

{

string query = $"Select Count(\*) from Employees;";

// first try to execute retreive command

try

{

con.Open(); // opens Database Connection

SqlCommand command = new SqlCommand(query, con); // command to execute the query

command.Parameters.AddWithValue("@Status", "Active"); // add parameters

SqlDataReader sqlDataReader = command.ExecuteReader(); // Execute the query

if (sqlDataReader.Read()) // if employees data found

{

EmployeeCount= sqlDataReader.GetInt32(0);

}

}

catch (Exception e) // if any exception returns the exception message

{

throw (e);

}

finally // closes the database connection at the end

{

con.Close();

}

}

return EmployeeCount; // returns count

}

}

}

FH:

using DellLibrary.BL;

using DellLibrary.DL\_Interfaces;

using DellLibrary.Utility;

using System;

using System.Collections.Generic;

using System.IO;

namespace DellLibrary.DL.FH

{

public class EmployeeDLFH : IUserDL, IEmployeeDL

{

private string filePath = "D:\\employee.txt"; // Path to the CSV file

public string AddEmployee(EmployeeBL user)

{

string message = Validations.IsValidNewUser(user); // Validate user

if (message == "True")

{

try

{

// Construct CSV line

string newEmployee = $"{user.GetName()},{user.GetUsername()},{user.GetPassword()},{user.GetEmail()},{user.GetDob()},{user.GetAddress()},{user.GetContact()},{user.GetGender()},{user.GetStatus()},{user.GetDesignation()},{user.GetHireDate()}";

// Append to CSV file

File.AppendAllText(filePath, newEmployee + Environment.NewLine);

message = "True"; // Success

}

catch (Exception e)

{

message = e.Message; // Error occurred

}

}

return message; // Return result message

}

public string UpdateEmployee(EmployeeBL user, string username, string email)

{

string message;

// Determine if the user is a CEO

bool isCEO = user.GetDesignation() == "CEO";

// Validate user information

message = Validations.IsValidUpdatedUser(user, username, email, isCEO);

// If the user is valid

if (message == "True")

{

try

{

string[] lines = File.ReadAllLines(filePath); // Read all lines from the CSV file

for (int i = 0; i < lines.Length; i++)

{

string[] parts = lines[i].Split(','); // Split the line into parts

// Check if the username matches

if (parts[1] == username)

{

// Update the user's information

lines[i] = $"{user.GetName()},{user.GetUsername()},{user.GetPassword()},{user.GetEmail()},{user.GetDob()},{user.GetAddress()},{user.GetContact()},{user.GetGender()}";

break; // Exit the loop since the user is found

}

}

// Write the updated lines back to the file

File.WriteAllLines(filePath, lines);

message = "True"; // Success

}

catch (Exception e)

{

message = e.Message; // Error occurred

}

}

return message; // Return the result message

}

public string RemoveEmployee(string username)

{

string message = "";

try

{

string[] lines = File.ReadAllLines(filePath); // Read all lines from the CSV file

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

foreach (string line in lines)

{

string[] parts = line.Split(',');

if (parts[1] != username) // If the username does not match, keep the line

{

updatedLines.Add(line);

}

}

// Write the updated lines back to the file

File.WriteAllLines(filePath, updatedLines);

message = "True"; // Success

}

catch (Exception e)

{

message = e.Message; // Error occurred

}

return message; // Return the result message

}

public List<EmployeeBL> GetAllEmployees()

{

List<EmployeeBL> employees = new List<EmployeeBL>();

try

{

// Read all lines from the CSV file

string[] lines = File.ReadAllLines(filePath);

// Loop through each line

foreach (string line in lines)

{

string[] parts = line.Split(','); // Split the line into parts

// Check if the designation is not CEO and if there's no resignation date

if (parts[9] != "CEO" && string.IsNullOrEmpty(parts[11]))

{

// Create an EmployeeBL object based on the data in the CSV line

EmployeeBL employee;

if (parts.Length >= 12 && !string.IsNullOrEmpty(parts[11]))

{

employee = new EmployeeBL(parts[0], parts[1], parts[2], parts[3], DateTime.Parse(parts[4]), parts[5], parts[6], parts[7], parts[8], parts[9], DateTime.Parse(parts[10]), DateTime.Parse(parts[11]));

}

else

{

employee = new EmployeeBL(parts[0], parts[1], parts[2], parts[3], DateTime.Parse(parts[4]), parts[5], parts[6], parts[7], parts[8], parts[9], DateTime.Parse(parts[10]));

}

employees.Add(employee); // Add the employee to the list

}

}

}

catch (Exception e)

{

throw e; // Throw any exceptions that occur

}

return employees; // Return the list of employees

}

public string ActivateEmployeeAccount(string username)

{

string message = "";

try

{

// Read all lines from the CSV file

string[] lines = File.ReadAllLines(filePath);

// Loop through each line

for (int i = 0; i < lines.Length; i++)

{

string[] parts = lines[i].Split(','); // Split the line into parts

// Check if the username matches

if (parts.Length > 1 && parts[1] == username)

{

// Update the status to 'Active'

parts[8] = "Active";

// Join the parts back into a line

lines[i] = string.Join(",", parts);

// Write the updated lines back to the file

File.WriteAllLines(filePath, lines);

message = "True"; // Success

break; // Exit the loop since the user is found

}

}

}

catch (Exception e)

{

message = e.Message; // Error occurred

}

return message; // Return the result message

}

public string DeactivateEmployeeAccount(string username)

{

string message = "";

try

{

string[] lines = File.ReadAllLines(filePath); // Read all lines from the CSV file

for (int i = 0; i < lines.Length; i++)

{

string[] parts = lines[i].Split(','); // Split the line into parts

// Check if the username matches

if (parts[1] == username)

{

// Update the user's status

parts[8] = "Deactivated";

lines[i] = string.Join(",", parts); // Join parts back into a line

break; // Exit the loop since the user is found

}

}

// Write the updated lines back to the file

File.WriteAllLines(filePath, lines);

message = "True"; // Success

}

catch (Exception e)

{

message = e.Message; // Error occurred

}

return message; // Return the result message

}

public List<EmployeeBL> GetEmployeesByDesignation(string designation, string status)

{

List<EmployeeBL> employees = new List<EmployeeBL>();

try

{

string[] lines = File.ReadAllLines(filePath); // Read all lines from the CSV file

foreach (string line in lines)

{

string[] parts = line.Split(','); // Split the line into parts

if (parts[9] == designation && parts[8] == status) // Check if designation and status match

{

// Create an EmployeeBL object

EmployeeBL employee = new EmployeeBL(parts[0], parts[1], parts[2], parts[3], DateTime.Parse(parts[4]), parts[5], parts[6], parts[7], parts[8], parts[9], DateTime.Parse(parts[10]));

employees.Add(employee);

}

}

}

catch (Exception)

{

// Handle exceptions if needed

}

return employees; // Return the list of employees

}

public EmployeeBL GetEmployeebyUsername(string username)

{

EmployeeBL employee = null;

try

{

string[] lines = File.ReadAllLines(filePath); // Read all lines from the CSV file

foreach (string line in lines)

{

string[] parts = line.Split(','); // Split the line into parts

// Check if the username matches

if (parts[1] == username)

{

// Create an EmployeeBL object

employee = new EmployeeBL(parts[0], parts[1], parts[2], parts[3], DateTime.Parse(parts[4]), parts[5], parts[6], parts[7], parts[8], parts[9], DateTime.Parse(parts[10]));

break; // Exit the loop since the user is found

}

}

}

catch (Exception)

{

// Handle exceptions if needed

}

return employee; // Return the employee object

}

public List<EmployeeBL> GetAllEmployeesByStatus(string status)

{

List<EmployeeBL> employees = new List<EmployeeBL>();

try

{

string[] lines = File.ReadAllLines(filePath); // Read all lines from the CSV file

foreach (string line in lines)

{

string[] parts = line.Split(','); // Split the line into parts

if (parts[8] == status) // Check if the status matches

{

// Create an EmployeeBL object

EmployeeBL employee = new EmployeeBL(parts[0], parts[1], parts[2], parts[3], DateTime.Parse(parts[4]), parts[5], parts[6], parts[7], parts[8], parts[9], DateTime.Parse(parts[10]));

employees.Add(employee);

}

}

}

catch (Exception)

{

// Handle exceptions if needed

}

return employees; // Return the list of employees

}

public bool UniqueAttributeCheck(string text, string attribute)

{

bool check = false;

try

{

string[] lines = File.ReadAllLines(filePath); // Read all lines from the CSV file

foreach (string line in lines)

{

string[] parts = line.Split(','); // Split the line into parts

if (parts[1] == text) // Check if the attribute matches

{

check = true; // Attribute found, it's not unique

break;

}

}

}

catch (Exception)

{

// Handle exceptions if needed

}

return check; // Return the result of the check

}

public UserBL UserSignIn(UserBL user)

{

EmployeeBL employee = null;

try

{

string[] lines = File.ReadAllLines(filePath); // Read all lines from the CSV file

foreach (string line in lines)

{

string[] parts = line.Split(','); // Split the line into parts

// Check if the username and password match

if (parts[1] == user.GetUsername() && parts[2] == user.GetPassword() && parts[8] == "Active")

{

// Create an EmployeeBL object

employee = new EmployeeBL(parts[0], parts[1], parts[2], parts[3], DateTime.Parse(parts[4]), parts[5], parts[6], parts[7], parts[8], parts[9], DateTime.Parse(parts[10]));

break; // Exit the loop since the user is found

}

}

}

catch (Exception)

{

// Handle exceptions if needed

}

return employee; // Return the employee object

}

public int GetEmployeeCount()

{

int employeeCount = 0;

try

{

string[] lines = File.ReadAllLines(filePath); // Read all lines from the CSV file

// Increment count for each active employee

foreach (string line in lines)

{

string[] parts = line.Split(','); // Split the line into parts

if (parts[8] == "Active") // Check if the status is active

{

employeeCount++;

}

}

}

catch (Exception)

{

// Handle exceptions if needed

}

return employeeCount; // Return the count

}

}

}

using System;

using System.Collections.Generic;

namespace DellConsole.UI

{

internal class AdminUI

{

// List of options available in the admin menu

private static List<string> menu = new List<string>() {"1.Add employee","2.Remove employee","3.Update employee", "4.View all employees","5.Exit"};

// Displays the admin menu options and returns the user's choice

public static string PrintAdminMenu()

{

// Display each menu option

foreach (string item in menu)

{

Console.WriteLine(item);

}

// Prompt the user to enter their choice

Console.Write("Enter option...");

// Return the user's choice

return Console.ReadLine();

}

}

}

using DellLibrary.BL;

using DellLibrary.Utility;

using System;

namespace DellConsole.UI

{

internal class EmployeeUI

{

// Takes inputs for new employee

public static EmployeeBL EmployeeInputs()

{

// Variables to store user inputs

string name, username, password, email, contact, gender, address;

DateTime dob;

// Loop for name input and validation

name = NameInput();

// Loop for username input and validation

username = UsernameInput();

// Loop for password input and validation

password = PasswordInput();

// Loop for email input and validation

email = EmailInput();

// Loop for date of birth input and validation

dob = DOBInput();

// Loop for gender input and validation

gender = GenderInput();

// Loop for contact input and validation

contact = ContactInput();

// Loop for address input

address = AddressInput();

// Create and return an EmployeeBL object with validated inputs

EmployeeBL employee = new EmployeeBL(name, username, password, email, dob, address, contact, gender, "Active", "SalesPerson", DateTime.Now);

return employee;

}

// Takes inputs to update existing employee

public static EmployeeBL UpdateEmployee(EmployeeBL employee,string email)

{

// Update name

employee.SetName(NameInput());

// Update password

employee.SetPassword(PasswordInput());

// Update email

employee.SetEmail(EmailInput(email));

// Update dob

employee.SetDob(DOBInput());

// Update gender

employee.SetGender(GenderInput());

// Update address

employee.SetAddress(AddressInput());

// Update contact

employee.SetContact(ContactInput());

return employee; // return the updated employee

}

// Method responsible for gathering and validating user's name input

private static string NameInput()

{

string name;

while (true)

{

// Prompt the user to enter their name

name = Utility.Input("name");

// Validate the entered name

string nameCheckResult = Validations.NameCheck(name);

// If validation fails, display error message and prompt again

if (nameCheckResult != "True")

{

Console.WriteLine(nameCheckResult);

}

// If validation succeeds, return value

else

{

return name;

}

}

}

// Method responsible for gathering and validating user's username input

private static string UsernameInput()

{

string username;

while (true)

{

// Prompt the user to enter their username

username = Utility.Input("username");

// Validate the entered username

string usernameCheckResult = Validations.UsernameCheck(username);

// If validation fails, display error message and prompt again

if (usernameCheckResult != "True")

{

Console.WriteLine(usernameCheckResult);

}

// If validation succeeds, return value

else

{

return username;

}

}

}

// Method responsible for gathering and validating user's password input

private static string PasswordInput()

{

string password;

while (true)

{

// Prompt the user to enter their password

password = Utility.Input("password");

// Validate the entered password

string passwordCheckResult = Validations.PasswordCheck(password);

// If validation fails, display error message and prompt again

if (passwordCheckResult != "True")

{

Console.WriteLine(passwordCheckResult);

}

// If validation succeeds, return value

else

{

return password;

}

}

}

// Method responsible for gathering and validating user's email input

private static string EmailInput()

{

string email;

while (true)

{

// Prompt the user to enter their email

email = Utility.Input("email");

// Validate the entered email

string emailCheckResult = Validations.EmailCheck(email);

// If validation fails, display error message and prompt again

if (emailCheckResult != "True")

{

Console.WriteLine(emailCheckResult);

}

// If validation succeeds, return value

else

{

return email;

}

}

}

// Method responsible for gathering and validating user's updated email input

private static string EmailInput(string email)

{

string email2;

while (true)

{

// Prompt the user to enter their email

email2 = Utility.Input("email");

// Validate the entered email

string emailCheckResult = Validations.UpdatedEmailCheck(email,email2);

// If validation fails, display error message and prompt again

if (emailCheckResult != "True")

{

Console.WriteLine(emailCheckResult);

}

// If validation succeeds, return value

else

{

return email;

}

}

}

// Method responsible for gathering and validating user's date of birth input

private static DateTime DOBInput()

{

DateTime dob;

while (true)

{

// Prompt the user to enter their date of birth

string dobInput = Utility.Input("DOB (YYYY-MM-DD)");

// Attempt to parse the entered date of birth

if (DateTime.TryParse(dobInput, out dob))

{

// Validate the parsed date of birth

string ageCheckResult = Validations.AgeCheck(dob);

// If validation fails, display error message and prompt again

if (ageCheckResult != "True")

{

Console.WriteLine(ageCheckResult);

}

// If validation succeeds, return value

else

{

return dob;

}

}

// If parsing fails, display error message and prompt again

else

{

Console.WriteLine("Invalid date format. Please enter in YYYY-MM-DD format.");

}

}

}

// Method responsible for gathering and validating user's gender input

private static string GenderInput()

{

string gender;

while (true)

{

// Prompt the user to enter their gender

Console.Write("Enter gender (F/M): ");

gender = Console.ReadLine().ToLower();

// Validate the entered gender

if (gender == "f" || gender == "m" || gender == "female" || gender == "male")

{

// Normalize the gender value

gender = gender == "f" || gender == "female" ? "Female" : "Male";

return gender;

}

// If validation fails, display error message and prompt again

else

{

Console.WriteLine("Invalid gender input. Please enter 'F' or 'M'.");

}

}

}

// Method responsible for gathering and validating user's contact input

private static string ContactInput()

{

string contact;

while (true)

{

// Prompt the user to enter their contact information

Console.Write("Enter contact: ");

contact = Console.ReadLine();

// Validate the entered contact information

string contactCheckResult = Validations.ContactCheck(contact);

// If validation fails, display error message and prompt again

if (contactCheckResult != "True")

{

Console.WriteLine(contactCheckResult);

}

// If validation succeeds, exit the loop

else

{

return contact;

}

}

}

// Method responsible for gathering user's address input

private static string AddressInput()

{

string address;

while (true)

{

// Prompt the user to enter their address

Console.Write("Enter address: ");

address = Console.ReadLine();

// If address is empty, display error message and prompt again

if (string.IsNullOrEmpty(address))

{

Console.WriteLine("Address cannot be empty.");

}

// If validation succeeds, return value

else

{

return address;

}

}

}

// Method responsible for printing an employee's info

public static void PrintEmployeeInfo(EmployeeBL employee)

{

// Print the employee's details

Console.WriteLine($"{employee.GetName(),-15}{employee.GetUsername(),-15}{employee.GetPassword(),-15}{employee.GetEmail(),-15}{employee.GetDob().ToString("yyyy-MM-dd"),-15}{employee.GetAddress(),-15}{employee.GetContact(),-15}{employee.GetGender(),-15}{employee.GetDesignation(),-15}{employee.GetHireDate().ToString("yyyy-MM-dd"),-15}");

}

}

}

using System;

namespace DellConsole.UI

{

internal class Utility

{

// Displays the main menu and returns the user's choice

public static string Menu()

{

Console.WriteLine("1.Manage employees\n2.Exit");

Console.Write("\nEnter option...");

return Console.ReadLine();

}

// Displays a message and prompts the user to press any key to continue

public static void PressAnyKeyToContinue(string message)

{

Console.WriteLine(message);

Console.Write("Press any key to continue...");

Console.ReadKey();

}

// Displays the header for the application

public static void Header()

{

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine(" \* Dell System \*");

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

// Displays the header for adding an employee

public static void AddEmpHeader()

{

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine(" \* Add Employee \*");

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

// Displays the header for removing an employee

public static void RemoveEmpHeader()

{

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine(" \* Remove Employee \*");

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

// Displays the header for managing employees

public static void ManageEmpHeader()

{

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine(" \* Managae Employee \*");

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

// Displays the header for updating an employee

public static void UpdateEmpHeader()

{

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine(" \* Update Employee \*");

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

// Displays the header for viewing employee data

public static void ViewEmpHeader()

{

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine(" \* View Employee \*");

Console.WriteLine(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

// Prompts the user to input a value for a specific attribute and returns the input

public static string Input(string attribute)

{

Console.Write("Enter "+attribute+": ");

return Console.ReadLine();

}

// Prompts the user to input option

public static string YesNoOption()

{

// Prompt the user for input

Console.WriteLine("\nDo you want to delete user data (Y/N)?");

// Keep looping until a valid input is received

while (true)

{

// Read the user's input from the console and convert it to lowercase

Console.Write("Enter option: ");

string option = Console.ReadLine().ToLower();

// Check if the input matches one of the valid options

if (option == "y" || option == "yes" || option == "no" || option == "n")

{

// Convert the input to uppercase (Y or N) for consistency

option = option == "yes" || option == "y" ? "Y" : "N";

// Return the valid option

return option;

}

}

}

}

}

using DellLibrary.DL.DB;

using DellLibrary.DL.FH;

using DellLibrary.DL\_Interfaces;

namespace DELLConsole.Utility

{

internal class ObjectHandler

{

// makes object of interfaces

private static readonly IEmployeeDL employeeDL = new EmployeeDLFH();

// private static readonly IEmployeeDL employeeDL = new EmployeeDLDB();

// returns the employeeDL object of interface

public static IEmployeeDL GetEmployeeDL() { return employeeDL; }

}

}

using DellConsole.UI;

using DELLConsole.Utility;

using DellLibrary.BL;

using System;

using System.Collections.Generic;

namespace DellConsole

{

internal class Program

{

static void Main()

{

while (true)

{

// Clear the console screen

Console.Clear();

// Display the header for the admin menu

Utility.Header();

// Display the main menu and get user's choice

string option = Utility.Menu();

// If the user chooses option 1 (Admin menu)

if (option == "1")

{

while (true)

{

// Clear the console screen

Console.Clear();

// Display the header for the manage employee menu

Utility.ManageEmpHeader();

// Print the admin menu and get user's choice

option = AdminUI.PrintAdminMenu();

// If the user chooses to add an employee

if (option == "1")

{

// Display header for adding employee

Console.Clear();

Utility.AddEmpHeader();

// Gather inputs for a new employee

EmployeeBL employee = EmployeeUI.EmployeeInputs();

// Add the employee to the data layer

string message = ObjectHandler.GetEmployeeDL().AddEmployee(employee);

// Display success or error message

if (message == "True")

{

Utility.PressAnyKeyToContinue("\nEmployee added successfully!");

}

else

{

Utility.PressAnyKeyToContinue(message);

}

}

// If the user chooses to remove an employee

else if (option == "2")

{

// Display header for removing employee

Console.Clear();

Utility.RemoveEmpHeader();

// Prompt for employee username

string username = Utility.Input("employee username");

try

{

// Attempt to get employee details

EmployeeBL employee = ObjectHandler.GetEmployeeDL().GetEmployeebyUsername(username);

if (employee != null)

{

// Display employee details and confirm deletion

Console.WriteLine($"\n\n{"Name",-15}{"Username",-15}{"Password",-15}{"Email",-15}{"DOB",-15}{"Address",-15}{"Contact",-15}{"Gender",-15}{"Designation",-15}{"Join Date",-15}");

EmployeeUI.PrintEmployeeInfo(employee);

string op = Utility.YesNoOption();

if (op == "Y")

{

// Remove employee from data layer

op = ObjectHandler.GetEmployeeDL().RemoveEmployee(username);

if (op == "True")

{

Utility.PressAnyKeyToContinue("\nEmployee deleted successfully!");

}

else

{

Utility.PressAnyKeyToContinue("\nEmployee data not deleted!");

}

}

else

{

Utility.PressAnyKeyToContinue("\nEmployee data not deleted!");

}

}

else

{

Utility.PressAnyKeyToContinue("\nEmployee data not found!");

}

}

catch (Exception ex)

{

Utility.PressAnyKeyToContinue(ex.Message);

}

}

// If the user chooses to update an employee

else if (option == "3")

{

// Display header for updating employee

Console.Clear();

Utility.UpdateEmpHeader();

// Gather username input to update employee data

string username = Utility.Input("employee username");

try

{

// Attempt to get employee details

EmployeeBL employee = ObjectHandler.GetEmployeeDL().GetEmployeebyUsername(username);

if (employee != null)

{

string email = employee.GetEmail();

// Display employee details and prompt for update

Console.WriteLine($"\n\n{"Name",-15}{"Username",-15}{"Password",-15}{"Email",-15}{"DOB",-15}{"Address",-15}{"Contact",-15}{"Gender",-15}{"Designation",-15}{"Join Date",-15}");

EmployeeUI.PrintEmployeeInfo(employee);

// Update employee information

employee = EmployeeUI.UpdateEmployee(employee, email);

string op = ObjectHandler.GetEmployeeDL().UpdateEmployee(employee, username, email);

if (op == "True")

{

Utility.PressAnyKeyToContinue("\nEmployee updated successfully!");

}

else

{

Utility.PressAnyKeyToContinue("\nEmployee data not updated!");

}

}

else

{

Utility.PressAnyKeyToContinue("\nEmployee data not found!");

}

}

catch (Exception ex)

{

Utility.PressAnyKeyToContinue(ex.Message);

}

}

// If the user chooses to view all employees

else if (option == "4")

{

// Display header for viewing all employees

Console.Clear();

Utility.ViewEmpHeader();

// Retrieve all employees with a specific designation

List<EmployeeBL> employees = ObjectHandler.GetEmployeeDL().GetEmployeesByDesignation("SalesPerson", "Active");

// Display all employees

Console.WriteLine($"\n\n{"Name",-15}{"Username",-15}{"Password",-15}{"Email",-15}{"DOB",-15}{"Address",-15}{"Contact",-15}{"Gender",-15}{"Designation",-15}{"Join Date",-15}");

foreach (EmployeeBL emp in employees)

{

EmployeeUI.PrintEmployeeInfo(emp);

}

Utility.PressAnyKeyToContinue("\nAll employees data!");

}

// If the user chooses to go back to the main menu

else if (option == "5")

{

break; // Exit the current loop to return to the main menu

}

// If the user enters an invalid option

else

{

Utility.PressAnyKeyToContinue("Wrong user input!");

}

}

}

// If the user chooses option 2 (Exit)

else if (option == "2")

{

// Exit the application

Environment.Exit(0);

}

// If the user enters an invalid option

else

{

Utility.PressAnyKeyToContinue("Wrong user input!");

}

}

}

}

}

# CRC:

# 