Database Connectivity Lab

Using SQLite as an embedded database

1 Beginning the lab

1. This lab consists of a .NET Framework console app. To begin the lab, create a new project named `DatabaseDemo` and place it in your lab directory.

2. Edit the `Program.cs` file as shown in listing 1.

Listing 1: Modify `Program.cs`

```csharp
using System;

namespace DatabaseDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            int userChoice = 0;
            string console = "";
            do
            {
                console = Util.GetConsole();
                Console.WriteLine(console);
                int.TryParse(Console.ReadLine(), out userChoice);
                if (userChoice == 0)
                    Util.ReconstituteDatabase();
                else if (userChoice == 1)
                    Util.EnterNewUser();
                else if (userChoice == 2)
                    Util.AuthenticateUser();
                else if (userChoice == 3)
                    Util.ChangePassword();
                else if (userChoice == 4)
                    Util.DeleteUser();
                else if (userChoice == 5)
                    Util.DisplayUsers();
                else if (userChoice == 9)
                    Environment.Exit(0);
                else
                    Console.WriteLine("Invalid choice, please try again.");
            } while (userChoice != 9);
        }
    }
}
```

2 Adding the `Util.cs` file

3. Add a class to the project in a new file. Name the class `Util.cs`.
4. Begin editing `Util.cs` as shown in listing 2. You will have to invoke the Nuget Package Manager and add the *System.Data.SQLite* package.

Listing 2: Begin editing `Program.cs`

```
using System;
using System.Text;
using System.Data.SQLite;
using System.Security.Cryptography;

namespace DatabaseDemo
{
    internal class Util
    {

    }
}
```

5. In the Main method, inside the `do ... while()` loop, the first statement gets a user interface (named `console`) and prints it. Write the `GetConsole()` method as shown in listing 3.

Listing 3: `GetConsole()`

```
internal static string GetConsole()
{
    StringBuilder console = new StringBuilder();
    console.Append("********************************************\n");
    console.Append("* Please enter one of the following choices:\n");
    console.Append("*\t0. To remove the users table and start over\n");
    console.Append("*\t1. To enter a new username/password pair\n");
    console.Append("*\t2. To authenticate a username\n");
    console.Append("*\t3. To change a user's password\n");
    console.Append("*\t4. To delete a user\n");
    console.Append("*\t5. To display all users and passwords\n");
    console.Append("*\t9. To exit the program\n");
    console.Append("********************************************\n");
    console.Append("Enter your choice :");
    return console.ToString();
}
```

6. Since we are starting without a database, the first thing we must do is create a database with the appropriate table. This task is handled by `ReconstituteDatabase()`. Add this method as shown in listig 4.

Listing 4: `ReconstituteDatabase()`

```
internal static void ReconstituteDatabase()
{
    dropTable();
    buildTable();
    Console.WriteLine("Users table reconstituted.");
}
```

7. The `ReconstituteDatabase()` method consists of two helper methods, `dropTable()` and `buildTable()`. Add these two methods as shown in listings **??** and 6. `dropTable()` drios te Users table if it exists.

Listing 5: `dropTable()`

```
private static void dropTable()
{
    Console.WriteLine("calling dropTable() ...");
    SQLiteConnection conn = CreateConnection();
    SQLiteCommand sqlite_cmd;
    string dropcommand = "drop table if exists Users";
    sqlite_cmd = conn.CreateCommand();
    sqlite_cmd.CommandText = dropcommand;
    int result = sqlite_cmd.ExecuteNonQuery();
    //Console.WriteLine($"result of drop table is {result}");
    conn.Close();
}
```

8. `buildTable()` creates table Users with two columns, `username` of type varchar(20) and `password` of type varchar(64).

Listing 6: `buildTable()`

```
private static void buildTable()
{
    Console.WriteLine("calling buildTable() ...");
    SQLiteConnection conn = CreateConnection();
    SQLiteCommand sqlite_cmd;
    string createtable = "CREATE TABLE Users (" +
                        "username nvarchar(50) primary key, " +
                        "password nvarchar(64) not null" +
                        ");";
    sqlite_cmd = conn.CreateCommand();
    sqlite_cmd.CommandText = createtable; ;
    int result =  sqlite_cmd.ExecuteNonQuery();
    //Console.WriteLine($"result of create table is {result}");
    conn.Close();
}
```

9. Each of the two previous methods, and all of the methods that run SQL queries, depend on a connection to the database. `CreateConnection()` is a void method that returns a database connection. *Note: every time we create a connection to the database, we must be sure to close the connection. Otherwise, we may run out of resources to connect to the database.*

Listing 7: `CreateConnection()`

```
private static SQLiteConnection CreateConnection()
{
    // Open the connection:
    //SQLiteConnection("Data Source= database.db; Version = 3; New = True; Compress = True;
        ");
    // Open the connection:
    SQLiteConnection sqlite_conn = new SQLiteConnection("Data Source= database.db; New =
        True; Compress = True; ");
    try
    {
        sqlite_conn.Open();
        Console.WriteLine("SQLite connection created ...");
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex);
    }
    return sqlite_conn;
}
```

3 SQL queries

Inn the business world, database functionality is oftenn reffered to as CRUD, that is, *create* (insert) data, *read* (select) data, `update` data, and *delete* data. The following methods implement these four functions.

10. Add the method that enters new data into the database, `EnterNewUser()`, as shown in listing 8.

Listing 8: `EnterNewUser()`

```
        internal static void EnterNewUser()
        {
            Console.WriteLine("Please enter a username and password for the new user: ");
            (string username, string password) = getUserPass();
            password = hashString(password);
            insertData(username, password);
        }
```

11. Methods that manipulate passwords hash the plain text passwords for security reasons. This functionality is implemented by ethod `hashString()`. Add that nethod as shown in listing 9.

Listing 9: `hashString()`

```
private static string hashString(string newPass)
{
SHA1Managed sha1 = new SHA1Managed();
byte[] data = sha1.ComputeHash(Encoding.UTF8.GetBytes(newPass));
StringBuilder sb = new StringBuilder();
for (int i = 0; i < data.Length; i++)
{
    sb.Append(data[i].ToString("x2"));
}
string h = sb.ToString();
return h;
}
```

12. Method `EnteerNewUser()` depends on a helper method, `insertData()`. Ijplement this method as shown in listing 10.

Listing 10: `insertData()`

```
private static void insertData(string u, string p)
{
    Console.WriteLine($"insertData({u}, {p})");
    SQLiteConnection conn = CreateConnection();
    SQLiteCommand sqlite_cmd = conn.CreateCommand();
    string query = $"INSERT INTO Users (username, password) VALUES(\"{u}\", \"{p}\");";
    sqlite_cmd.CommandText = query;
    int result = sqlite_cmd.ExecuteNonQuery();
    //Console.WriteLine($"result of insert is {result}");
    conn.Close();
    Console.WriteLine("Insert successfully executed ...");
}
```

13. Having entered a new user, we need somme means to view all of our users. Method `DisplayUsers()` ists all our users and the hashed password to the console. Implement the method as shown in listing 11.

Listing 11: `DisplayUsers()`

```
internal static void DisplayUsers()
{
    Console.WriteLine("calling DisplayUsers()  ...");
    SQLiteConnection conn = CreateConnection();
    SQLiteCommand sqlite_cmd = conn.CreateCommand();
    sqlite_cmd.CommandText = "SELECT * FROM Users";

    SQLiteDataReader  sqlite_datareader = sqlite_cmd.ExecuteReader();
    Console.WriteLine("===============================================");
    while (sqlite_datareader.Read())
    {
        string user = sqlite_datareader.GetString(0);
        string cred = sqlite_datareader.GetString(1);
        Console.WriteLine($"{user}, {cred}");
    }
    Console.WriteLine("===============================================");
    conn.Close();
}
```

14. In order to authenticate a user, we need to get the username and password from the console, hash the password, retrieve the username and password from the database, and compare the usernames and hashed passwords. Method `AuthenticateUser()` performs these tasks. Add the ethod as shown in listing 12.

Listing 12: `AuthenticateUser()`

```
internal static void AuthenticateUser()
{
    (string username, string password) = getUserPass();
    (string user, string pass) = getUser(username);
    password = hashString(password);
    if(username == user && password == pass)
        Console.WriteLine($"user {user} is authenticated");
    else
        Console.WriteLine($"user {user} is NOT authenticated");
}
```

15. `AuthenticateUser()` depends on a helper method named `gettUser()` that retrieves a specific user and password from the database. This method is shown in listing 13.

Listing 13: `getUser()`

```
private static (string, string) getUser(string u)
{
    Console.WriteLine("calling getUser()  ...");
    SQLiteConnection conn = CreateConnection();
    SQLiteCommand sqlite_cmd = conn.CreateCommand();
    string query = $"SELECT * FROM Users WHERE username like \"{u}\"";
    //Console.WriteLine(query);
    sqlite_cmd.CommandText = query;
    string user = "";
    string pass = "";
    SQLiteDataReader  sqlite_datareader = sqlite_cmd.ExecuteReader();
    while (sqlite_datareader.Read())
    {
        user = sqlite_datareader.GetString(0);
        pass = sqlite_datareader.GetString(1);
    }
    conn.Close();
    return (user, pass);
}
```

16. The method `getUserPass()` is a helper method that allows the user to enter the usernane and passord at the console. This is shown in listing 14.

Listing 14: `getUserPass()`

```
private static (string username, string password) getUserPass()
{
    Console.Write("Enter username: ");
    string username = Console.ReadLine();
    Console.Write("Enter password for user: ");
    string password = Console.ReadLine();
    return (username, password);
}
```

Listing 15: `ChangePassword()`

```
internal static void ChangePassword()
{
    (string username, string password) = getUserPass();
    password = hashString(password);
    (string user, string pass) = getUser(username);
    Console.WriteLine($"{username}/{password} ---- {user}/{pass}");
    if (username == user && password == pass)
        updatePassword(username);
    else
        Console.WriteLine("Sorry, cannot update password for user {username}");
}
```

Listing 16: `DeleteUser()`

```
internal static void DeleteUser()
{
    (string username, string password) = getUserPass();
    (string user, string pass) = getUser(username);
    password = hashString(password);
    if (username == user && password == pass)
        runDeleteQuery(username);
    else
        Console.WriteLine($"user {user} is NOT deleted");
}
```

Listing 17: `updatePassword()`

```
private static void updatePassword(string username)
{
    Console.WriteLine($"updatePassword({username})");
    Console.WriteLine($"Please enter a new password for {username}");
    string pwd = Console.ReadLine();
    pwd = hashString(pwd);

    SQLiteConnection conn = CreateConnection();
    SQLiteCommand sqlite_cmd = conn.CreateCommand();
    string query = $"UPDATE Users set password = \"{pwd}\" where username like \"{username
        }\";";
    Console.WriteLine(query);
    sqlite_cmd.CommandText = query;
    int result = sqlite_cmd.ExecuteNonQuery();
    Console.WriteLine($"result of update is {result}");
    conn.Close();
    Console.WriteLine("Password update successfully executed ...");
}
```

Listing 18: `runDeleteQuery()`

```
private static void runDeleteQuery(string username)
{
    Console.WriteLine($"deleteData({username})");
    SQLiteConnection conn = CreateConnection();
    SQLiteCommand sqlite_cmd = conn.CreateCommand();
    string query = $"DELETE FROM Users where username like \"{username}\";";
    sqlite_cmd.CommandText = query;
    int result = sqlite_cmd.ExecuteNonQuery();
    //Console.WriteLine($"result of delete is {result}");
    conn.Close();
    Console.WriteLine("Delete successfully executed ...");
}
```