

## **P6 – Password Hashing in User Authentication(.Net)**

### **Overview**

Task : Learn how to secure your password by applying a combination of Password Hash and Salt.

### **Objectives**

To explore how user password may be securely store in the database by means of password hashes.

## What is Password Hashing?

hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

hash("hollo") = 58756879c05c68dfac9866712fad6a93f8146f337a69af120fda8e90f383853542

hash("george") = c0e81794384491161f1777c232bc6bd9ec38f616560b e7dd238f3364946366

- Hash algorithms are one way functions.
- Output of a hash is a fixed-length "fingerprint" that cannot be reversed.
- Any changes to the source, the resulting hash is completely different (see the example above).
- Hashing makes helps to protect passwords, because we want to store passwords in a form that protects them even if the password file itself is compromised, but at the same time, we need to be able to verify that a user's password is correct.

The general steps for account registration and authentication in a hash-based account system is as follows:

1. The user creates an account. (***We will revisit the registration process in other practical***)
2. Their password is hashed and stored in the database. The actual password (un-hashed) is and should not be written to the hard drive.
3. When the user attempts to login, the hash of the password they entered is checked against the hash of their real password (retrieved from the database).
4. If the hashes match, the user is granted access. If not, the user is told they entered invalid login credentials.
  - You should not inform the user if the username or password is wrong. Always display a generic message like "Invalid username or password." This prevents attackers from enumerating valid usernames without knowing their passwords.
5. Repeat step 3 and 4 everytime someone tries to login to their account.

Only **cryptographic hash functions** may be used to implement password hashing. Hash functions like SHA256, SHA512, RipeMD, and WHIRLPOOL are cryptographic hash functions.

It is easy to think that all you have to do is run the password through a cryptographic hash function and your users' passwords will be secure. This is far from the truth. There are many ways to recover passwords from plain hashes very quickly.

There are several easy-to-implement techniques that make these "attacks" much less effective. To motivate the need for these techniques, consider this very website. On the front page, you

can submit a list of hashes to be cracked, and receive results in less than a second. Clearly, simply hashing the password does not meet our needs for security.

## The Basics: Hashing with Salt

Malicious hackers are able to crack plain hashes very quickly using lookup tables and rainbow tables. To mitigate the above issues, we need to randomize the hashing using salt. But how do we generate the salt, and how do we apply it to the password?

Salt should be generated using a **Cryptographically Secure Pseudo-Random Number Generator** (CSPRNG). As the name suggests, CSPRNGs are designed to be cryptographically secure, meaning they provide a high level of randomness and are completely unpredictable. We don't want our salts to be predictable, so we must use a CSPRNG. The following table lists some CSPRNGs that exist for some popular programming platforms.

Platform	CSPRNG
Java	<a href="#">java.security.SecureRandom</a>
Dot NET (C#, VB)	<a href="#">System.Security.Cryptography.RNGCryptoServiceProvider</a>

The salt needs to be unique per-user per-password. Every time a user creates an account or changes their password, the password should be hashed using a new random salt. Never reuse a salt. The salt also needs to be long, so that there are many possible salts. As a rule of thumb, make your salt is at least as long as the hash function's output. The salt should be stored in the user account table alongside the hash.

## ***To Store a Password***

1. Generate a long random salt using a CSPRNG.
2. Prepend the salt to the password and hash it with a **standard** cryptographic hash function such as SHA256.
3. Save both the salt and the hash in the user's database record.

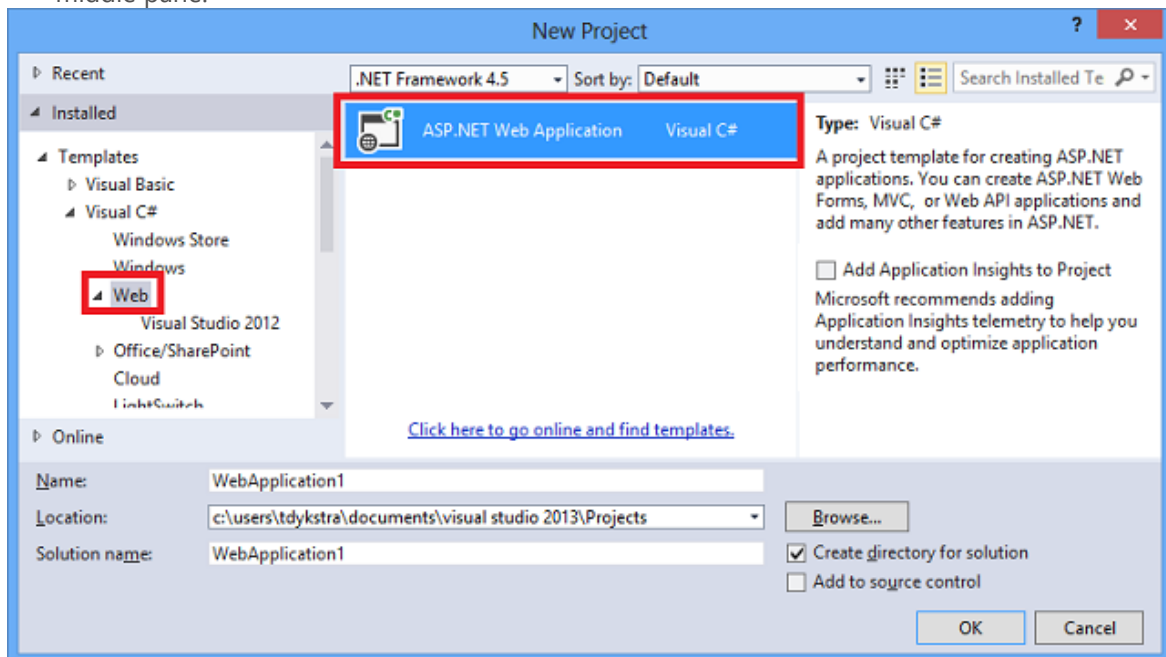
## ***To Validate a Password***

1. Retrieve the user's salt and hash from the database.
2. Prepend the salt to the given password and hash it using the same hash function.
3. Compare the hash of the given password with the hash from the database. If they match, the password is correct. Otherwise, the password is incorrect.

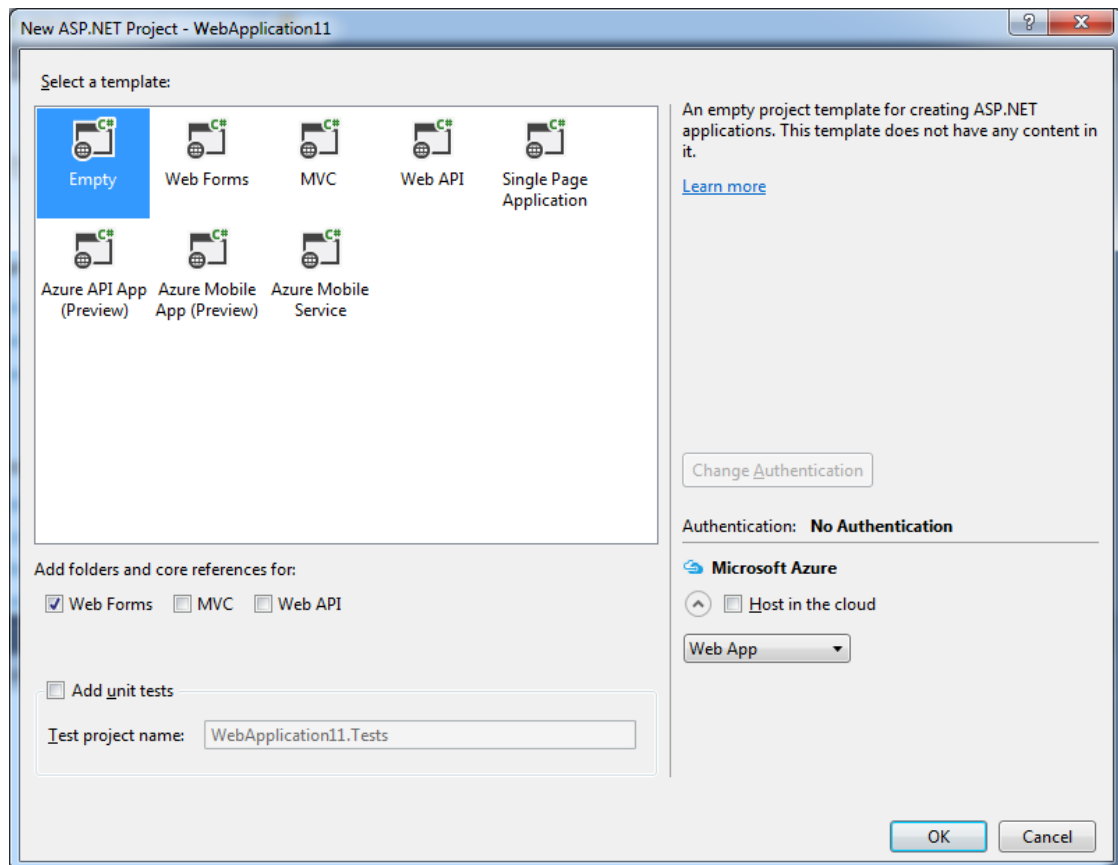
## Create a New website

The following steps show how to create a web project:

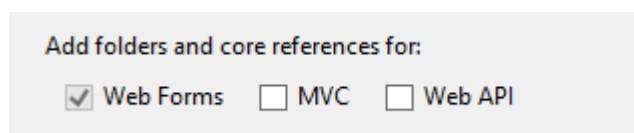
1. Click **New Project** in the **Start** page or in the **File** menu.
2. In the **New Project** dialog, click **Web** in the left pane and **ASP.NET Web Application** in the middle pane.



3. Specify project **Name**, **Location**, and other options, and then click **OK**.  
The **New ASP.NET Project** dialog appears.



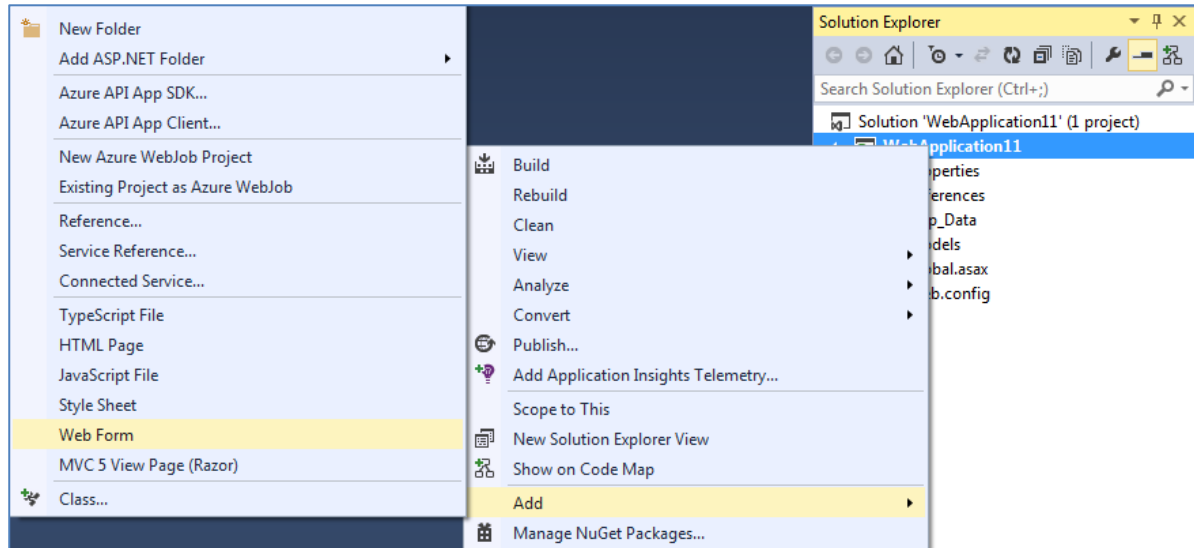
4. Click a template : Empty
5. If you want to add support for additional frameworks not included in the template, click the appropriate check box. (In the example shown, you could add MVC and/or Web API to a Web Forms project.)



## Create a Registration.aspx form

Add a new Web Form (C#) – Registration.aspx

Right-click application name → choose Add → web form



The screenshot shows a web browser window with the address bar displaying 'localhost:37375/Registration.aspx'. The page content is titled 'Account Registration' in a bold, black font. Below the title, there are five input fields for registration details: 'Email (UserID)', 'Password', 'Confirm Password', 'NRIC', and 'Mobile (+65)'. Each field is represented by a text box. At the bottom of the form, there is a 'Submit' button.

### Textboxes

Email – tb\_userid

Password – tb\_pwd

Confirm Password - tb\_cfpwd

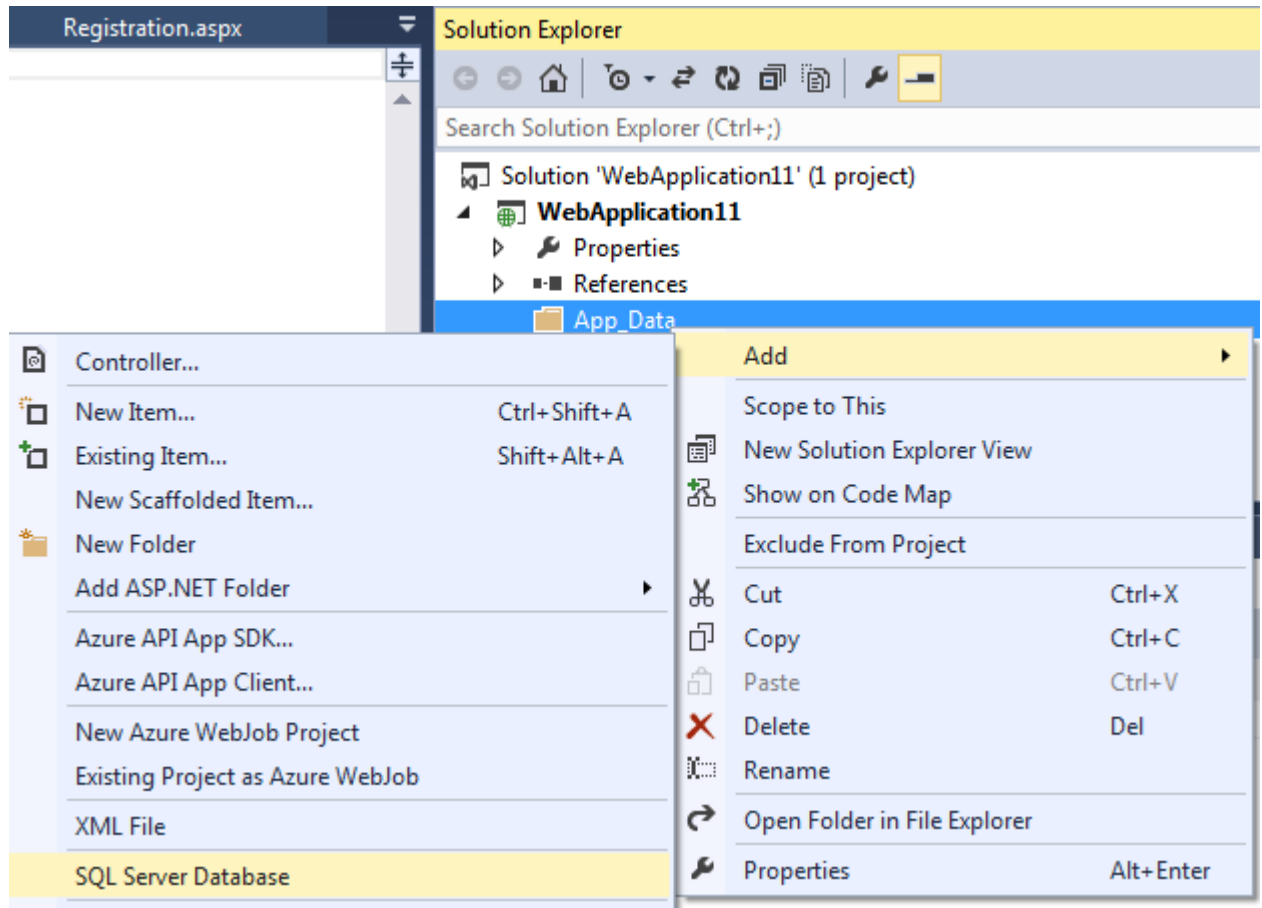
NRIC - tb\_nric

Mobile - tb\_mobile

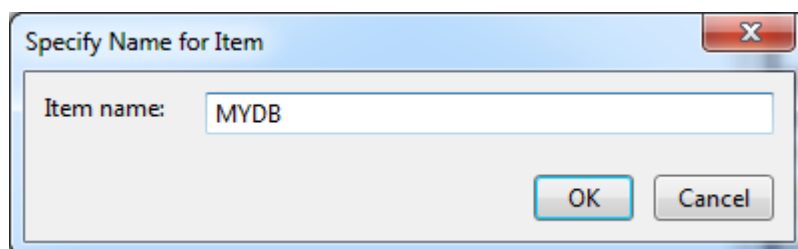
*Note :We will re-visit the input validation in later practical.*

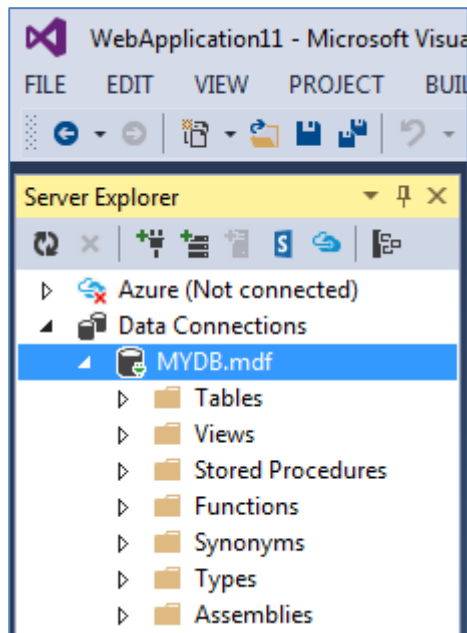
## Create a mdf file – MYDB.mdf

Right-click name of application → Choose Add → SQL Database Server.

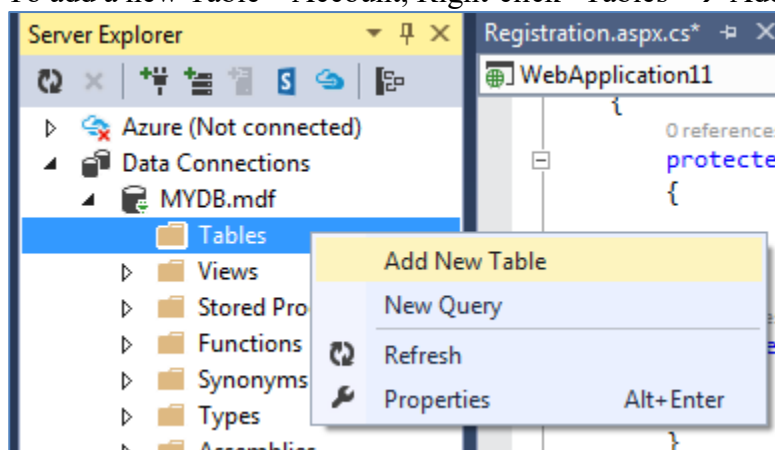


Choose a database name - MYDB

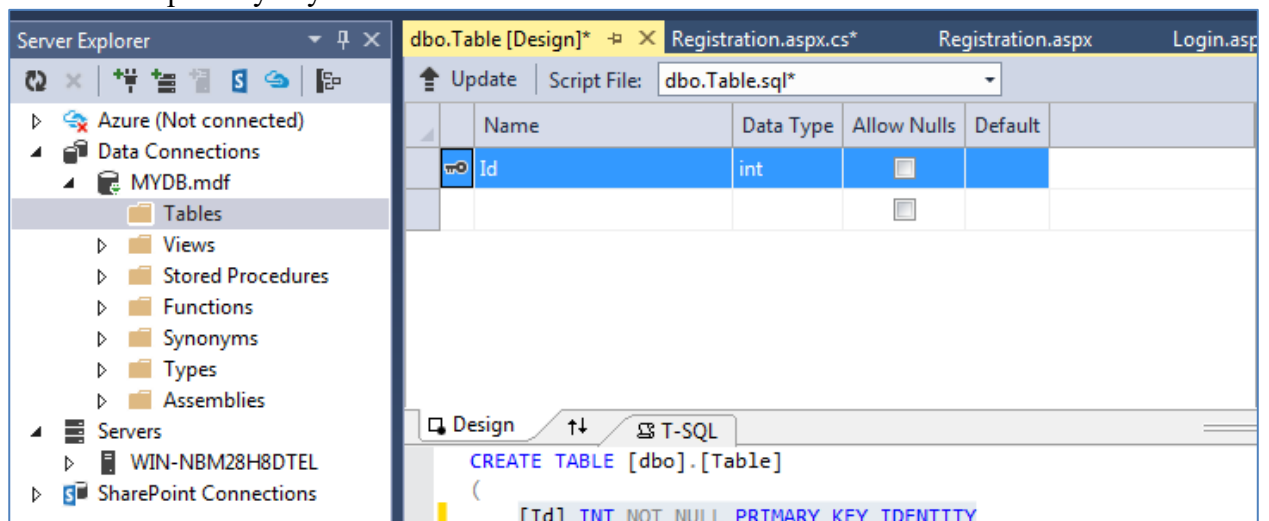




To add a new Table – Account, Right-click “Tables” → Add New Table

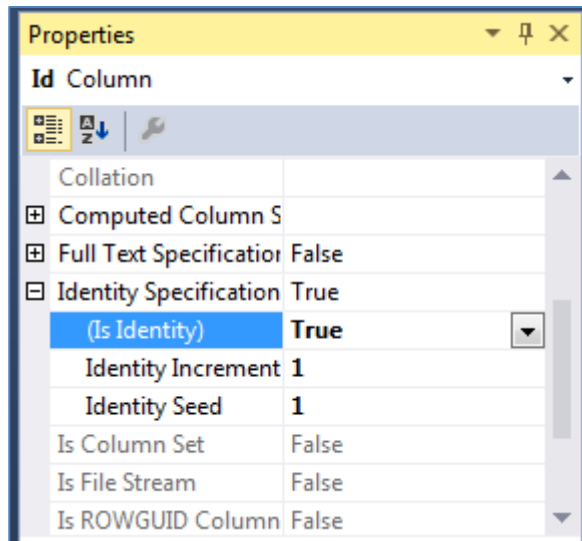


Set Id at the primary key with auto increment of 1.



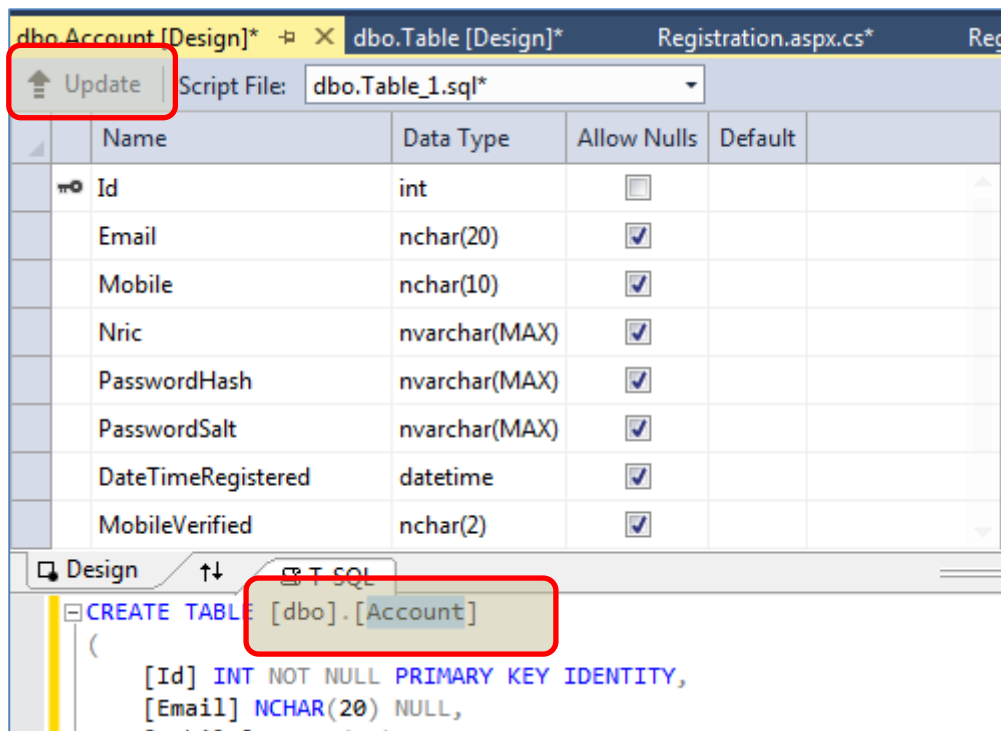


Set the Id to “primary key” with auto-increment in the properties page.

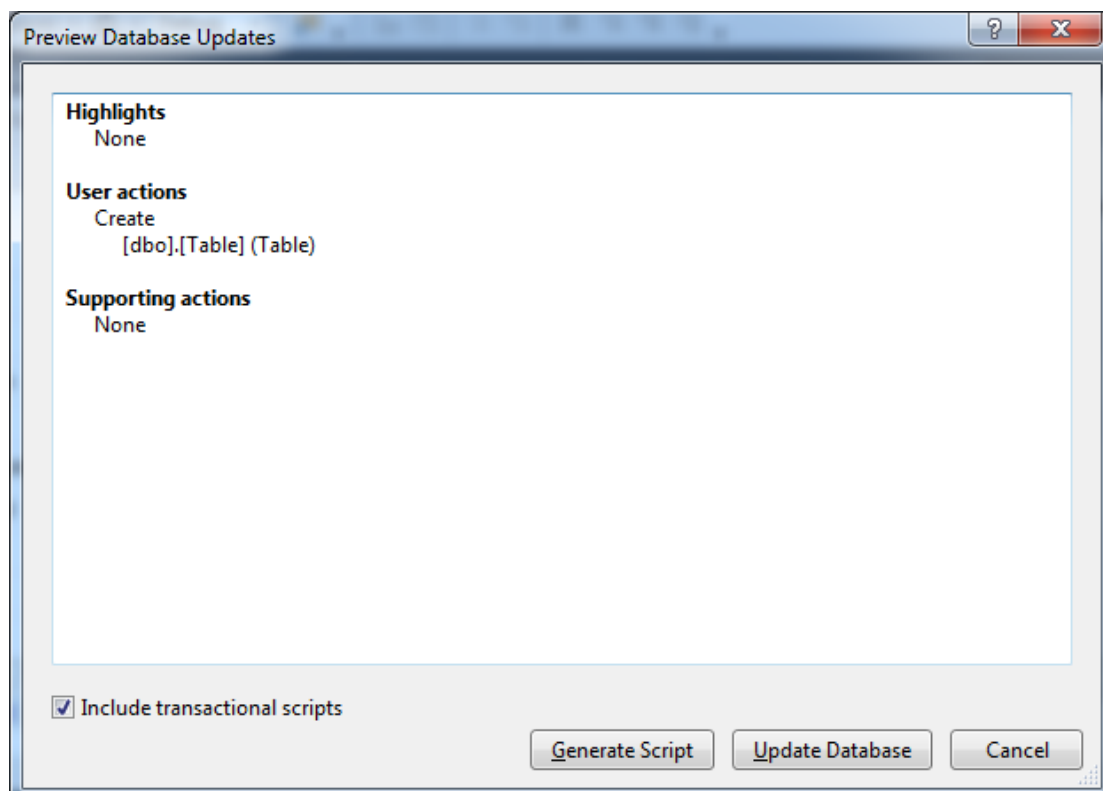


Continue to create the other columns.

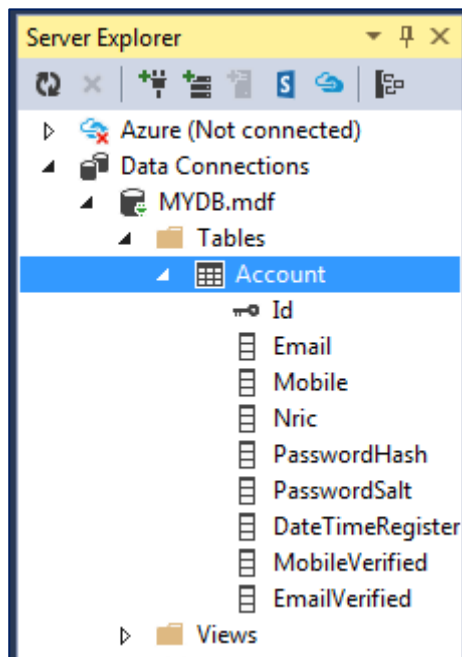
dbo.Table [Design]*					
Update   Script File: <input type="text" value="dbo.Table.sql*"/>					
	Name	Data Type	Allow Nulls	Default	
	Id	int	<input type="checkbox"/>		
	Email	nchar(20)	<input checked="" type="checkbox"/>		
	Mobile	nchar(10)	<input checked="" type="checkbox"/>		
	Nric	nvarchar(MAX)	<input checked="" type="checkbox"/>		
	PasswordHash	nvarchar(MAX)	<input checked="" type="checkbox"/>		
	PasswordSalt	nvarchar(MAX)	<input checked="" type="checkbox"/>		
	DateTimeRegistered	datetime	<input checked="" type="checkbox"/>		
	MobileVerified	nchar(2)	<input checked="" type="checkbox"/>		
	EmailVerified	nchar(2)	<input checked="" type="checkbox"/>		



Set the name of the table to “Account” and hit Update.



Hit “Update Database”. Check your newly created Account DB table.



## Registration.aspx - Create User Account to Database

### Hash Password with Salt

- Use the following codes to create a random Salt
- Combine (Concatenate) Password and Salt and
- Create a hash (Password + Salt) and save it into the database
- Save a copy of Salt in database

### Namespace required:

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

### Class Variables:

```
public partial class Registration : System.Web.UI.Page
{
    string MYDBConnectionString =
    System.Configuration.ConfigurationManager.ConnectionStrings["MYDBConnection"].ConnectionString;
    static string finalHash;
    static string salt;
    byte[] Key;
    byte[] IV;
```

Submit Button:

```
protected void btn_Submit_Click(object sender, EventArgs e) {
//string pwd = get value from your Textbox

    string pwd = tb_pwd.Text.ToString().Trim(); ;

    //Generate random "salt"
    RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
    byte[] saltByte = new byte[8];

    //Fills array of bytes with a cryptographically strong sequence of random values.
    rng.GetBytes(saltByte);
    salt = Convert.ToBase64String(saltByte);

    SHA512Managed hashing = new SHA512Managed();

    string pwdWithSalt = pwd + salt;
    byte[] plainHash = hashing.ComputeHash(Encoding.UTF8.GetBytes(pwd));
    byte[] hashWithSalt = hashing.ComputeHash(Encoding.UTF8.GetBytes(pwdWithSalt));

    finalHash = Convert.ToBase64String(hashWithSalt);

    RijndaelManaged cipher = new RijndaelManaged();
    cipher.GenerateKey();
    Key = cipher.Key;
    IV = cipher.IV;

    createAccount();
}
```

### CreateAccount Method: (Use Parameterized Query to prevent SQL Injection)

```
try
{
    using (SqlConnection con = new SqlConnection(MYDBConnectionString))
    {
        using (SqlCommand cmd = new SqlCommand("INSERT INTO Account VALUES(@Email,
@Mobile,@Nric,@PasswordHash,@PasswordSalt,@DateTimeRegistered,@MobileVerified,@EmailV
erified)"))
        {
            using (SqlDataAdapter sda = new SqlDataAdapter())
            {
                cmd.CommandType = CommandType.Text;
                cmd.Parameters.AddWithValue("@Email", tb_userid.Text.Trim());
                cmd.Parameters.AddWithValue("@Mobile", tb_mobile.Text.Trim());
                cmd.Parameters.AddWithValue("@Nric", encryptData(tb_nric.Text.Trim()));
                cmd.Parameters.AddWithValue("@PasswordHash", finalHash);
                cmd.Parameters.AddWithValue("@PasswordSalt", salt);
                cmd.Parameters.AddWithValue("@DateTimeRegistered", DateTime.Now);
                cmd.Parameters.AddWithValue("@MobileVerified", DBNull.Value);
                cmd.Parameters.AddWithValue("@EmailVerified", DBNull.Value);
                cmd.Connection = con;
                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
            }
        }
    }
}
catch (Exception ex)
{
    throw new Exception(ex.ToString());
}
```

### Database Connection String

Check web.config file for the db connectionstring

```
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
  </system.web>
  <connectionStrings>
    <add name="MYDBConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;
AttachDbFilename=|DataDirectory|.mdf;Initial Catalog=MYDB;Integrated Security=True"
providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

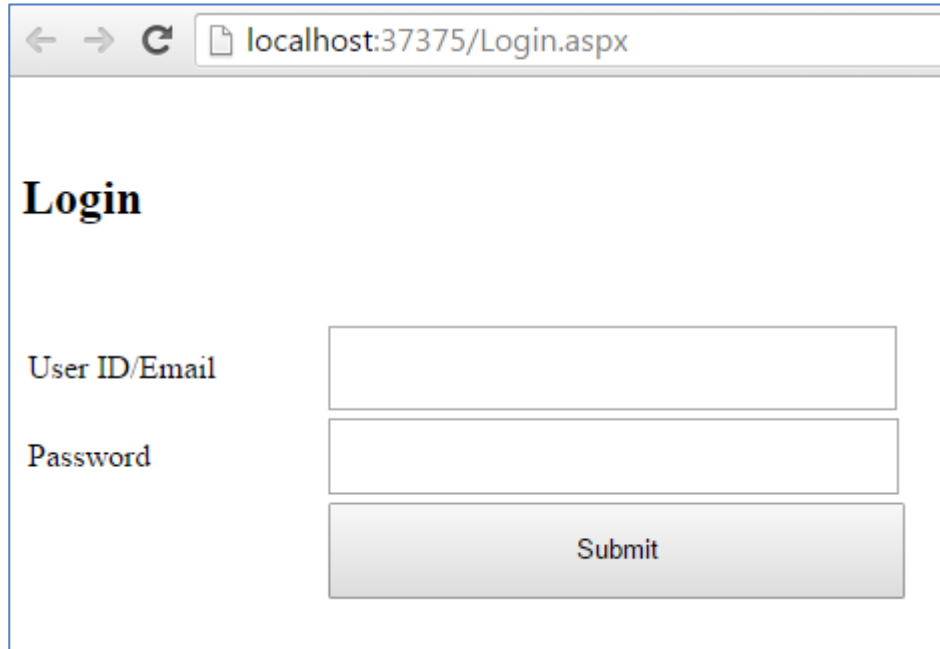
### Test Run – Registration.aspx

Proceed to create your first account. Verify in your database that the Password+Salt is hashed.

## Create Login.aspx

Add a new Web Form (c#) login.aspx

Create the UI shown below :



### Textboxes

Email – tb\_userid

Password – tb\_pwd

### Submit Button

```
string pwd = tb_pwd.Text.ToString().Trim();
string userid = tb_userid.Text.ToString().Trim();

SHA512Managed hashing = new SHA512Managed();
string dbHash = getDBHash(userid);
string dbSalt = getDBSalt(userid);

try
{
    if (dbSalt != null && dbSalt.Length > 0 && dbHash != null && dbHash.Length > 0)
    {
        string pwdWithSalt = pwd + dbSalt;
        byte[] hashWithSalt = hashing.ComputeHash(Encoding.UTF8.GetBytes(pwdWithSalt));
        string userHash = Convert.ToBase64String(hashWithSalt);

        if (userHash.Equals(dbHash))
        {
            Response.Redirect("Success.aspx", false);
        }
    }

    else {
```

```

        errorMsg = "Userid or password is not valid. Please try again.";
        Response.Redirect("Login.aspx");
    }
}
catch (Exception ex){
    throw new Exception(ex.ToString());
}

finally { }
}

```

#### getDBHash

```

protected string getDBHash(string userid) {

    string h = null;

    SqlConnection connection = new SqlConnection(MYDBConnectionString);
    string sql = "select PasswordHash FROM Account WHERE Email=@USERID";
    SqlCommand command = new SqlCommand(sql, connection);
    command.Parameters.AddWithValue("@USERID", userid);

    try
    {
        connection.Open();

        using (SqlDataReader reader = command.ExecuteReader())
        {
            while (reader.Read())
            {
                if (reader["PasswordHash"] != null)
                {
                    if (reader["PasswordHash"] != DBNull.Value)
                    {
                        h = reader["PasswordHash"].ToString();
                    }
                }
            }
        }
    }
    catch (Exception ex){
        throw new Exception(ex.ToString());
    }

    finally { connection.Close(); }
    return h;
}

```

getDBSalt

```
protected string getDBSalt(string userid)
{
    string s = null;

    SqlConnection connection = new SqlConnection(MYDBConnectionString);
    string sql = "select PASSWORDSALT FROM ACCOUNT WHERE Email=@USERID";
    SqlCommand command = new SqlCommand(sql, connection);
    command.Parameters.AddWithValue("@USERID", userid);

    try
    {
        connection.Open();

        using (SqlDataReader reader = command.ExecuteReader())
        {
            while (reader.Read())
            {
                if (reader["PASSWORDSALT"] != null)
                {
                    if (reader["PASSWORDSALT"] != DBNull.Value)
                    {
                        s = reader["PASSWORDSALT"].ToString();
                    }
                }
            }
        }

        catch (Exception ex)
        {
            throw new Exception(ex.ToString());
        }

        finally { connection.Close(); }
        return s;
    }
}
```



encryptData

```
protected byte [] encryptData(string data)
{
    byte[] cipherText = null;
    try
    {
        RijndaelManaged cipher = new RijndaelManaged();
        cipher.IV = IV;
        cipher.Key = Key;
        ICryptoTransform encryptTransform = cipher.CreateEncryptor();
        //ICryptoTransform decryptTransform = cipher.CreateDecryptor();
        byte[] plainText = Encoding.UTF8.GetBytes(data);
        cipherText = encryptTransform.TransformFinalBlock(plainText, 0,
plainText.Length);

    }
    catch (Exception ex)
    {
        throw new Exception(ex.ToString());
    }

    finally { }
    return cipherText;
}
```

Test run your login page.