Course: EGDF20
Module: EGE202 Application Programming

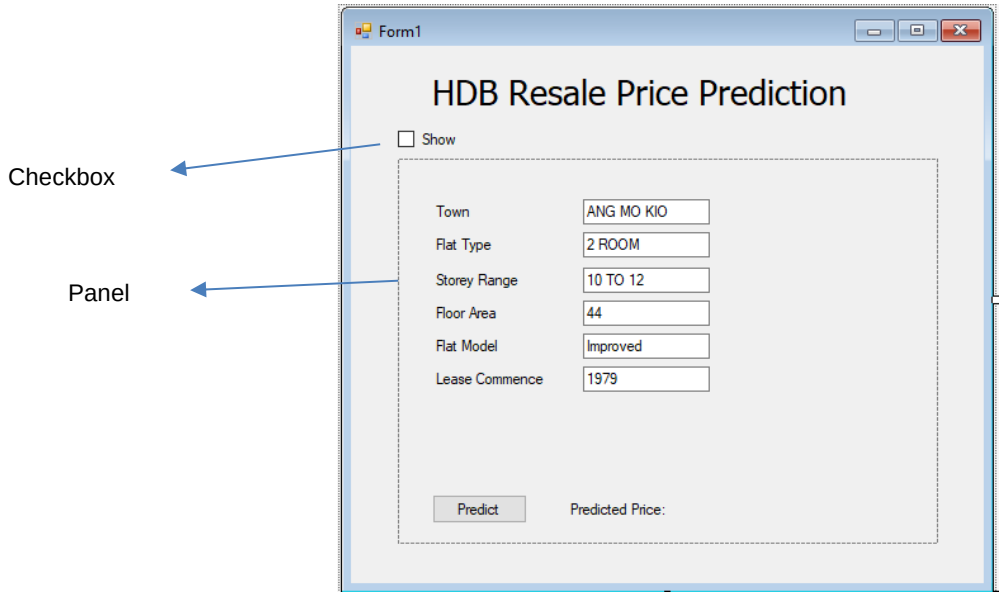
Practical 8: Integrating Web Resources using WebAPI

Objectives: At the end of this lab, the student is able to consume Web resources from WinForm application and also implement multiple Forms in their application and understand how to perform navigation between Forms. They will also experiment with container controls such as Panel.

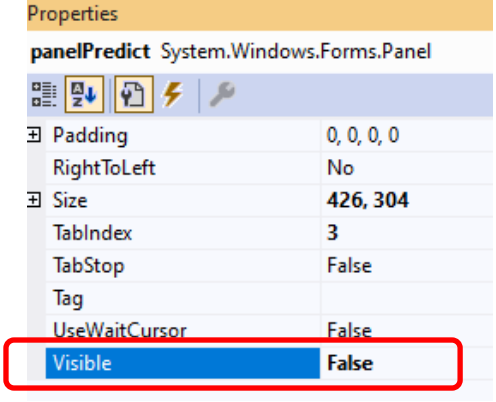
Exercise 1 – Consuming WebAPI

Part 1: Preparing UI for HDB Price Prediction

1. Download the pre-created **Visual Studio** solution **HDBPredict(Student).zip** and unzipped it to your own folder.
2. Under the ***File*** menu, click ***Open -> Project/Solution*** to open the project.



3. Build and run the application. Ensure that there are no errors.
4. The *Panel* control with name ***panelPredict*** contains *7 Labels*, *6 TextBox* and *1 Button*.
5. We can hide the Panel (along with all its child controls) by setting the ***Visibility*** to *False*.
6. Set the ***Panel*** visibility to *False* and test the application to ensure that it is hidden



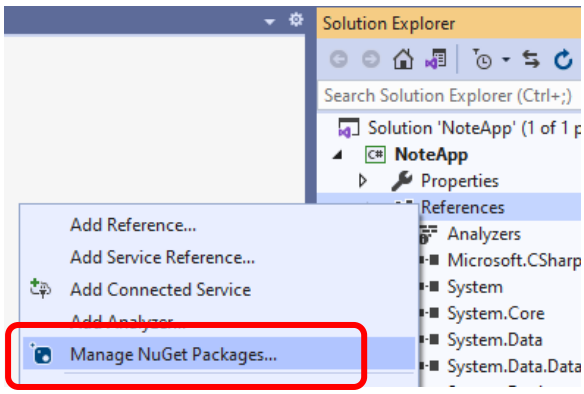
7. In the ***Form Designer*** double click on the ***checkbox (chkShow)*** to create *chkShow_CheckedChanged (…)* event handler and add the following codes.

```
private void chkShow_CheckedChanged (object sender, EventArgs e)
{
    if (chkShow.Checked)
        panelPredict.Visible = true;
    else
        panelPredict.Visible = false;
}
```

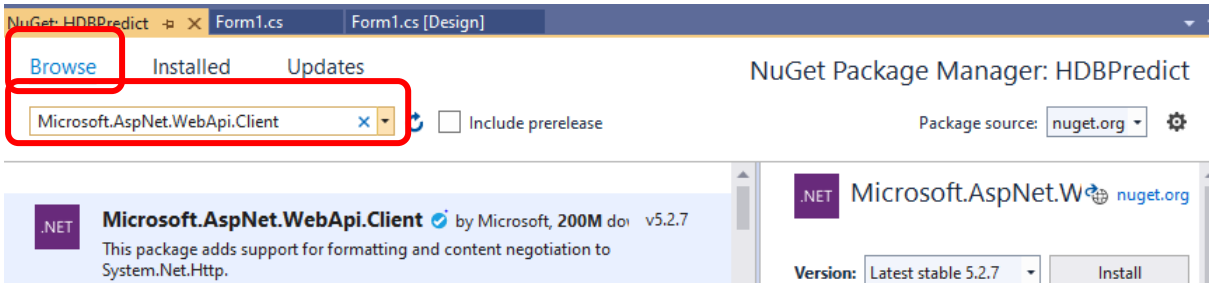
8. Build and test the application. Ensure that by checking the *CheckBox,* the *Panel* will appear and vice versa.

Part 2: Setting up Web API Client Library

1. At the ***Solution Explorer***, right click References and select ***Manage NuGet Packages***.



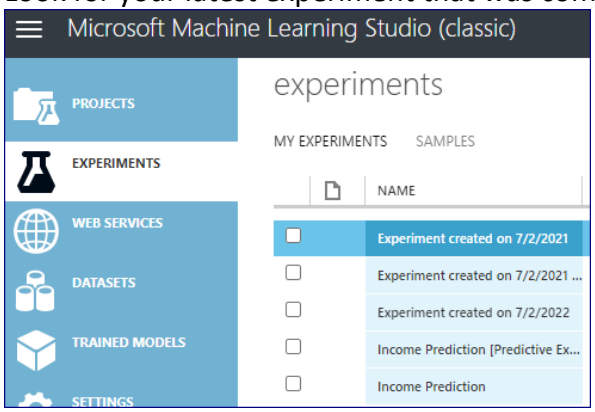
2. Then search for "***Microsoft.AspNet.WebApi.Client"*** in Browse tab and click install.



Part 3: Deploying Azure ML Experiment as Web Services

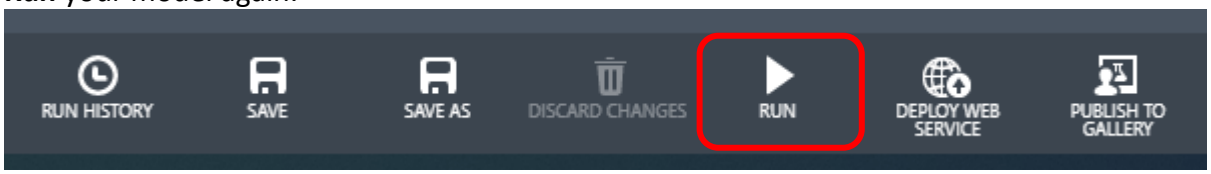
1. Login in to Azure ML Studio (Classic) if you need to login again. (https://studio.azureml.net/)

2. Look for your latest experiment that was completed in the SDL 4.

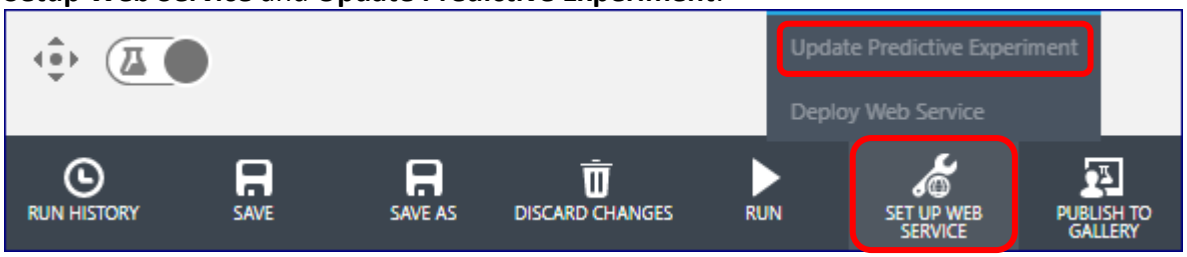


3. Double click on your experiment.

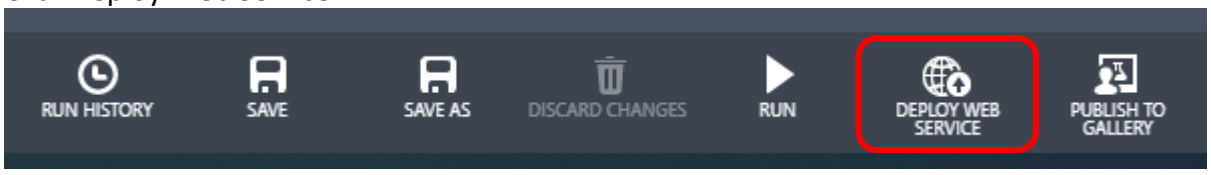
4. **Run** your model again.



5. **Setup Web Service** and **Update Predictive Experiment**.

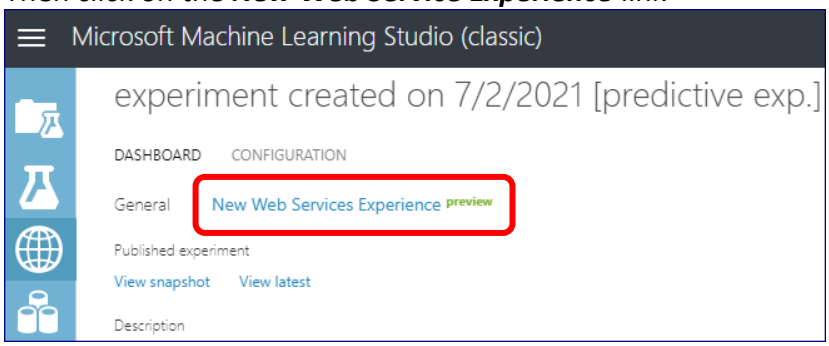


6. Click Deploy Web Service

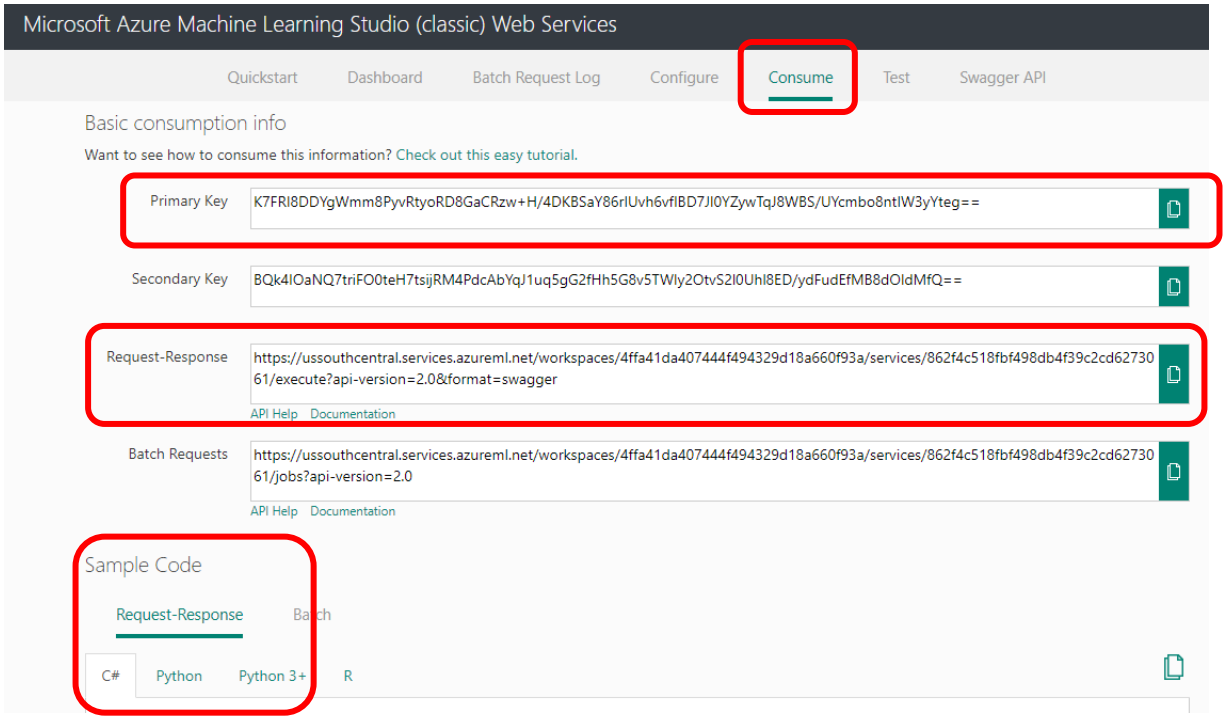


7. Once Web Service has been successfully deployed, click on the ***Go to Web Service*** link

8. Then click on the ***New Web Service Experience*** link



9. Then click on the Consume tab



10. Take note of ***Primary Key*** and ***Request-Response*** URL. You will need this information later.
11. Also take note that it generates samples codes in C#. This sample code will be the reference for performing inference on the ML model that we trained in the Azure ML Studio.

Part 4: Consuming Web API

1. Now double click on the "Predict" ***Button*** in the ***Form Designer***. That will automatically create *btnPredict_Click (…)* function.
2. Modify *btnPredict_Click (…)* to include the following codes:
   (The codes inside the Big Red Box can be copied from the file [Lab08-WebServiceCode.txt] in Brightspace.)

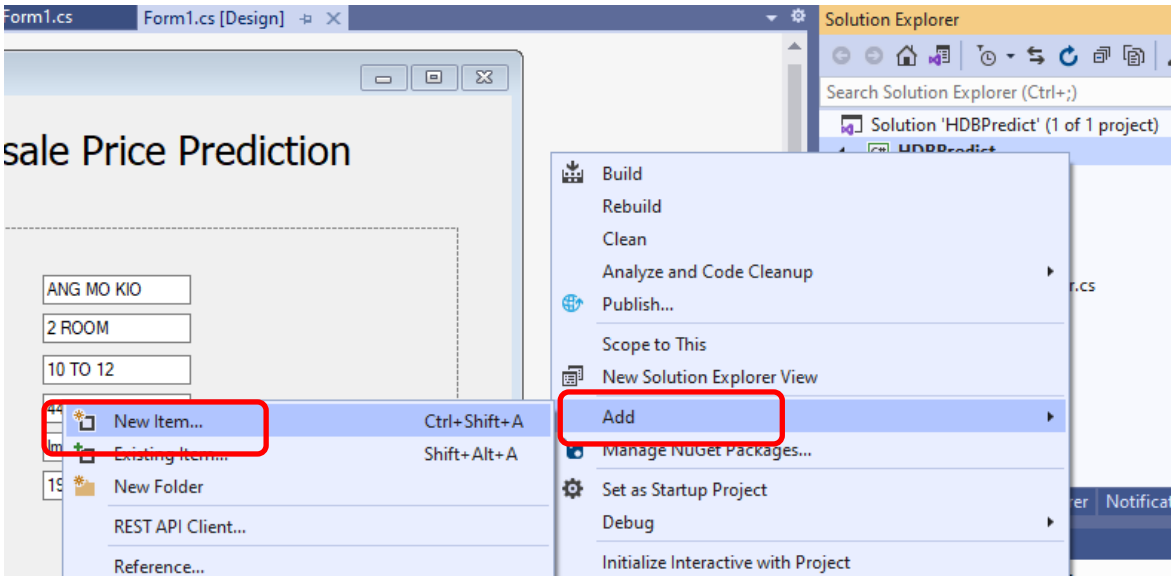
```
private async void btnPredict_Click(object sender, EventArgs e)
{
        using (var client = new HttpClient())
            {
                var scoreRequest = new
                {
                    Inputs = new Dictionary<string, List<Dictionary<string, string>>>()
                    {
                        {
                            "input1", new List<Dictionary<string, string>>()
                            {
                                new Dictionary<string, string>()
                                {
                                    {"town", txtTown.Text},
                                    {"flat_type", txtFlatType.Text},
                                    {"storey_range", txtStoreyRange.Text},
                                    {"floor_area_sqm", txtFloorArea.Text},
                                    {"flat_model", txtFlatModel.Text},
                                    {"lease_commence_date", txtLeaseCommence.Text},
                                    {"resale_price", "0"}
                                }
                            }
                        },
                    },
                    GlobalParameters = new Dictionary<string, string>()
                    { }
                };
                const string apiKey = "3E3ucu + 8PzfN1Cp4zgikx2Zg1jRrGgufHKhJL + QaFlSAyW6N /
JKAhwpS9fOvTMURezUoeOHnInv7OVe3ThTXJw ==="; // Replace this with the API key for the web service
                client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
apiKey);
                client.BaseAddress = new
Uri("https://ussouthcentral.services.azureml.net/workspaces/7ca67118937c451f8d59dbbcfe50386a/services/
cc52ecfefbe144579d84c8df8d58028a/execute?api-version=2.0&format=swagger");

                HttpResponseMessage response = await client.PostAsJsonAsync("", scoreRequest);
                if (response.IsSuccessStatusCode)
                {
                    string result = await response.Content.ReadAsStringAsync();
                    JObject data = (JObject)JsonConvert.DeserializeObject(result);
                    string strTemp = data["Results"]["output1"][0]["Scored Labels"].ToString();
                    lblPredict.Text = "Predicted Price: " + strTemp;
                }
                else
                {
                    lblPredict.Text = "Predicted Price: Error";
                }
            }
```

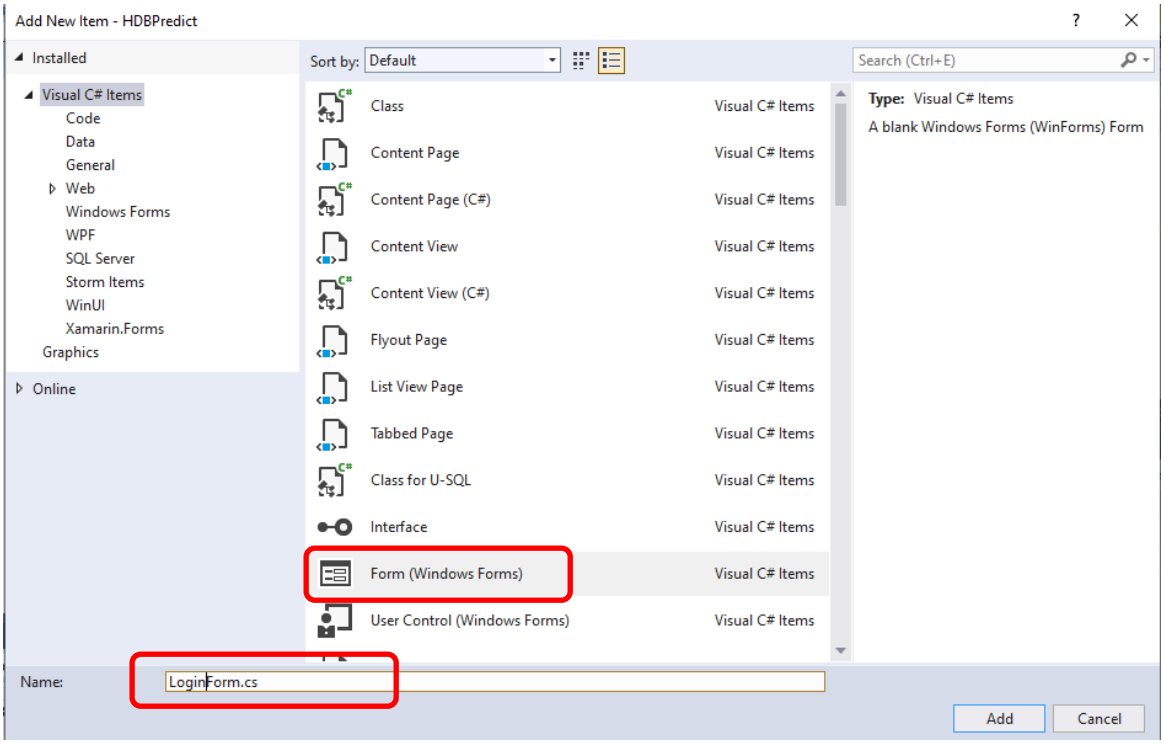
3. Make sure that the *apiKey* and *client.BaseAddress* matches the ***Primary Key*** and ***Request-Response*** URL from the ML studio (Part 3, step 6).
4. Build and test the application.

Part 2: Creating an additional Login Form

1. At the ***Solution Explorer*** right click ***HDBPredict*** and select ***Add->New Item***.

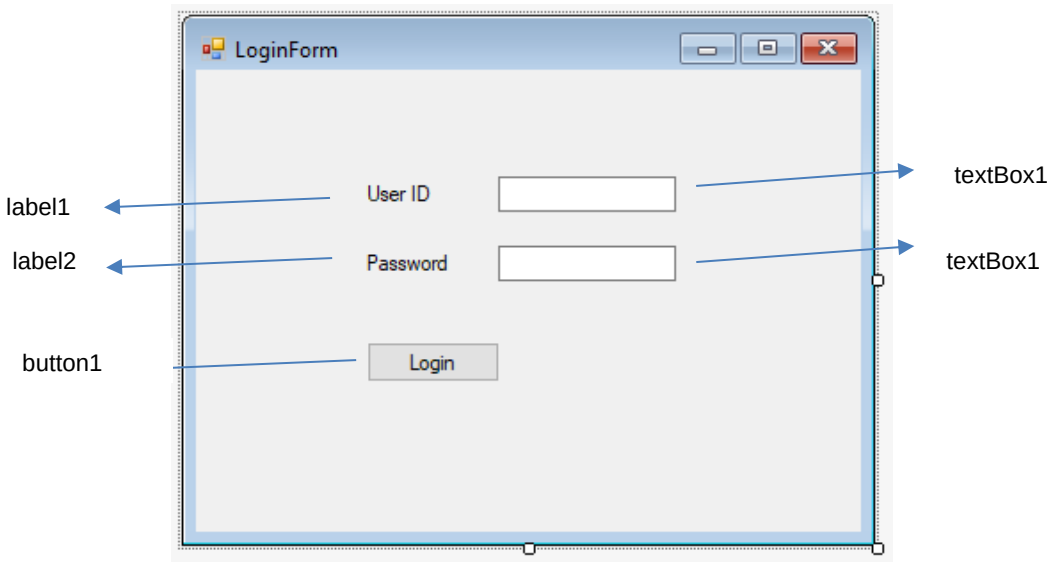


2. From the pop-up dialog, choose ***Form (Windows Form)*** and rename the ***Name*** to "***LoginForm.cs***".



3. A new ***LoginForm*** *Form* control should appear in the ***Form Designer***.
4. From the toolbox drag *2 Label, 2 TextBox and 1 Button* control to the *Form* area. Modify their properties based on the following.

{Name} From	{Name} To	{Text}
label1	lblUser	User ID
label2	lblPwd	Password
textBox1	txtUser	
textBox2	txtPwd	
button1	btnLogin	Login



5. Build and run the application. Which Form is being showed? *Form1 or LoginForm*?
6. At the ***Solution Explorer*** double click ***Program.cs***. Make the following changes:

```
static void Main()
 {
     Application.EnableVisualStyles();
     Application.SetCompatibleTextRenderingDefault(false);
     Application.Run(new LoginForm());
 }
```

7. Build and run the application. Which Form is being showed? *Form1 or LoginForm*?
8. Next add the following code declaration right at the start of *LoginForm* class.

```
public partial class LoginForm : Form
{
    Dictionary<string, string> id_pwd;

    public LoginForm()
    {
        InitializeComponent();
    }
```

9. In the ***Form Designer*** double click on the ***LoginForm Designer*** to create *LoginForm_Load(…)* event handler.

```
private void LoginForm_Load(object sender, EventArgs e)
{
        id_pwd = new Dictionary<string, string>();
```

```
            id_pwd.Add("admin", "p@ssw0rd");    // 1st accepted account
            id_pwd.Add("ege202", "");           // 2nd accepted account
        }
```

10. Now double click on the "Login" ***Button*** in the ***LoginForm Designer***. That will automatically create *btnLogin_Click (…)* function.
11. Modify *btnLogin_Click (…)* to include the following codes:

```
private void btnLogin_Click(object sender, EventArgs e)
{
    string id, pwd;

    id = txtUser.Text;
    pwd = txtPwd.Text;

    if (id_pwd.ContainsKey(id))
    {
        if (id_pwd[id] == pwd)
        {
            Form1 frm = new Form1();
            frm.Show();
        }
        else
            MessageBox.Show("Wrong User ID or Password");
    }
    else
        MessageBox.Show("Wrong User ID or Password");
}
```

12. Build and run your application. Try out the "user id" and "password" that we created on step 9.
13. Examine the codes implemented in step 11.

Actions	Observation / Action

Explain how these codes work.
 	
`if (id_pwd.ContainsKey(id))`
`{`
`  if (id_pwd[id] == pwd)`
`  {`
`  Form1 frm = new Form1();`
`  frm.Show();`
`  }`
`  else`
`  MessageBox.Show("Wrong User ID or Password");`
`}`	

Run the application. 1. **Enter the right user id and password to login. Did the HDBPredict Form appear? What about the LoginForm?**	

2. **At the HDBPredict form click the close button on the Form application title bar. After the HDBPredict Form closes, is the LoginForm still there?**

Modify the codes in step 11

```
if (id_pwd[id] == pwd)
{
    Form1 frm = new Form1();
    frm.Closed += (s, args) => this.Close();
    frm.Show();
}
```

What is your observation after the change?

Modify the codes in step 11

```
if (id_pwd[id] == pwd)
{
    this.Hide();
    Form1 frm = new Form1();
    frm.Closed += (s, args) => this.Close();
    frm.Show();
}
```

What is your observation after the change?

Note: The following is a method to create anonymous eventhandler (function or eventhandler without names)

Method 1:

```
frm.Closed += delegate (object s, EventArgs args)
                {
                     this.Close();
                };
```

Method 2: Using Lambda Expression

```
frm.Closed += (s, args) => this.Close();
```