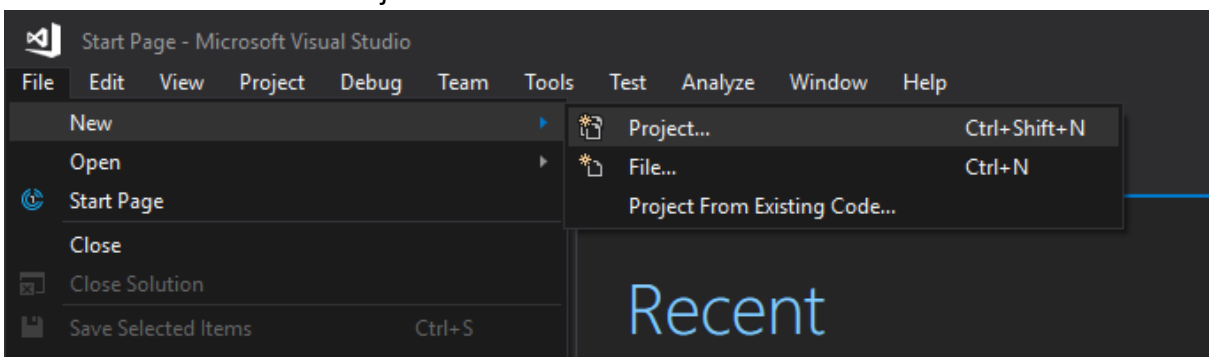
GrapeCity India Internship Workshop: Image Gallery Demo

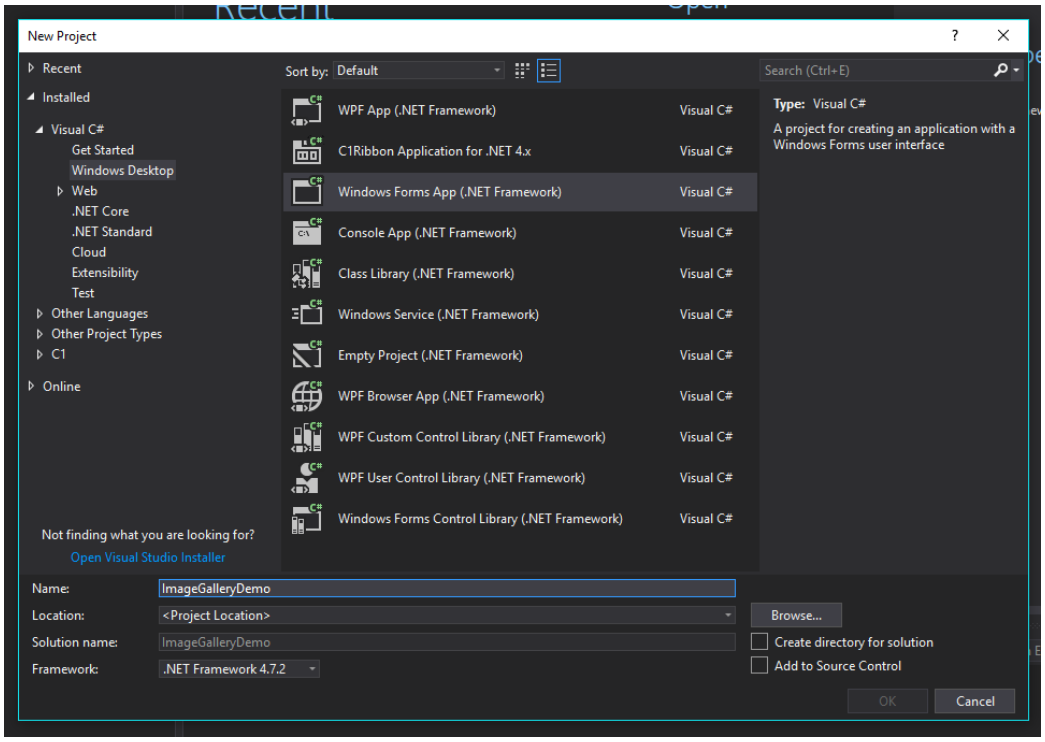
1. **Create a Project:**

   First of all, we need to create a project using Visual Studio 2017. To create a project, follow these steps:

   a) Open Visual Studio 2017.
   b) Go to File -> New -> New Project.



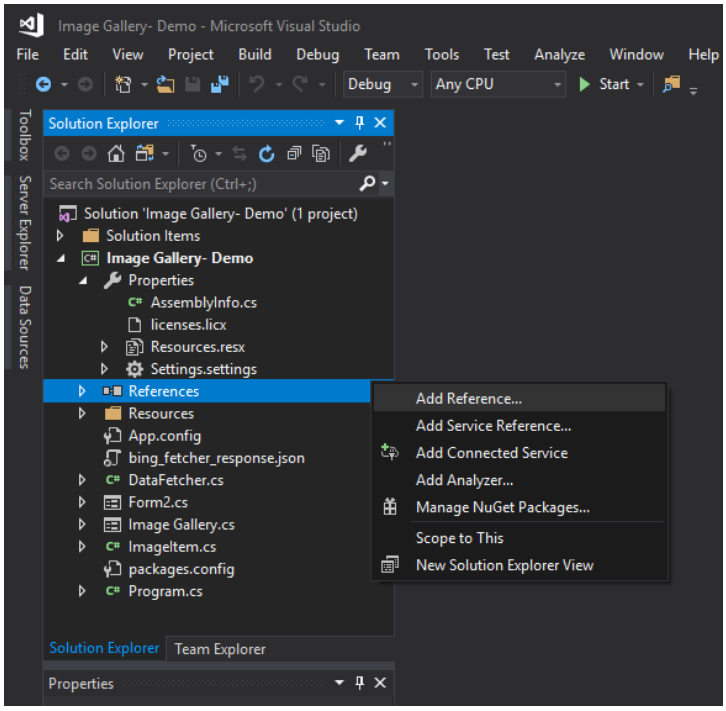
   c) In the New Project Dialog, expand Visual C# and select Windows Desktop.
   d) Select Windows Form App (.NET Framework).
   e) Provide an appropriate name for the project. Here we are calling it “Image Gallery Demo”.
   f) Specify an appropriate location.
   g) Click on OK.



2. **Add references to your solution:**

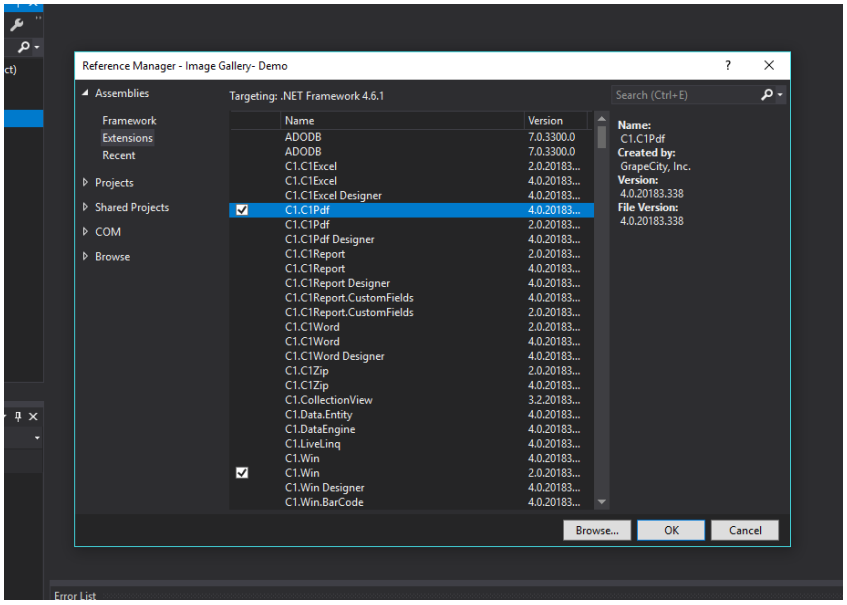
   Now we need to add references to the libraries that we are going to use in our projects. To add references to your project, follow these steps:

   a) Right click on References in the solution explorer.
   b) Click on Add Reference.



c) In the Add Reference dialog box, expand Assemblies.

d) Click on Extensions.

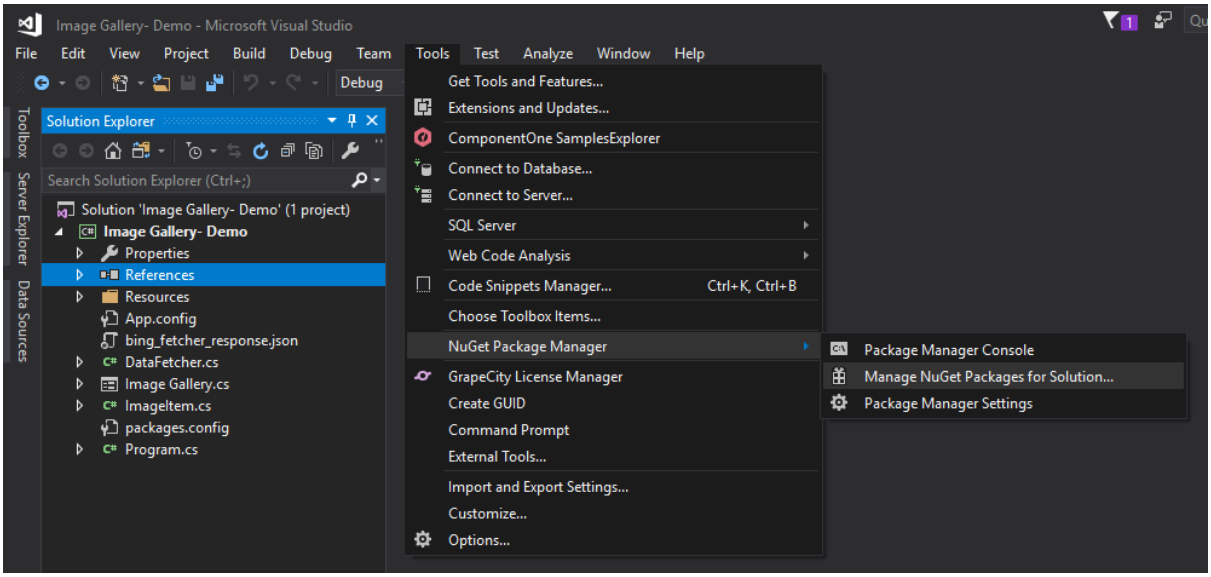


e) Add reference to these libraries by selecting the checkbox on the left:

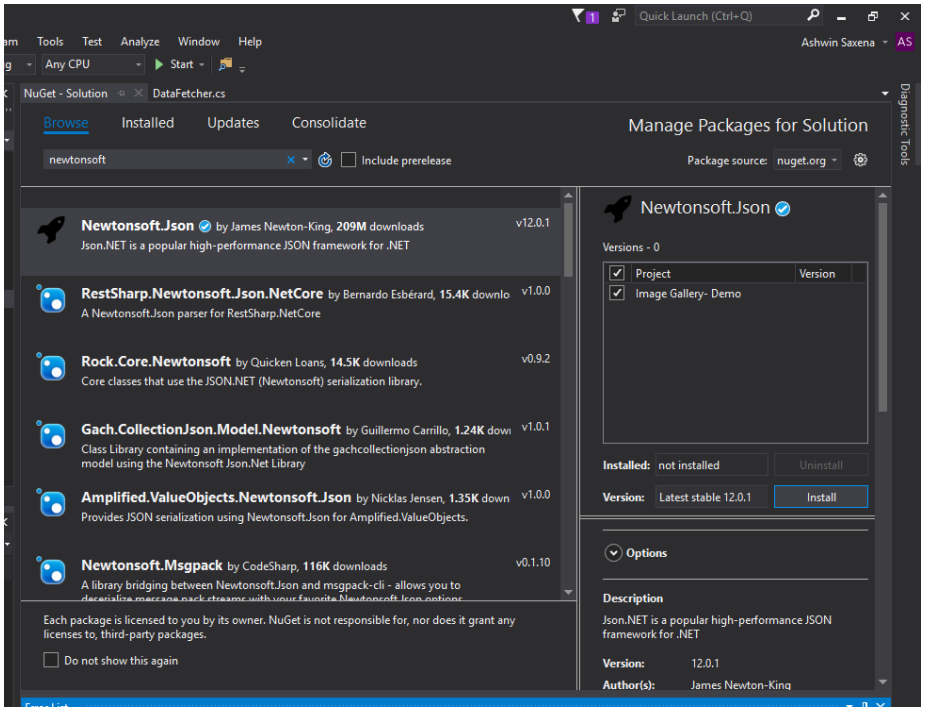
- C1.C1Pdf
- C1.C1Zip
- C1.Win
- C1.Win.C1DX
- C1.Win.C1TileControl

To parse the JSON data received from the server, we need to add a library called Newtonsoft.Json. To add reference to this library, follow these steps:

a) Go to Tools -> NuGet Package Manager -> Manage NuGet Packages for Solution.



b) A new window is opened. Go to Browse Tab and in the search bar of this tab, type Newtonsoft.Json and press Enter.

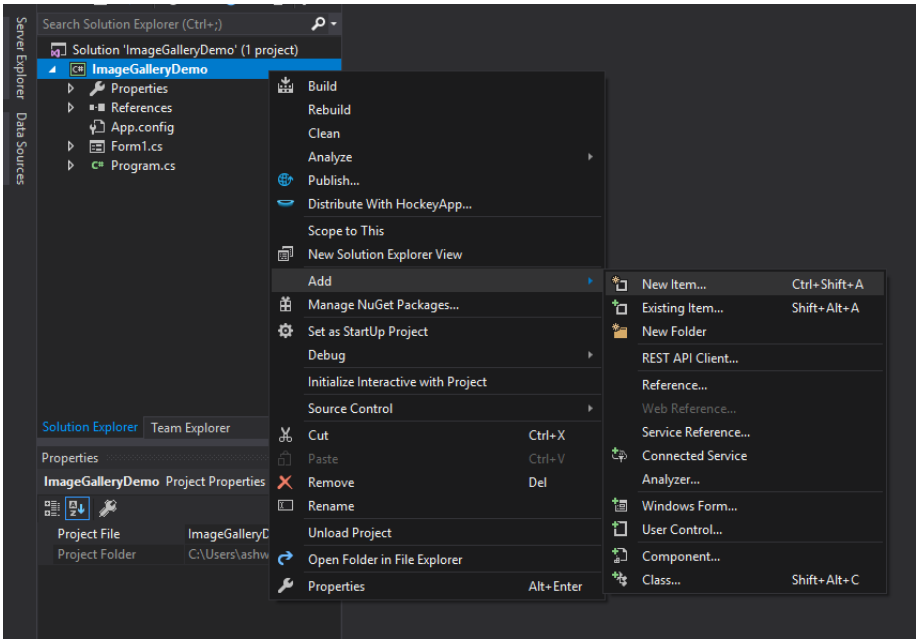


c) Click on the first result, a dialog box will open on the right.
d) Select the project Image Gallery Demo and click install.

3. **Create ImageItem class:**

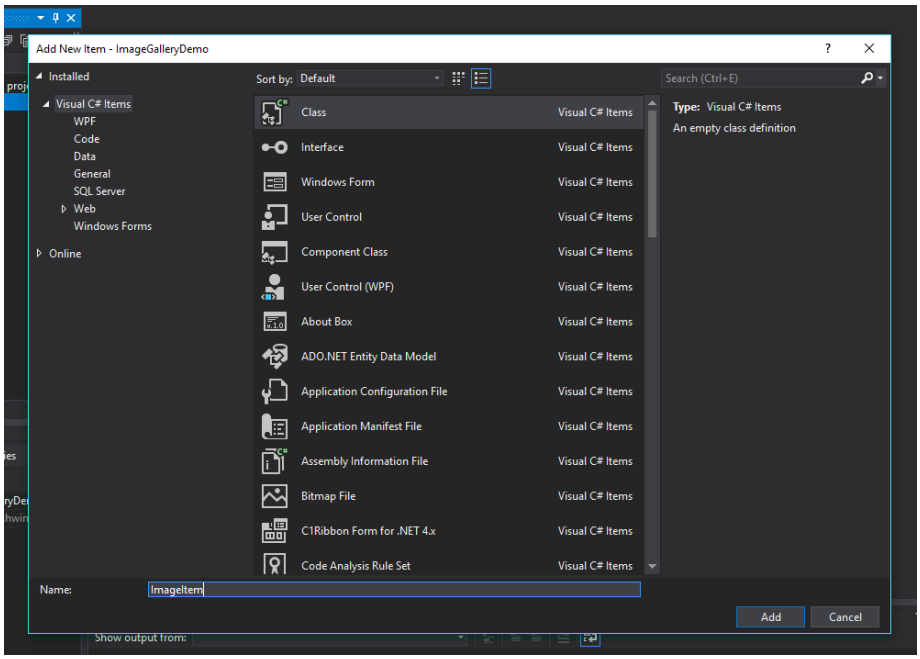
Now, we will create a class “ImageItem”. This class will help us to store the parsed JSON data. To create a class follow these steps:

a) In the Solution Explorer, right click on the project.
b) Select Add -> New Item. A dialog box will open up.



c) In the list of items, select Visual C# Class.

d) For the class name, type ImageItem and click Add.



e) Add this code inside the class:

```
public class ImageItem
{
    public string Id { get; set; }
    public string Name { get; set; }
    public byte[] Base64 { get; set; }
    public string Format { get; set; }
}
```

f) There are four properties in the class:

- Id – To set and get the id of the image.
- Name – To set and get the name of image.

- Base64 – To set and get the base64 URI of image.
- Format – To set and get the format of the image.

4. **Add DataFetcher class:**

We also need a class to fetch the data from server, parse it and provide the data back to the application. Follow the above procedure to add another class and name it DataFetcher.

First, we need to add the reference to the packages that we are going to use in this class. So, on the top of the file, add these 2 lines:

```
using Newtonsoft.Json;
using System.Net.Http;
```

System.Net.Http will be used to fetch the data from server and Newtonsoft.Json will be used to parse the JSON data which gets returned by the server.

a) Add GetDataFromService method using the code below:

```
async Task<string> GetDatafromService(string searchstring)
{
            string readText = null;
            try
            {
                var azure =
@"https://imagefetcher20200529182038.azurewebsites.net";
                string url = azure + @"/api/fetch_images?query=" +
searchstring + "&max_count=5";
                using (HttpClient c = new HttpClient())
                {
                    readText = await c.GetStringAsync(url);
                }
            }
            catch
            {
                readText =
File.ReadAllText(@"Data/sampleData.json");
            }
            return readText;
 }
```

This method uses the HttpClient class to fetch the JSON data from the server. The method GetStringAsync() is used to return response as a string in an asynchronous operation.

b) Add method to parse the JSON data returned.

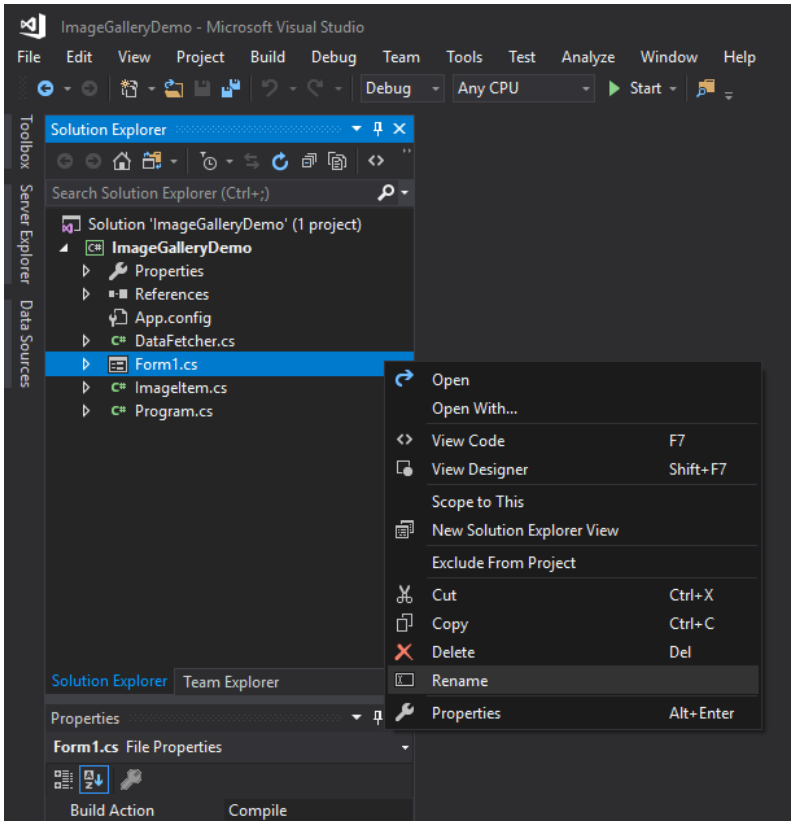
```
    public async Task<List<ImageItem>> GetImageData(string search)
    {
        string data =await GetDatafromService(search);
        return JsonConvert.DeserializeObject<List<ImageItem>>(data);
    }
```

Here, we use the JsonConvert.DeserializeObject() method of Newtonsoft.Json to parse the json data into an instance of ImageItem.

5. **Create the form to show the images:**

   Now, we need to add a windows form which will be used to show the images fetched by the server. This form will use the C1TileControl to show the images. Follow these steps to add a form to your solution:

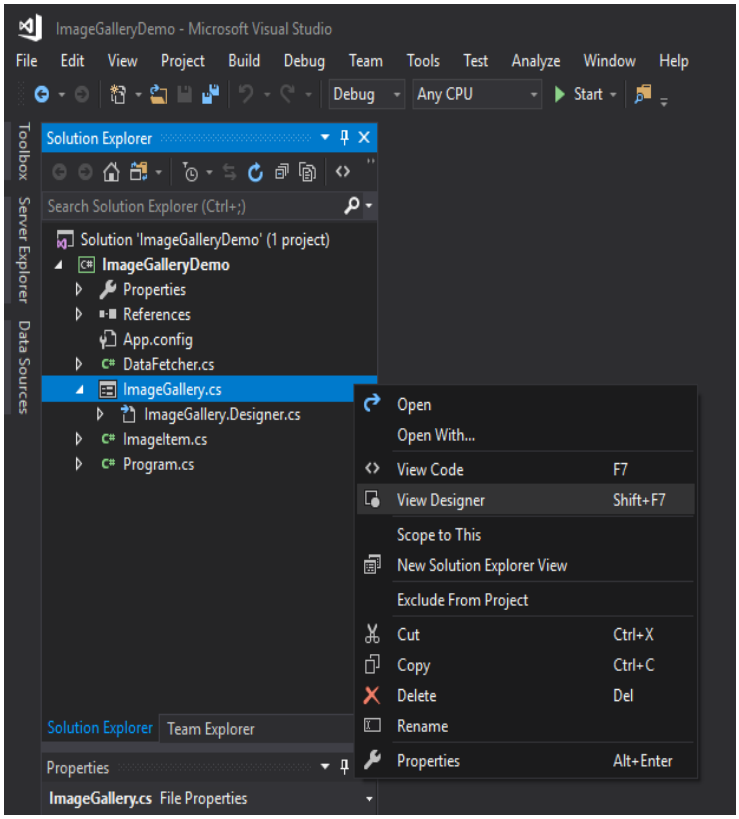
a) In the solution, you will see a Form1.cs file. This is the form which is already added by visual studio. Right click this form and click rename.



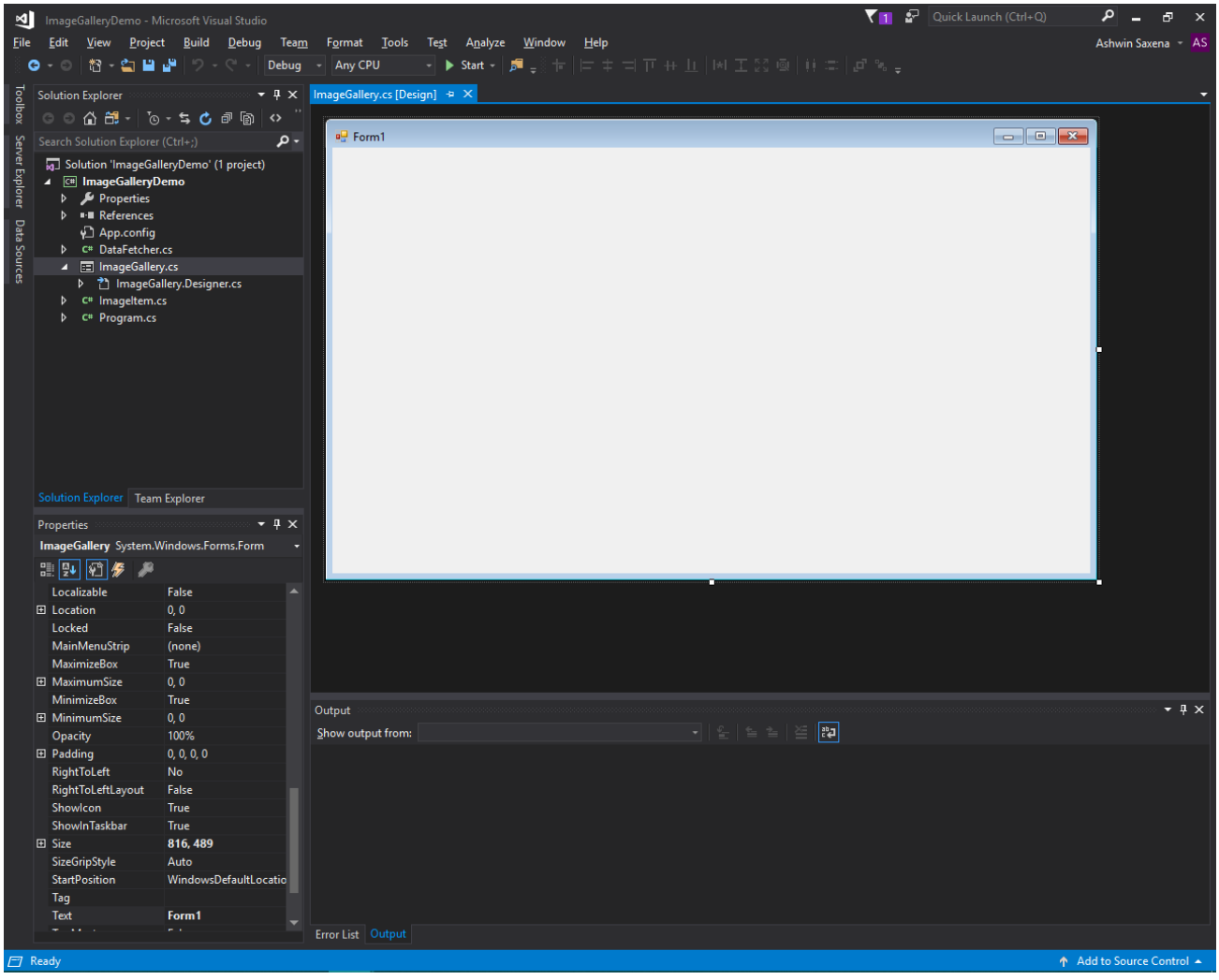
b) Name this form as ImageGallery.cs and click Yes on the dialog box that will open up.

Now, we need to add controls to this form. Follow these steps to add different controls to the form:

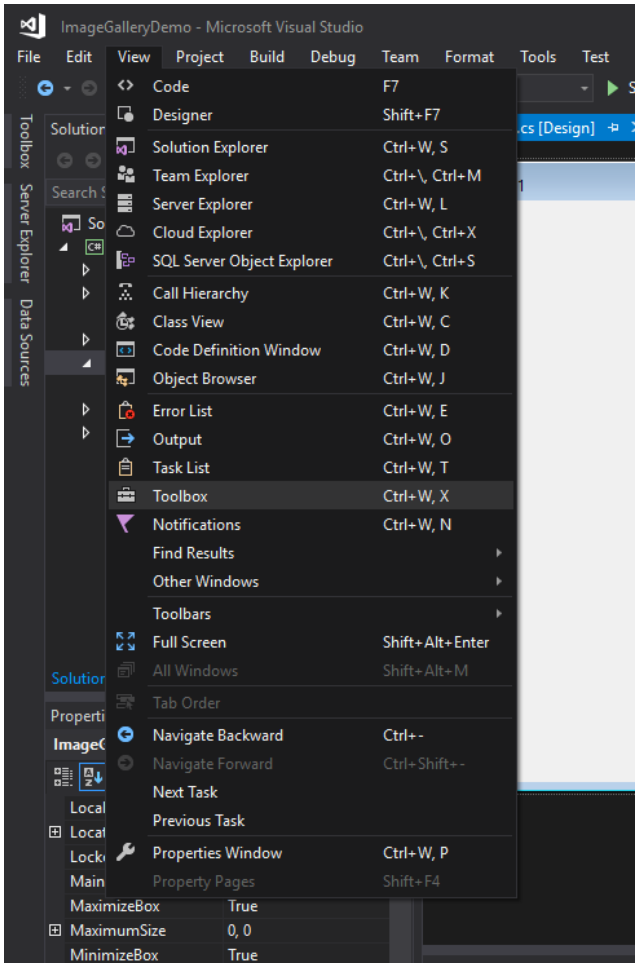
a) Right click on ImageGallery.cs and click on View Designer.



b) When you will click designer, a blank form will open up like this:

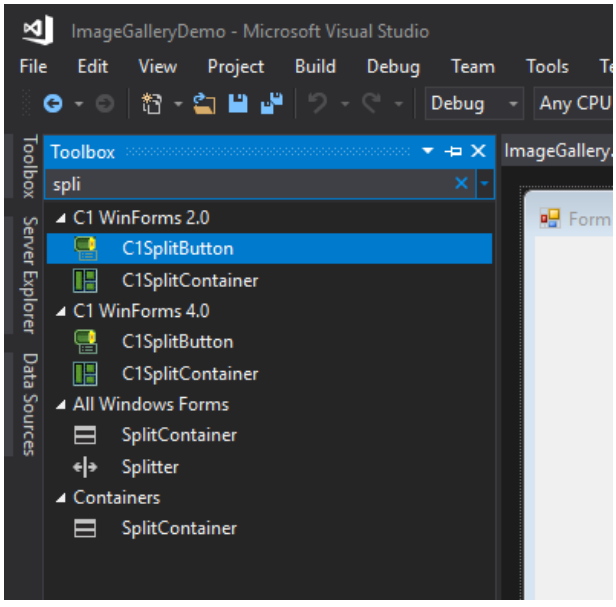


c) Now, go to View -> Toolbox. A list of controls will open on the right. This contains the list of all the controls that can be added to the form. To add a control, you just need to drag the control from toolbox to the position where you want to add the control.

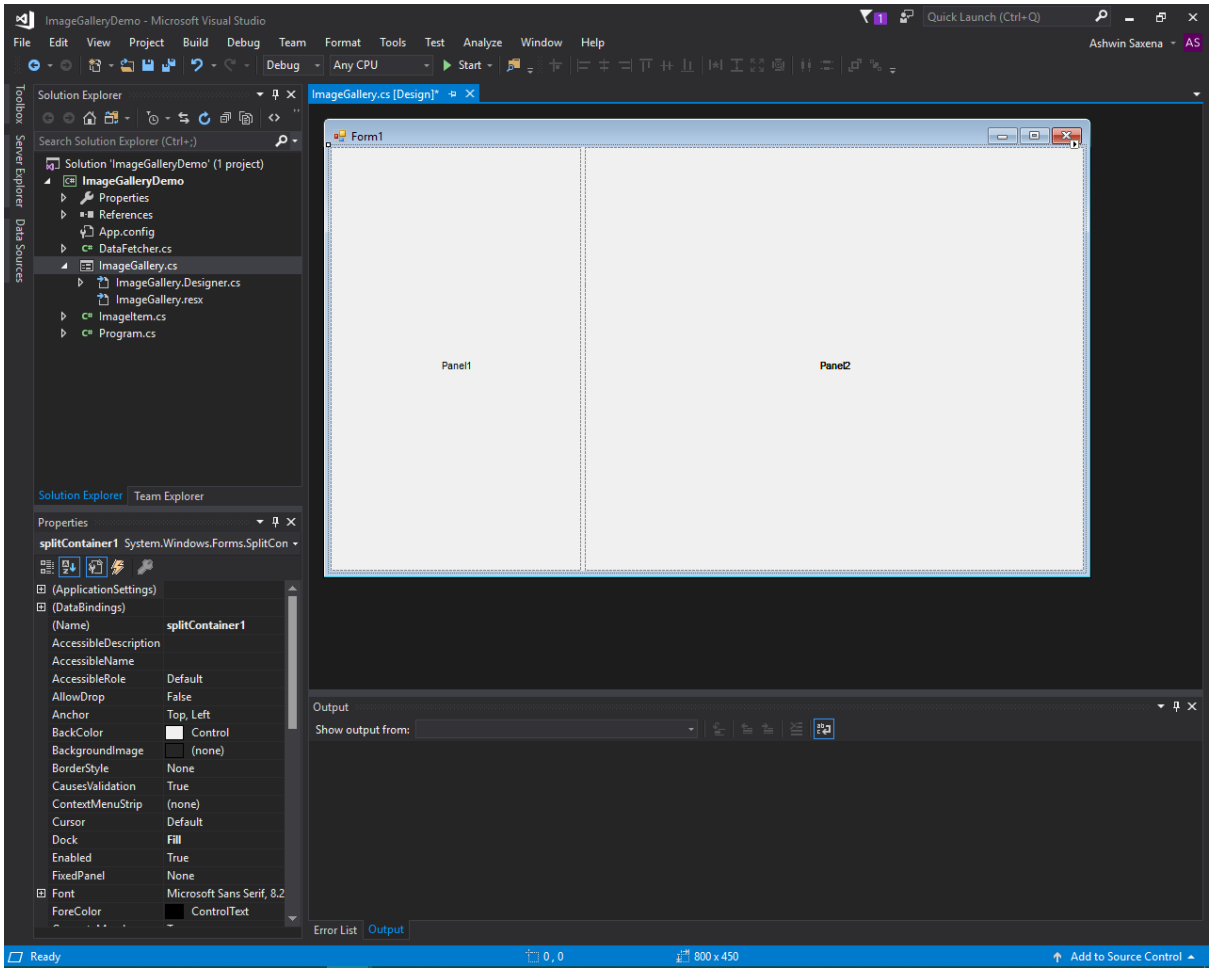


Add a split container control:

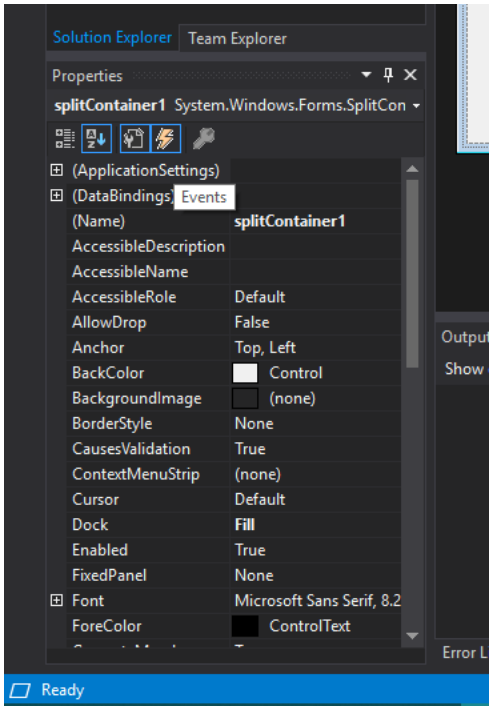
a) Search for SplitContainer in the list. Once you find it, drag it to the empty form.



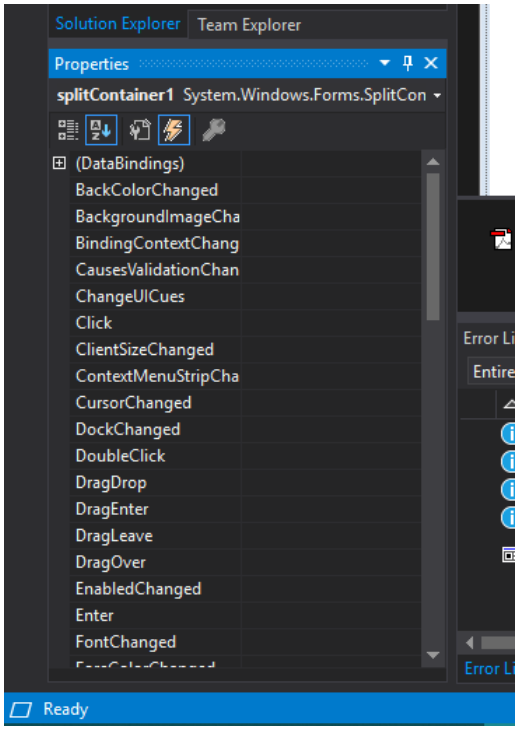
The form will look something like this:



b) Now, go to the properties window on the bottom left corner (you can also open properties window by right clicking on the control and clicking Properties). There will be a list of properties in the box.

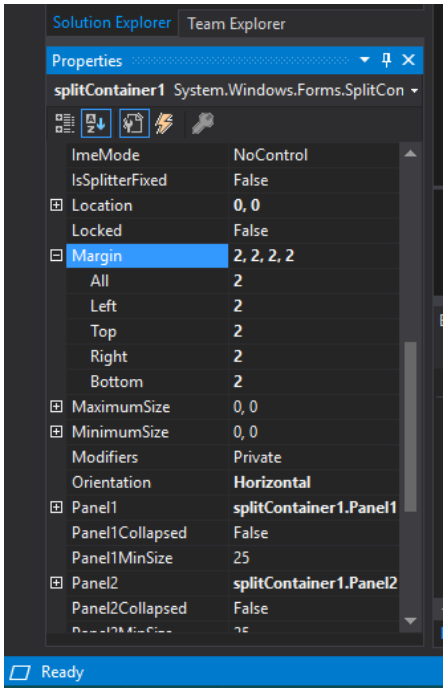


c) At the top of property window, just below the property title, there is a lightning bolt sign. If you click on it, a list of events will open. This will be used to add event handler to the control.



d) Now go back to the properties by clicking the settings icon just left of the events icon.

e) In the list of properties, go to the Dock property and set it to Fill. Now, go to the Margin property and click on the plus (+) icon on the left. Another list is shown which are the sub properties of a property. Now, in this list, set the All property to 2. This will give the split container a margin of 2 from all sides. Also set the orientation property of split container to Horizontal. Set the Splitter Distance property to 40. Set FixedPanel property as Panel1 and IsSplitterFixed as True.



Add form properties:

We also need to add or change some properties of the form. Select the ImageGalleryForm from the Combo Box present just above the lightning bolt sign.

MaximumSize: 810, 810
MaximizeBox: false
ShowIcon: false
Size: 780, 800

StartPosition: CenterParent
Text: “Image Gallery”

Add a table layout control:

a) Go back to View -> Toolbox and from the list of controls, drag the TableLayoutPanel to the Panel1 of split container. Now go to the properties window and apply these settings:

ColumnCount: 3

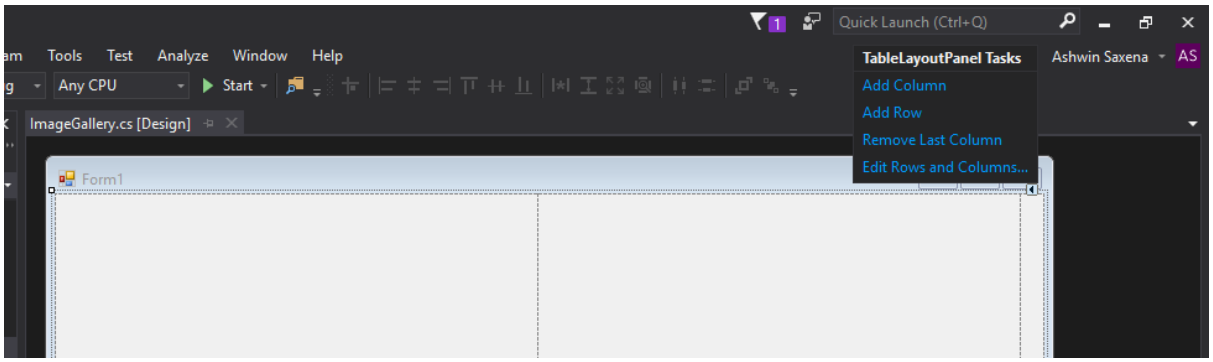
Dock: Fill

Location: 0, 0

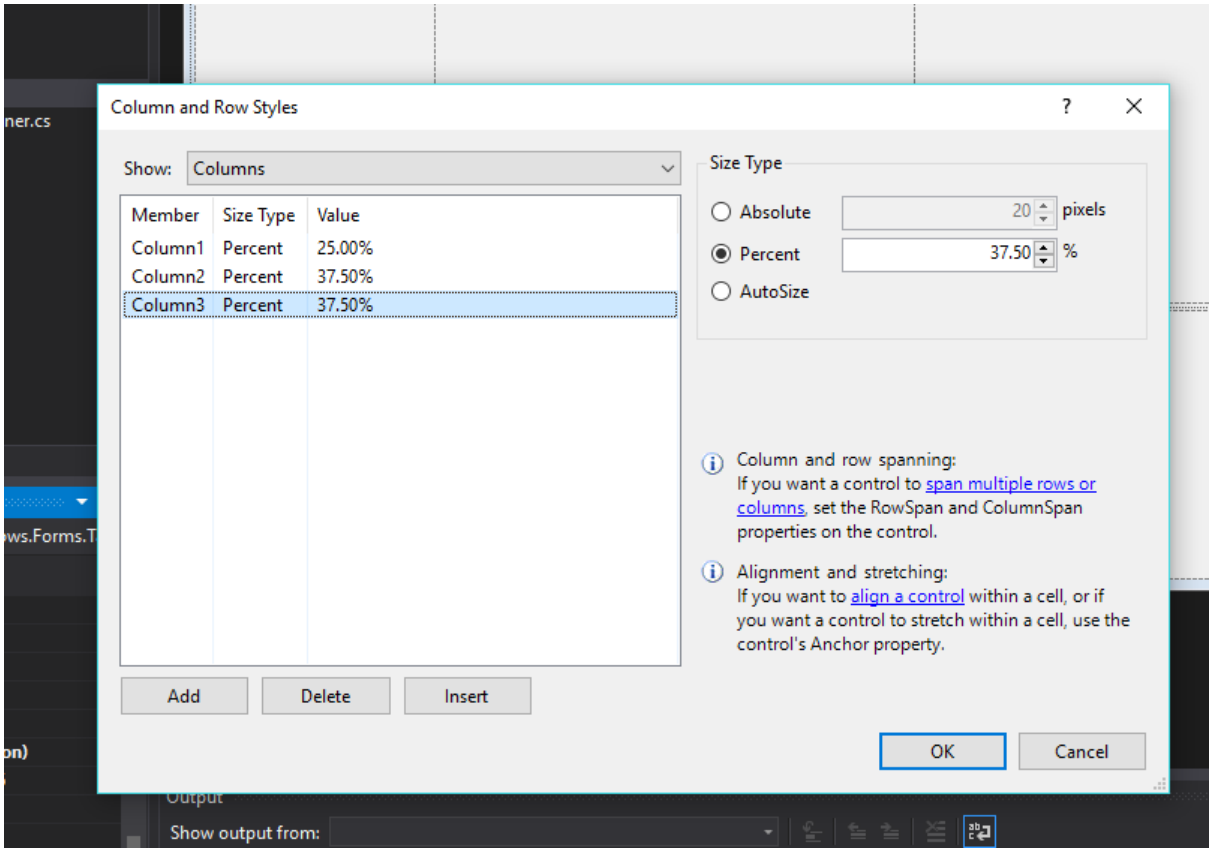
RowCount: 1

Size: 800, 40

b) Now, click on small arrow on the right corner of table layout panel and select Edit Rows and Columns. A dialog box will open up.



c) Select Column1 and set its Size Type on the right to Percent – 25%. For Column2 and Column3, set Size Type to a percentage of 37.50%.



Add a Panel:

a) Go to toolbox and add a Panel control to the 2nd column of Table Panel.
Go to properties and set the following properties:

Location:477,0
Size:287,40
Dock:Fill

b) Add Paint event handler to this control

We are going to add some code to this at a later stage. Now, go back to the ImageGallery designer.

Add a search box:

The search box will be used to find images based on a search key.

a) Go back to the toolbox and add a TextBox control to the panel in the second column of Table and go to properties and apply these properties:

Name: _searchBox
BorderStyle: None
Dock: Fill
Location: 16, 9
Size: 244, 16
Text: “Search Image”
Anchor:Top,Left,Bottom,Right

Add search button:

This will be used to search the images.

a) Add a panel control to the 3rd column of table panel and apply these properties:

Location: 479, 12
Margin: 2, 12, 45, 12
Size: 40, 16
TabIndex: 1

b) Add a picture box control to this panel and apply these properties:

Name: _search
Dock: Left
Location: 0, 0
Margin: 0, 0
Size: 40, 16
SizeMode: Zoom
Anchor:Top,Left,Bottom,Right

c) Add a click event by double clicking on the right box of Click in the event dialog box.
d) Add a search image to this control by double clicking on the text box of Image property of this control. A dialog box will open. Select the image you want to show by selecting Local Resource and click Import.

Add an export image button:

a) Add another picture box to the 2nd panel of split container. This will be used to export the images as PDF document. Then go to the properties window of this control and apply these properties:

Name: _exportImage
Location : 29, 3
Size: 135, 28
SizeMode: StretchImage
TabIndex: 2
Visible: false

b) We only want this export button to be visible only if an image is selected.
c) Also, add Click and Paint event handler for this control. You also need to add an image to this control.

Add a C1TileControl:

Now, we are going to add a C1TileControl. This will be used to show images on the form.

a) Go to toolbox and drag a C1TileControl to the 2nd panel just below the export image button and apply these properties:

Name: _imageTileControl

AllowRearranging: true
CellHeight: 78
CellSpacing: 11
CellWidth: 78
Dock: Fill
Size: 764, 718
SurfacePadding: 12, 4, 12, 4
SwipeDistance: 20
SwipeRearrangeDistance: 98

b) Now, we are going to add 3 event handlers to these events, TileChecked, TileUnchecked and Paint.

Add a C1PdfDocument control:

This control will be used to create the PDF document. Drag and drop a C1PdfDocument control to the form.

Add a StatusStrip control:

a) Go to toolbox, drag and drop StatusStrip control of WinForms to the Form

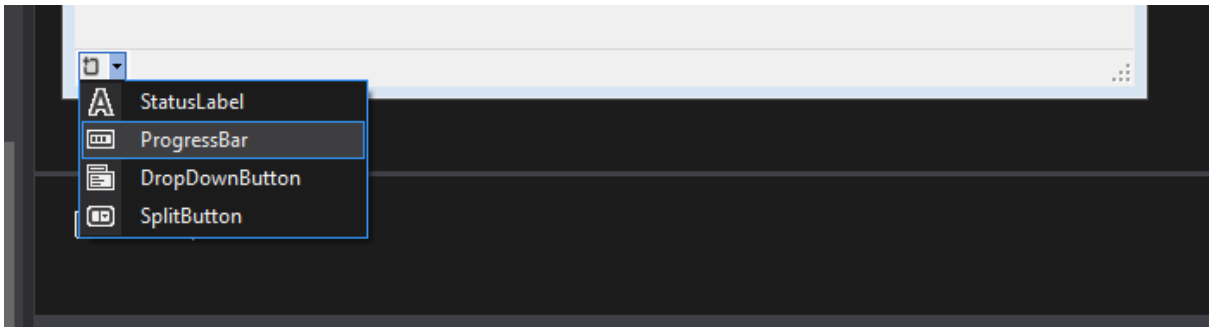
and apply the following properties:

Dock: Bottom

Visible : False

b) Now, click on the arrow present on right of the StatusStrip control and select ProgressBar and apply the following property to ProgressBar:

Style: Marquee

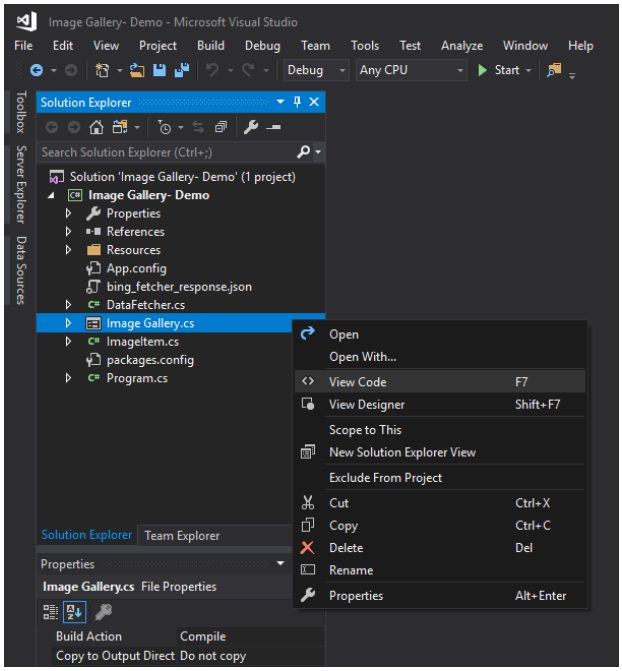


6. **Add methods to ImageGallery.cs:**

Now we need to add variables and methods to the form class.

Right click on ImageGallery.cs and

click View Code.



First, add references to these packages in the starting of the file:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Windows.Forms;
using C1.Win.C1Tile;
```

Now, in the class, add an instance of DataFechter class and a list of ImageItem class. Also add a class variable to keep a count of the checked tiles:

```
DataFetcher datafetch = new DataFetcher();
   List<ImageItem> imagesList;
   int checkedItems = 0;
```

Remember we added a click event to search. Now we will add a code to fetch the images using DataFetcher class. Add this code in the click event:

```
Private async void OnSearchClick(object sender, EventArgs e)
   {
       statusStrip1.Visible = true;
       imagesList = await
datafetch.GetImageData(_searchBox.Text);
       AddTiles(imagesList);
       statusStrip1.Visible = false;
   }
```

Now, we are going the add AddTiles method will loop through all the images and add it to the tile control:

```
Private void AddTiles(List<ImageItem> imageList)
   {
       _imageTileControl.Groups[0].Tiles.Clear();

       foreach (var imageitem in imageList)
       {
           Tile tile = new Tile();
           tile.HorizontalSize = 2;
           tile.VerticalSize = 2;

           _imageTileControl.Groups[0].Tiles.Add(tile);

           Image img = Image.FromStream(new
MemoryStream(imageitem.Base64));

           Template tl = new Template();
           ImageElement ie = new ImageElement();
           ie.ImageLayout = ForeImageLayout.Stretch;
           tl.Elements.Add(ie);
           tile.Template = tl;
           tile.Image = img;

       }

   }
```

This method takes a list of images, iterates through each image and add tile for each image. It converts the base64 encoding to the corresponding image using MemoryStream class.

We also need to keep a counter of how many tiles are selected. This counter will be used to show or hide the export button. So, in TileChecked event, we will increment the counter and in the TileUnchecked event, we will decrement it. Also, if the counter is zero, hide the export button:

```
private void OnTileChecked(object sender, TileEventArgs e)
   {
       checkedItems++;
       _exportImage.Visible = true;
   }

   private void OnTileUnchecked(object sender, TileEventArgs e)
   {
       checkedItems--;
       _exportImage.Visible = checkedItems > 0;
   }
```

Now, we will add a callback for click event for export button:

```
private void OnExportClick(object sender, EventArgs e)
    {
        List<Image> images = new List<Image>();
        foreach (Tile tile in _imageTileControl.Groups[0].Tiles)
        {
            if(tile.Checked)
            {
                images.Add(tile.Image);
            }
        }
        ConvertToPdf(images);
        SaveFileDialog saveFile = new SaveFileDialog();
        saveFile.DefaultExt = "pdf";
        saveFile.Filter = "PDF files (*.pdf)|*.pdf*";

        if (saveFile.ShowDialog() == DialogResult.OK)
        {

            imagePdfDocument.Save(saveFile.FileName);

        }
     }
```

This method iterates through all the tiles, gets its image and convert this list of images to pdf using ConvertToPdf method, which we are going to write in a moment. After converting to PDF, it opens a SaveFileDialog to save the image. Now, we will write ConvertToPdf method:

```
private void ConvertToPdf(List<Image> images)
   {
        RectangleF rect = imagePdfDocument.PageRectangle;
        bool firstPage = true;
        foreach (var selectedimg in images)
        {
               if (!firstPage)
                {
                    imagePdfDocument.NewPage();
                }
                firstPage = false;

                rect.Inflate(-72, -72);
                imagePdfDocument.DrawImage(selectedimg, rect);

        }

    }
```

This method creates a page for each image and draws the image using DrawImage method.

Now, we will add a callback method for paint event for different controls that we added before. These callbacks will be used to add other customizations like creating a grey border or drawing a line:

```
      private void OnSearchPanelPaint(object sender, PaintEventArgs
e)
        {
            Rectangle r = _searchBox.Bounds;
            r.Inflate(3, 3);
            Pen p = new Pen(Color.LightGray);
            e.Graphics.DrawRectangle(p, r);

        }


private void OnExportImagePaint(object sender, PaintEventArgs e)
        {
            Rectangle r = new Rectangle(_exportImage.Location.X,
_exportImage.Location.Y, _exportImage.Width, _exportImage.Height)
            r.X -= 29;
            r.Y -= 3;
            r.Width--;
            r.Height--;
            Pen p = new Pen(Color.LightGray);
            e.Graphics.DrawRectangle(p, r);
            e.Graphics.DrawLine(p, new Point(0, 43), new
Point(this.Width, 43));
        }


 private void OnTileControlPaint(object sender, PaintEventArgs e)
        {
            Pen p = new Pen(Color.LightGray);
            e.Graphics.DrawLine(p, 0, 43, 800, 43);
        }
```

The OnSearchPanelPaint callback is used to add a grey border to the search box, OnExportImagePaint is used for drawing a grey border for export to pdf button and OnTileControlPaint is used to draw a separator.

Now, we just need to run the application. To run the application, press F5 key or go to Debug -> Start Without Debugging. Once the application is running, click on the search image and a list of images will open up.

You can select the images by right clicking them and then you can export it by clicking the export button.

The application will look something like this:

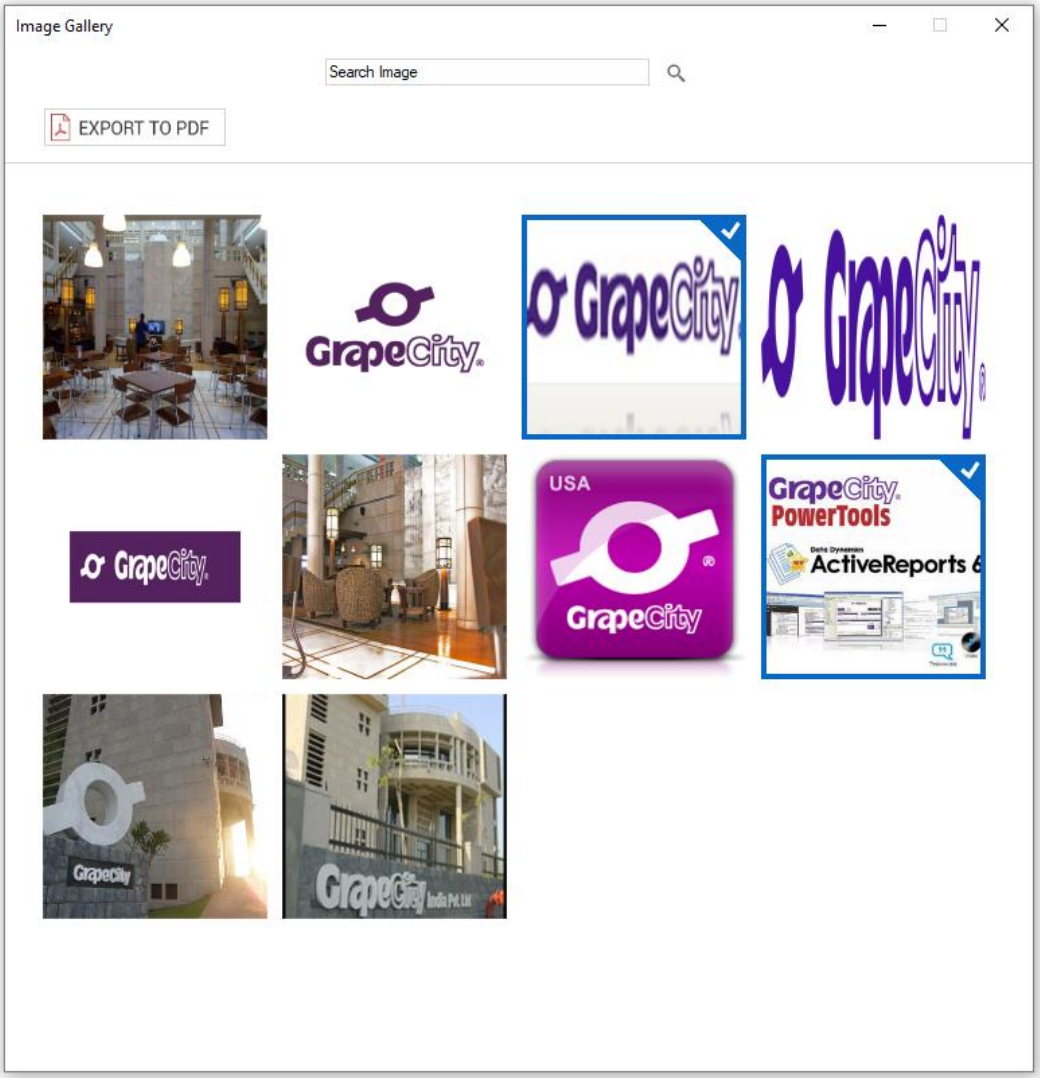


Image Gallery
Search Image
EXPORT TO PDF
GrapeCity
USA
GrapeCity
GrapeCity
PowerTools
ActiveReports