IRA A. FULTON SCHOOLS OF engineering

school of computing, informatics, & decision systems engineering

*CSE 446/598*
*Software Integration and Engineering*

# Unit 1
# Service Standards and Service Development

# Lecture 1-5
# RESTful Services Development and Case Studies

Dr. Yinong Chen
https://myasucourses.asu.edu/

# Lecture Roadmap

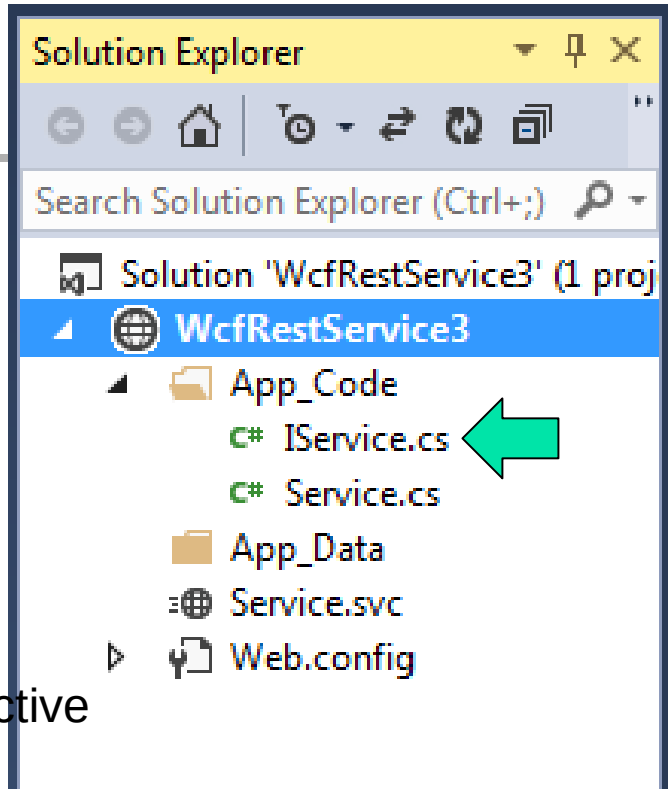
- **Developing REST Services:**
  - Developing RESTful Service
  - Defining Input and Output Formats
- **Case Study: Image Verifier in RESTful Service**
  - RESTful Service of a Random String Generator
  - RESTful Service of an Image Verifier
  - Synchronous RESTful Service Calls
- **Case Study: Consuming Services a Phone App**
  - Asynchronous SOAP Calls
  - Asynchronous RESTful Calls
- **RESTful services in ASP .Net Core**

# Last Lecture: Convert a SOAP Service to a RESTful Service

Textbook Chapter 7.3 on RESTful services

- Create a SOAP-based Service in WCF;
- Use a client to test the service to make sure it works;
- Convert the SOAP-based Service into a RESTful service in follow steps:
  1. Add "using Sytem.ServiceModel.Web" in file IService.cs;
  2. Add the attributes [WebGet] for each operation contract
  3. Add the following line in the service's markup source code
     Factory="System.ServiceModel.Activation.WebServiceHostFactory"
  4. Remove SOAP endpoint for access, so that HTTP can be directly used for accessing the service.

# Step 2: IService.cs




```
using System;
using System.ServiceModel;
using System.ServiceModel.Web;
[ServiceContract]
public interface Iservice {
    [OperationContract]
    [WebGet]    // Add this HTTP GET attribute/directive
    int absValue(int x);    [OperationContract]
    [WebGet(ResponseFormat = WebMessageFormat.Xml)]
    double PiValue();
    [OperationContract]
    [WebGet(UriTemplate = "add2?x={x}&y={y}", ResponseFormat =
WebMessageFormat.Json)]
    int addition(int x, int y);
}
```

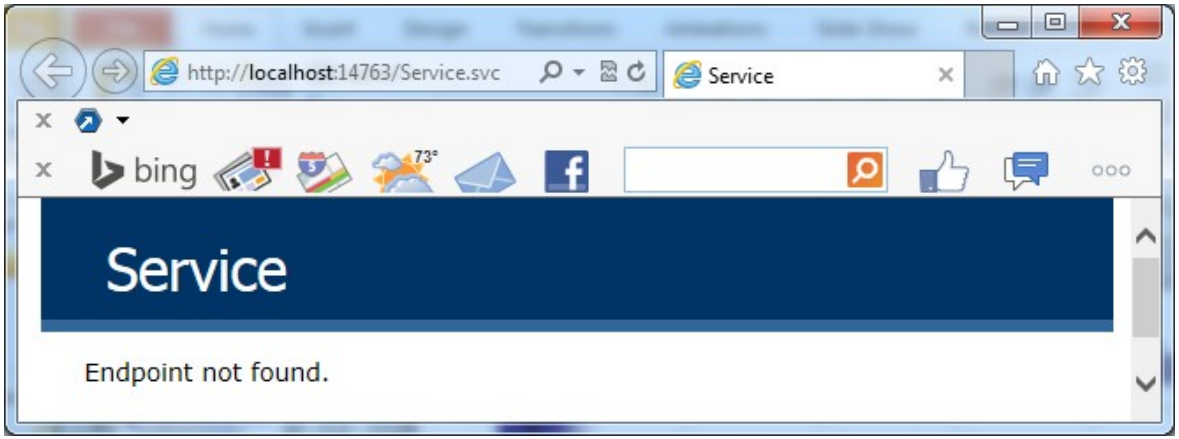
Add HTTP Get method and UriTemplate

add2 and addition are different

What are different ways of defining UriTemplate? To be answered later.

# Testing the Service in Browser

- View the Service in Browser




- Test the service in browser
  - http://localhost:14763/Service.svc/PiValue
    it returns: <double>3.1415926535897931</double>
  - http://localhost:14763/Service.svc/absValue?x=-27
  - it returns: <int>27</int>
  - http://localhost:14763/Service.svc/add2?x=15&y=17
    it returns: 32

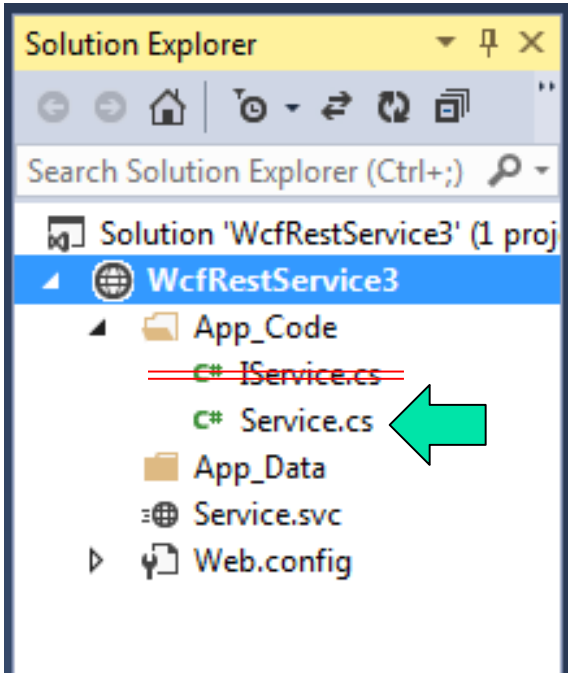
```
[WebGet(UriTemplate = "add2?x={x}&y={y}",
ResponseFormat = WebMessageFormat.Json)]
```

Hide the method name: Focus on the resource, instead of operations

# Creating RESTful Service without Using IService

## We define interface in Service.cs

```
using System; using System.ServiceModel; // Service.cs file
using System.ServiceModel.Activation; using System.ServiceModel.Web;
namespace RestService {
    [ServiceContract]
    [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
    [ServiceBehavior(InstanceContextMode =
InstanceContextMode.PerCall)]
//public class Service1 {
// Continued to next page
```




No end point will be created if IService.cs file is removed. You do not need to open Web.config file to remove endpoint.

# RESTful Service: Using UriTemplate

http://msdn.microsoft.com/en-us/library/system.servicemodel.web.webgetattribute.aspx

```
public class Service1 {
    [OperationContract]
    [WebGet(ResponseFormat = WebMessageFormat.Xml)]
    public double PiValue() {
        double pi = System.Math.PI; return (pi);
    }
    [OperationContract]
    [WebGet]    // Not use a UriTemplate, simply use default
    public int absValue(int x) {
        if (x >= 0) return (x); else return (-x);
    }
    [OperationContract]
    [WebGet(UriTemplate = "add2?x={x}&y={y}", ResponseFormat =
WebMessageFormat.Json)]    // Add this HTTP GET attribute/directive
    public int addition(int x, int y) { return (x+y); }
  }
}
```

Define Xml data format

Define two parameters

Define Json data format

# Testing Service Operations in Browser

localhost:49783/Service1/PiValue

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<double>3.1415926535897931</double>
```

Default parameter passing style

localhost:49783/Service1/absValue?x=-27

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<int>27</int>
```

localhost:49783/Service1/add2?x=15&y=17

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<int>32</int>
```

# Access the Deployed Services

- The service is deployed to ASU Service Repository at the address:

  http://neptune.fulton.ad.asu.edu/WSRepository/Services/WcfRestService4/Service1/

- We can test the remote service by replacing

  http://neptune.fulton.ad.asu.edu/WSRepository/Services/WcfRestService4/Service1/add2?x=15&y=17

  for “localhost:49783”:

neptune.fulton.ad.asu.edu/WSRepository/Services/WcfRestService4/Service1/add2?x=15&y=17

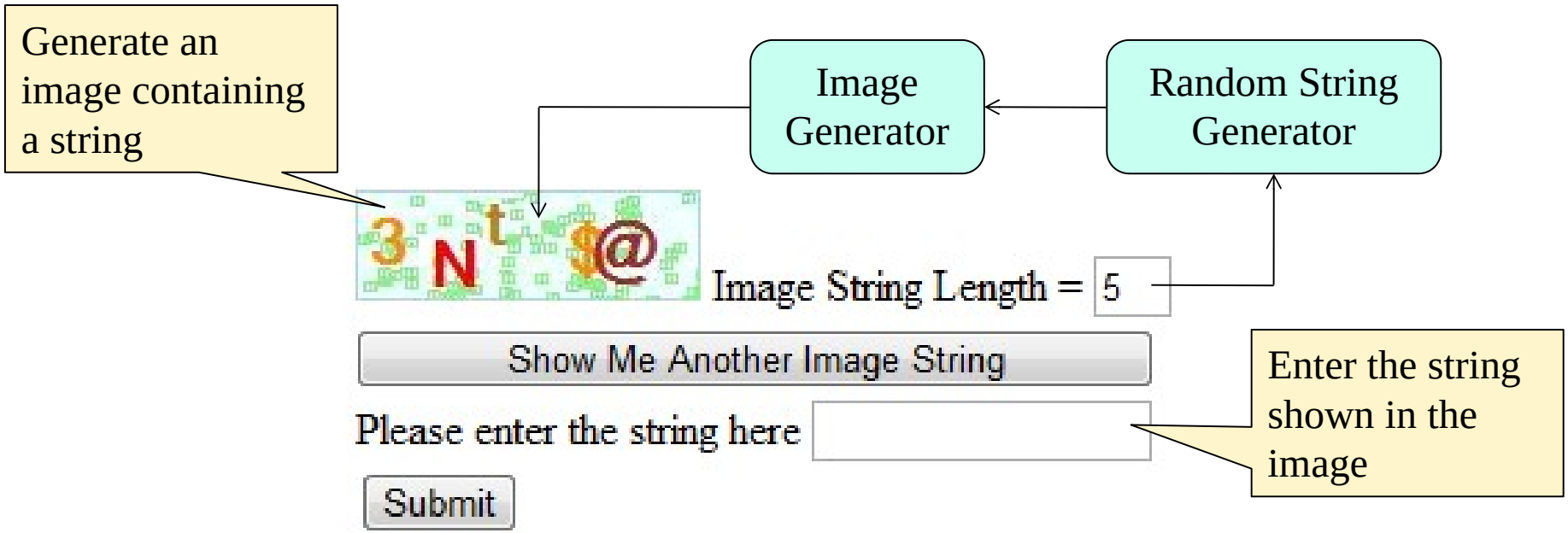
32

# Lecture Roadmap: Case Study 1

- **Developing REST Services:**
  - Developing RESTful Service
  - Defining Input and Output Formats
- **Image Verifier in RESTful Service**
  - RESTful Service of a Random String Generator
  - RESTful Service of an Image Verifier
  - Synchronous RESTful Service Calls
- Consuming Services in a Phone App
  - Asynchronous SOAP Calls
  - Asynchronous RESTful Calls
- RESTful services in ASP .Net Core (2nd Generation)

# Development of an Image Verifier

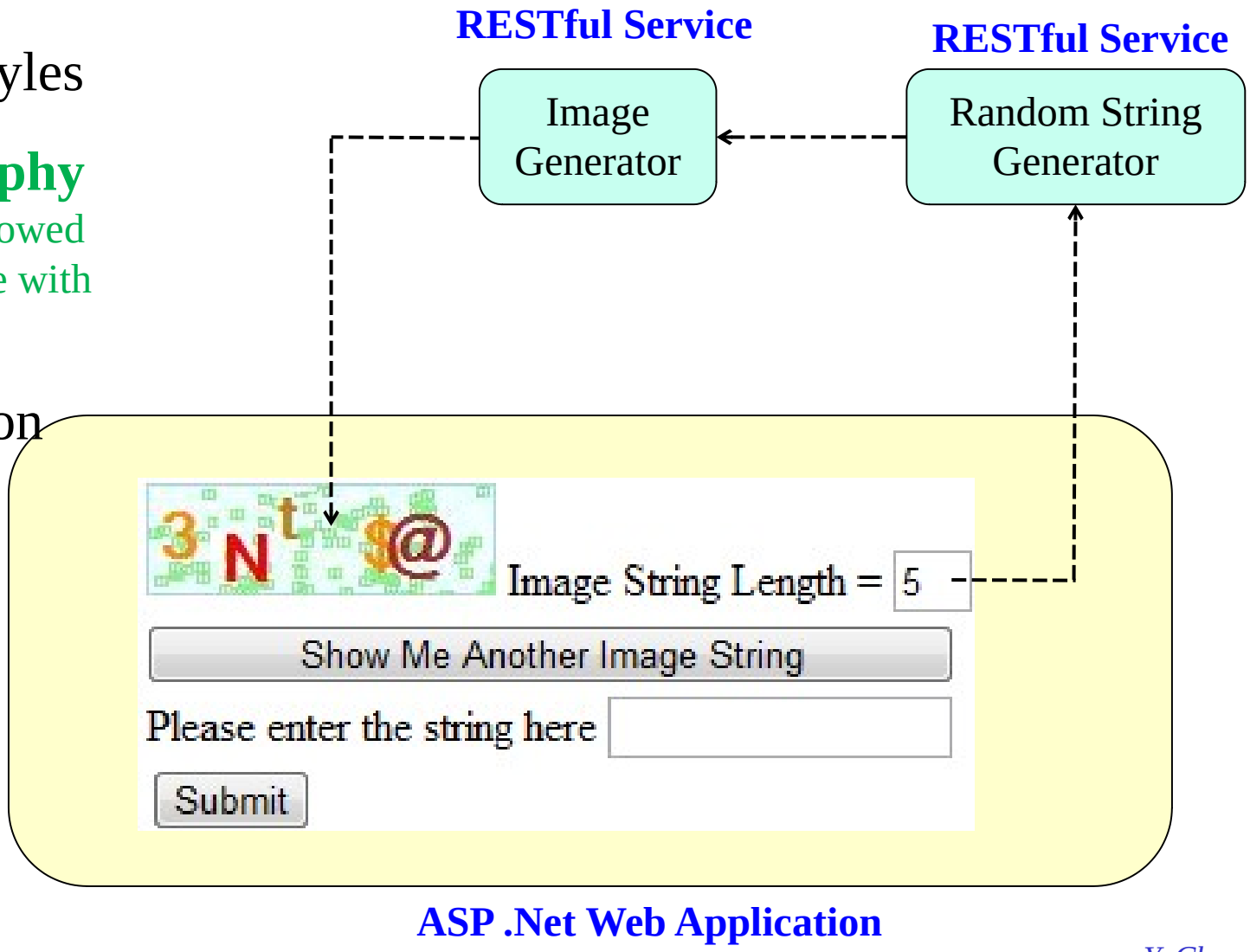
- Image verifiers have been widely used as a way of preventing programmed attached to Web sites.
- An image verifier consists of a random string generator and an image generator.



# Architecture Design of the Image Verifier

Composition Architecture Styles

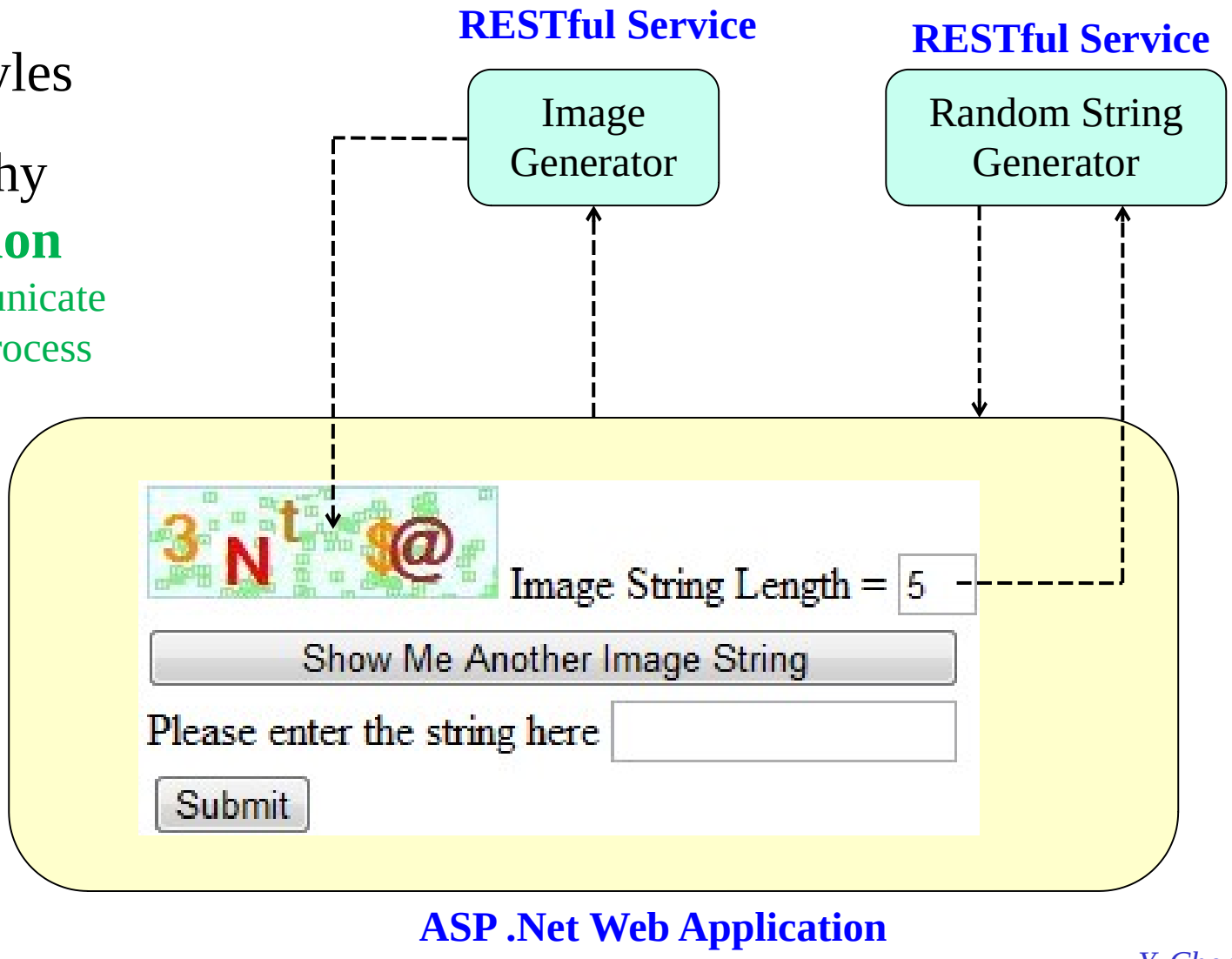
- ☑ **Choreography**
  Services are allowed to communicate with each other
- ☐ Orchestration



# Architecture Design of the Image Verifier

Composition Architecture Styles

- ☐ Choreography
- ☑ **Orchestration**
  Services communicate with a central process only



# Lecture Roadmap

- **Developing REST Services:**
  - Developing RESTful Service
  - Defining Input and Output Formats
- **Image Verifier in RESTful Service**
  - RESTful Service of a Random String Generator
  - RESTful Service of an Image Verifier
  - Synchronous RESTful Service Calls
- Consuming Services in Silverlight / Phone Apps
  - Asynchronous SOAP Calls
  - Asynchronous RESTful Calls
- RESTful services in ASP .Net Core (2nd Generation)

# Example: Random String Generator

- A RESTful service deployed at:


http://neptune.fulton.ad.asu.edu/WSRepository/Services/RandomString/Service.svc/

- It has two operations:
  - Take an integer (length) as input, it returns a random string of the given length
  - Without providing a parameter, it returns a random string with a random length between 4 and 20.
- The random string will contain an uppercase letter, a lowercase letter, a digit, and a special character. It can be used as a strong password.
- They can be accessed in URI:

http://neptune.fulton.ad.asu.edu/WSRepository/Services/RandomString/Service.svc/GetRandomString/5

http://neptune.fulton.ad.asu.edu/WSRepository/Services/RandomString/Service.svc/GetRandomString

# Random String Generator: IService.cs

We are using the style with svc endpoints created but disabled

```
using System; using System.ServiceModel;
using System.ServiceModel.Web;
[ServiceContract]
public interface IService {
    [WebGet(UriTemplate = "/GetRandomString", RequestFormat =
    WebMessageFormat.Xml, ResponseFormat =
    WebMessageFormat.Xml, BodyStyle =
    WebMessageBodyStyle.Bare)]
    string GetRandomString0();

    [WebGet(UriTemplate = "/GetRandomString/{Length}",
    RequestFormat = WebMessageFormat.Xml, ResponseFormat
    = WebMessageFormat.Xml, BodyStyle =
    WebMessageBodyStyle.Bare)]
    string GetRandomString(string Length);
}
```

Names must be different: cannot overload

Names to be used in URI can be the same

# Lecture Roadmap

- **Developing REST Services:**
  - Developing RESTful Service
  - Defining Input and Output Formats
- **Image Verifier in RESTful Service**
  - RESTful Service of a Random String Generator
  - RESTful Service of an Image Verifier
  - Synchronous RESTful Service Calls
- Consuming Services in Silverlight / Phone Apps
  - Asynchronous SOAP Calls
  - Asynchronous RESTful Calls
- RESTful services in ASP .Net Core (2nd Generation)

# Image Verifier Interface: IService.cs

```
using System; using System.IO;
using System.ServiceModel;
using System.ServiceModel.Web;
[ServiceContract]
public interface IService {
    [WebGet(UriTemplate = "/GetVerifierString/{myLength}",
    RequestFormat = WebMessageFormat.Xml, ResponseFormat =
    WebMessageFormat.Xml, BodyStyle = WebMessageBodyStyle.Bare)]
    string GetVerifierString(string myLength); // generates a string

    [WebGet(UriTemplate = "/GetImage/{myString}", RequestFormat =
    WebMessageFormat.Xml, ResponseFormat =
    WebMessageFormat.Xml, BodyStyle = WebMessageBodyStyle.Bare)]
    Stream GetImage(string myString); // generates image of the string
}
```

This operation will call the RandomString/Service.svc

# Image Verifier: Service.cs Architecture

RESTful service

**Class Service**

```
GetVerifierString (length)

{
    call remote service;
    return string;
}
```

```
GetImage(string)
{
    generate image;
    return image URI;
}
```

Return a random string

RESTful service

**Class Service**

/Service.svc/GetRandomString

/Service.svc/GetRandomString/5

# Image Verifier: Service.cs Code (1)

Notice how a RESTful service is access here

```
public class Service : IService {
    public string GetVerifierString(string myLength) {
        // Create the base address to the RandomString service
1       Uri baseUri = new
    Uri("http://neptune.fulton.ad.asu.edu/WSRepository/Services/RandomString/Service.svc");
        // Define UriTemplate for passing parameter
2       UriTemplate myTemplate = new UriTemplate("GetRandomString/{Length}");
        // Assign values to variable to obtain the complete URI
3       Uri completeUri = myTemplate.BindByPosition(baseUri, myLength);
4       WebClient channel = new WebClient(); // create a channel
5       byte[] abc = channel.DownloadData(completeUri); // return byte array
6       Stream strm = new MemoryStream(abc); // convert to mem stream
7       DataContractSerializer obj = new
8   DataContractSerializer(typeof(string));
9       string randString = obj.ReadObject(strm).ToString();
        return randString;
}
```

The purpose is to create a language-independent string

# Image Verifier: Service.cs Code (1) Explained

The service to be called:
http://neptune.fulton.ad.asu.edu/WSRepository/Services/RandomString/Service.svc/GetRandomString/5

From the previous page:

1 // Create the base address to the RandomString service

2 Uri baseUri = new Uri("http://neptune.fulton.ad.asu.edu/WSRepository/Services/RandomString/Service.svc");

// Create the Template to be used by the client

3 UriTemplate myTemplate = new UriTemplate("GetRandomString/{Length}");

// Assign values to variable to obtain the complete URI

Uri completeUri = myTemplate.BindByPosition(baseUri, myLength);

- Can we use string appending, instead?

Uri completeUri = baseUri + myLength;

# Image Verifier: Service.cs Architecture

RESTful service

**Class Service**

GetVerifierString (length)

```
{
    call remote service;
    return string;
}
```

GetImage(string)

```
{
    generate image;
    return image URI;
}
```

Return a random string

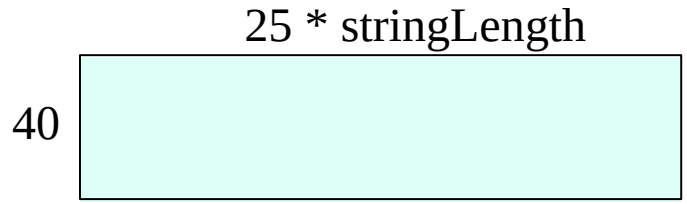
RESTful service

**Class Service**

/Service.svc/GetRandomString

/Service.svc/GetRandomString/5

# Image Verifier: Service.cs Code (2)




```
// This operation creates image based on the input string
public Stream GetImage(string myString) {
    WebOperationContext.Current.OutgoingResponse.ContentType = "image/jpeg";
    int mapwidth = (int)(myString.Length * 25);   // define bitmap width
    Bitmap bMap = new Bitmap(mapwidth, 40); // based on string length
    Graphics graph = Graphics.FromImage(bMap);
    graph.Clear(Color.Azure);                     // background color
    graph.DrawRectangle(new Pen(Color.LightBlue, 0), 0, 0, bMap.Width - 1, bMap.Height - 1); // draw a frame
```

We have two variables: bMap and graph.
We use graph variable to modify bMap.

# Image Verifier: Service.cs Code (3)

We continue to use graph to modify bMap variable.

```
Random rand = new Random();
Pen badPen = new Pen(Color.LightGreen, 0);
for (int i = 0; i < 100; i++) {    // create random noise pattern
    int x = rand.Next(1, bMap.Width - 1);
    int y = rand.Next(1, bMap.Height - 1);
    graph.DrawRectangle(badPen, x, y, 4, 3);
    graph.DrawEllipse(badPen, x, y, 2, 3);  // position and size
}
char[] charString = myString.ToCharArray();
Font font = new Font("Boopee", 18, FontStyle.Bold);
Color[] clr = {Color.Black, Color.Red, Color.DarkViolet, Color.Green,
Color.DarkOrange, Color.Brown, Color.DarkGoldenrod, Color.Plum };
```

Define an array of different colors

# Image Verifier: Service.cs Code (4)

```
// Draw the characters in the string onto the graphics object
for (int i = 0; i < myString.Length; i++) {
    int d = rand.Next(20, 25); // distance between characters
    int p = rand.Next(1, 15); // up and down position
    int c = rand.Next(0, 7); // randomly choose a color
    string str1 = Convert.ToString(charString[i]); //char->string
    Brush b = new System.Drawing.SolidBrush(clr[c]);
    graph.DrawString(str1, font, b, 1 + i * d, p);
}
System.IO.MemoryStream ms = new System.IO.MemoryStream();
bMap.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg);
ms.Position = 0;
graph.Dispose(); bMap.Dispose();
return ms;  // return the image generated
}
}
```

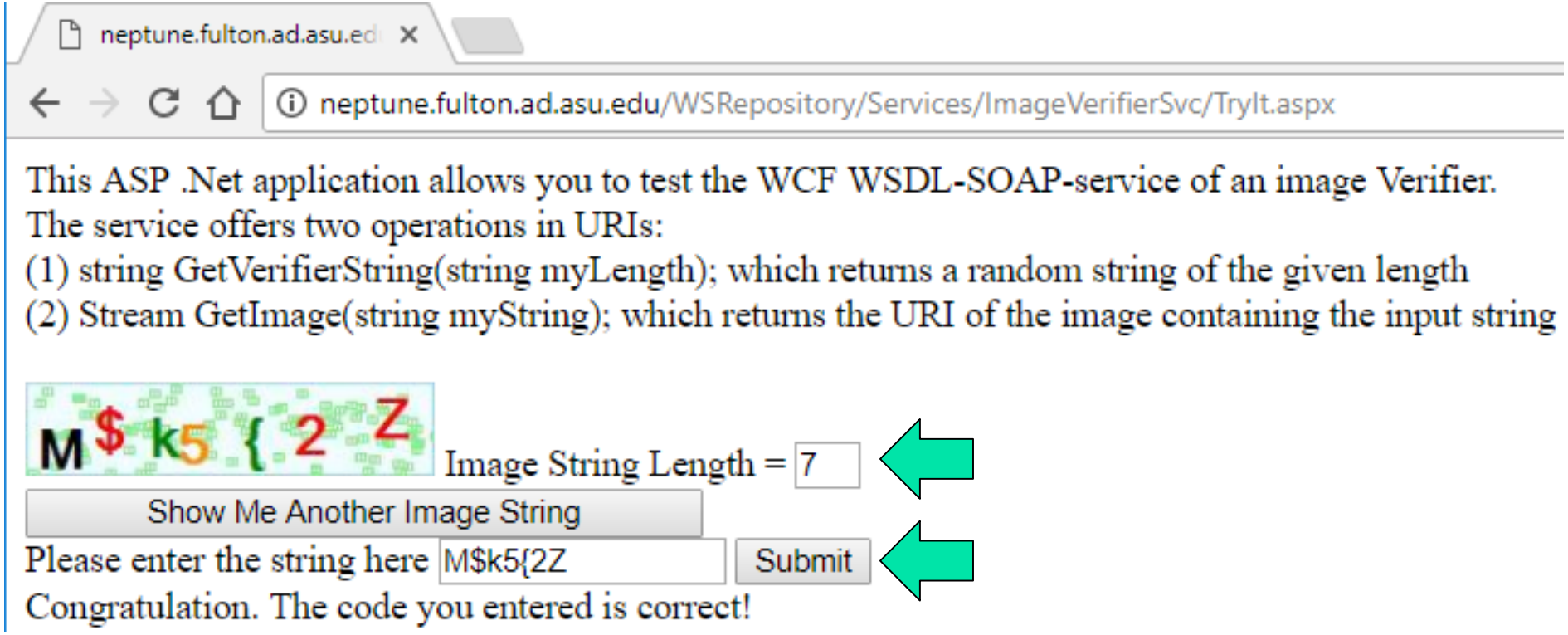
Now, we save bMap into memory string variable ms

# Lecture Roadmap

- **Developing REST Services:**
  - Developing RESTful Service
  - Defining Input and Output Formats
- **Image Verifier in RESTful Service**
  - RESTful Service of a Random String Generator
  - RESTful Service of an Image Verifier
  - Synchronous RESTful Service Calls
- Consuming Services in Silverlight / Phone Apps
  - Asynchronous SOAP Calls
  - Asynchronous RESTful Calls
- RESTful services in ASP .Net Core (2nd Generation)

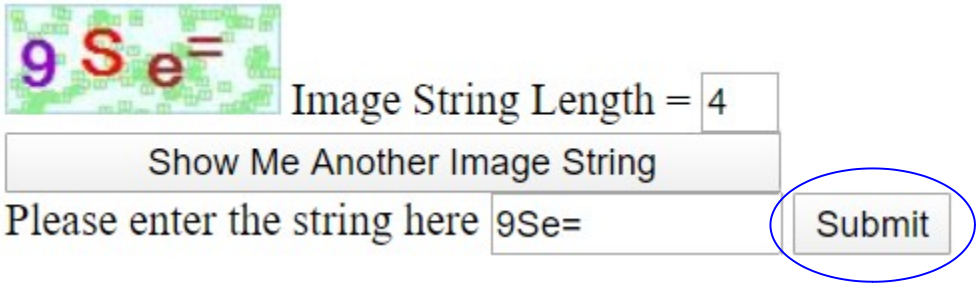
# GUI Design

http://neptune.fulton.ad.asu.edu/WSRepository/Services/ImageVerifierSvc/TryIt.aspx



# Code Behind the GUI: TryIt.aspx.cs (1)

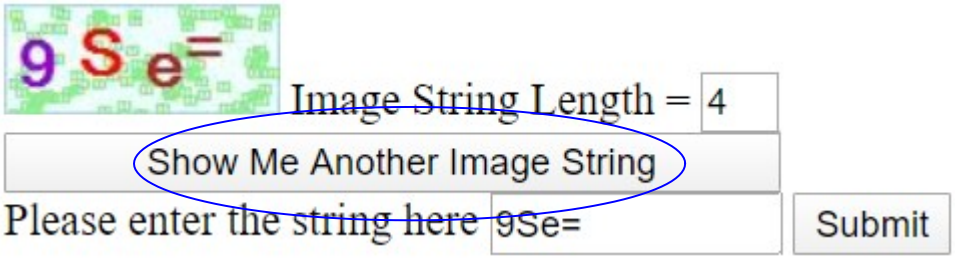
```
using System; using System.Net; using System.IO;
using System.Runtime.Serialization;
public partial class TestVerifier : System.Web.UI.Page {
    protected void Page_Load(object sender, EventArgs e) { }
    protected void btnVerifyImage_Click(object sender, EventArgs e) {
        if (Session["generatedStr"].Equals(TextBox1.Text)) {
            Label1.Text =
                        "Congratulation. The code you entered is correct!";

        }
        Else {
            Label1.Text = "I am sorry, the string you entered does not match
the image. Please try again!";
        }
    }
```



# Code Behind the GUI: TryIt.aspx.cs (2)

```
protected void btnGetImage_Click(object sender, EventArgs e) {
    // create the base address
    Uri baseUri = new
Uri("http://neptune.fulton.ad.asu.edu/WSRepository/Services/ImageVerifier/Service.svc/");
    // create the path from tree root to the child node
    UriTemplate myTemplate = new
                                UriTemplate("GetVerifierString/{myLength}");
    // Assign values to variable to complete URI
    Uri completeUri = myTemplate.BindByPosition(baseUri,
                                                    txtBoxlength.Text);
```

Prepare the URI to call the GetImage service

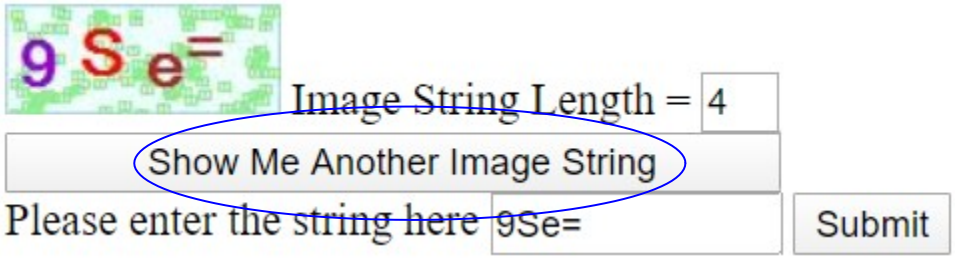


# Code Behind the GUI: TryIt.aspx.cs (3)

call the service GetVerifierString

```
WebClient channel = new WebClient();
byte[] abc = channel.DownloadData(completeUri);
Stream strm = new MemoryStream(abc);
DataContractSerializer obj = new
                                DataContractSerializer(typeof(string));
string generatedString = obj.ReadObject(strm).ToString();
Session["generatedStr"] = generatedString; // Save
Image1.Visible = true;
Image1.ImageUrl =
"http://neptune.fulton.ad.asu.edu/WSRepository/Services/ImageVerifier/Service.svc/GetImage/" +
                                                                generatedString;
btnGetImage.Text = "Show Me Another Image String";
  }
}
```

call the service GetImage and return image




Read textbook Chapter 5 on image generation for full detail

# Google and Microsoft RESTful Services

Example in Text Section 7.3.6

- Google map services
  - https://developers.google.com/maps/documentation/webservices/
- Microsoft Bing map services
  - http://msdn.microsoft.com/en-us/library/ff701713.aspx

## Google Maps APIs Web Services

The Google Maps web services are a collection of HTTP interfaces
maps applications. This guide serves only to introduce the web serv
different services. Individual documentation for each service is loca

- Google Maps Directions API
- Google Maps Distance Matrix API
- Google Maps Elevation API
- Google Maps Geocoding API
- Google Maps Geolocation API
- Google Maps Roads API
- Google Maps Time Zone API
- Google Places API Web Service

## Bing Maps REST Services

| | |
|---|---|
| Getting Started with the Bing Maps REST Services | Gets you started with the Bing Ma |
| What's New in the REST Services | Describes the latest updates to th |
| Getting Traffic Incident Data | Shows how to get traffic incident |
| Using the REST Services with .NET | Shows how to use .NET with the R |
| Locations API | Shows how to geocode and rever |
| Elevations API | Shows how to get elevation inforr |
| Imagery API | Shows how to use the Imagery AF such as map tile URLs and imager |

# Google Map Service Call Example

- http://maps.googleapis.com/maps/api/distancematrix/xml?origins=Phoenix+AZ|Tucson+AZ&destinations=Los+Angeles|San+Francisco%&mode=driving&lan

Add your API key to authenticate each request


```
<?xml version="1.0" encoding="UTF-8"?>
- <DistanceMatrixResponse>
    <status>OK</status>
    <origin_address>Phoenix, AZ, USA</origin_address>
    <origin_address>Tucson, AZ, USA</origin_address>
    <destination_address>Los Angeles, CA, USA</destination_address>
    <destination_address>San Francisco, CA, USA</destination_address>
  - <row>
    - <element>
        <status>OK</status>
      - <duration>
          <value>19217</value>
          <text>5 hours 20 mins</text>
        </duration>
      - <distance>
          <value>598638</value>
          <text>599 km</text>
        </distance>
      </element>
    - <element>
        <status>OK</status>
      - <duration>
          <value>38771</value>
          <text>10 hours 46 mins</text>
        </duration>
      - <distance>
          <value>1211572</value>
          <text>1,212 km</text>
        </distance>
```


| Distance / Time | Los Angeles | San Francisco |
|---|---|---|
| Phoenix | | |
| Tucson | | |

# Lecture Roadmap: Case Study 2

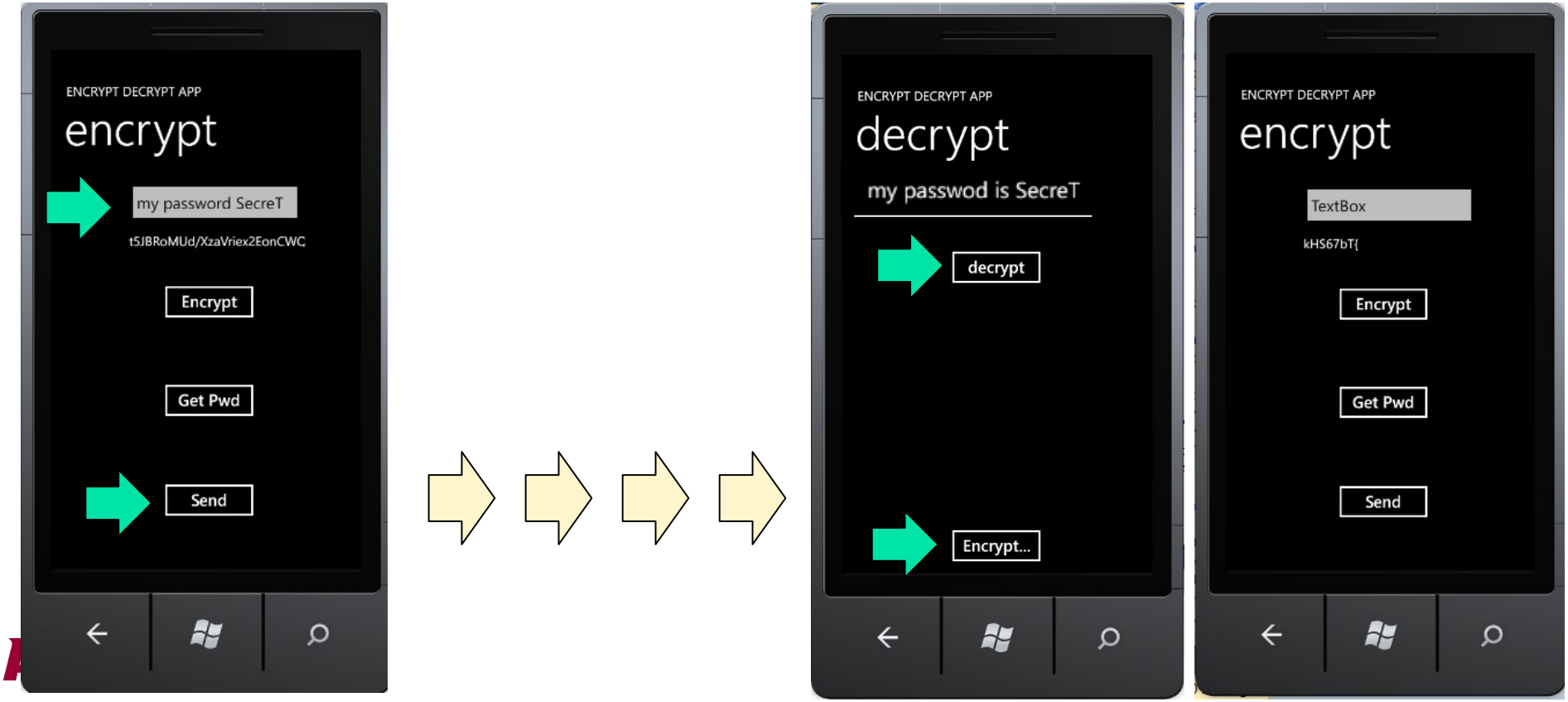
- **Developing REST Services:**
  - Developing RESTful Service
  - Defining Input and Output Formats
- **Image Verifier in RESTful Service**
  - RESTful Service of a Random String Generator
  - RESTful Service of an Image Verifier
  - Synchronous RESTful Service Calls
- **Consuming Services in a Phone App**
  - Asynchronous SOAP Calls
  - Asynchronous RESTful Calls
- RESTful services in ASP .Net Core (2nd Generation)

# Asynchronous for Mobile Communication

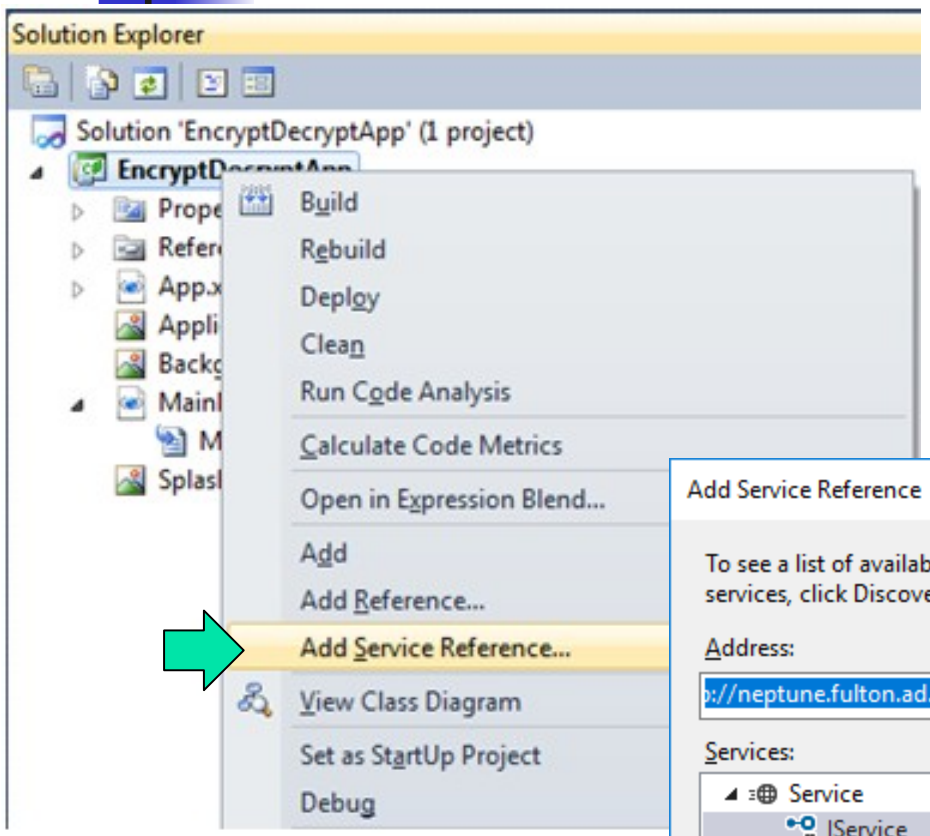
- Why is asynchronous communication important in mobile communication?
  - Mobile devices use wireless communication.
  - It has a higher network transient error rate.
  - Asynchronous communication split a longer communication period into two shorter periods.
  - It is less vulnerable to transient errors

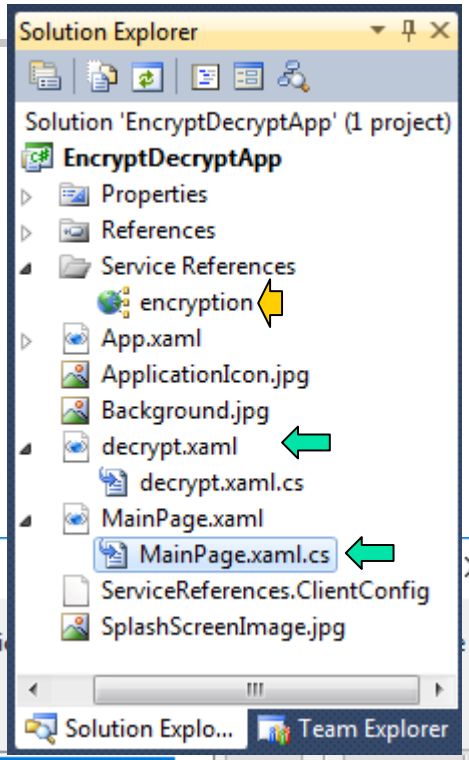
# Case Study: Secure Phone Messenger

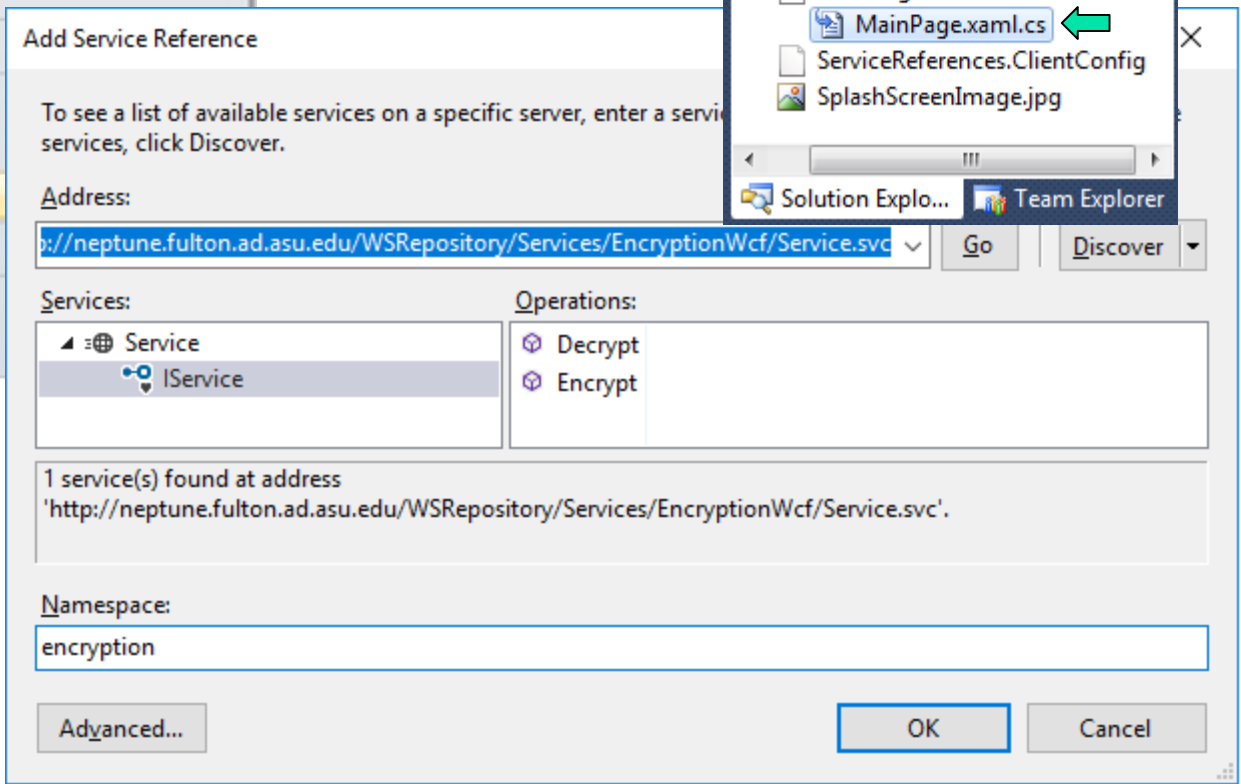
- Phone apps rely on Web services to provide necessary functionalities;
- Develop a secure phone messenger among friends and business partner



# Add Encryption/Decryption SOAP Service



# Code Behind GUI: Encrypt.xaml.cs

```
// namespace EncryptDecryptApp {
    private void Encrypt_Click(object sender, RoutedEventArgs e) {
        string msg1 = textBox1.Text;
        encryption.ServiceClient prxyEncrypt = new encryption.ServiceClient();
        prxyEncrypt.EncryptCompleted += new EventHandler
    <EncryptCompletedEventArgs>(EncryptEventhandler); //add callback event hdler
        prxyEncrypt.EncryptAsync(msg1);  // After service call complete
EventHandler<System.ComponentModel.AsyncCompletedEventArgs>
                                                (prxyEncrypt_CloseCompleted);
    }
```

Encrypt

1 2 3 4 5

Callback

```
    private void EncryptEventhandler(object sender,
                                                EncryptCompletedEventArgs e) {
        textBlock1.Text = e.Result;
    }
    private void send_Click(object sender, RoutedEventArgs e) {
        this.NavigationService.Navigate(new Uri("/decrypt.xaml?msg=" +
textBlock1.Text, UriKind.Relative));
} } }
```

Send

6

Could be a global URI

# Making Asynchronous Call Using Event

encryption.ServiceClient prxyEncrypt = new encryption.ServiceClient();

1 Create prxyEncrypt object

prxyEncrypt.EncryptCompleted event

prxyEncrypt.EncryptAsync( ) // service method

2 Add Event Handler

5

prxyEncrypt.EncryptCompleted event EncryptEventhandler

prxyEncrypt.EncryptAsync( ) // service method

3 Call service and return

prxyEncrypt.EncryptAsync(msg1) // String to be encrypted

4 Service call completes

prxyEncrypt.EncryptAsync(msg1)

6

5 Trigger event

prxyEncrypt.EncryptCompleted event occur

6

```
// EncryptEventhandler Code
private void EncryptEventhandler(object sender, EncryptCompletedEventArgs e)
{
        textBlock1.Text = e.Result;
}
```

# Code Behind GUI: decrypt.xaml.cs

decrypt.xaml

decrypt.xaml.cs

```
// namespace EncryptDecryptApp  {
        private void decrypt_Click(object sender, RoutedEver
            string msgEncrypted = textBlock1.Text;
            encryption.ServiceClient prxyEncrypt = new encryption.ServiceClient();
            prxyEncrypt.DecryptCompleted += new
EventHandler<DecryptCompletedEventArgs>(prxyEncrypt_DecryptAsync);
            prxyEncrypt.DecryptAsync(msgEncrypted);
        }
        private void prxyEncrypt_DecryptAsync(object sender,
                                                DecryptCompletedEventArgs e) {
            textBlock1.Text = e.Result;
        }
        private void Encrypt_Click(object sender, RoutedEventArgs e) {
            string msg = "";
            this.NavigationService.Navigate
                        (new Uri("/MainPage.xaml?msg=" + msg, UriKind.Relative));
 }  }  }
```

decrypt

Encrypt...

Could be a global URI

# Lecture Roadmap

- **Developing REST Services:**
  - Developing RESTful Service
  - Defining Input and Output Formats
- **Image Verifier in RESTful Service**
  - RESTful Service of a Random String Generator
  - RESTful Service of an Image Verifier
  - Synchronous RESTful Service Calls
- **Consuming Services in Phone Apps**
  - Asynchronous SOAP Calls
  - Asynchronous RESTful Calls
- RESTful services in ASP .Net Core

# Calling RESTful Service from Phone App

http://www.public.asu.edu/~ychen10/teaching/cse446sie/AsyncRest.mp4


- The RESTful service to call is

http://neptune.fulton.ad.asu.edu/WSRepository/Services/RandomString/Service.svc/GetRandomString/{length}

- For example

http://neptune.fulton.ad.asu.edu/WSRepository/Services/RandomString/Service.svc/GetRandomString/14

- It returns a “Strong Password” in XML format:

<string xmlns="http://schemas.microsoft.com/2003/10/Serialization/">

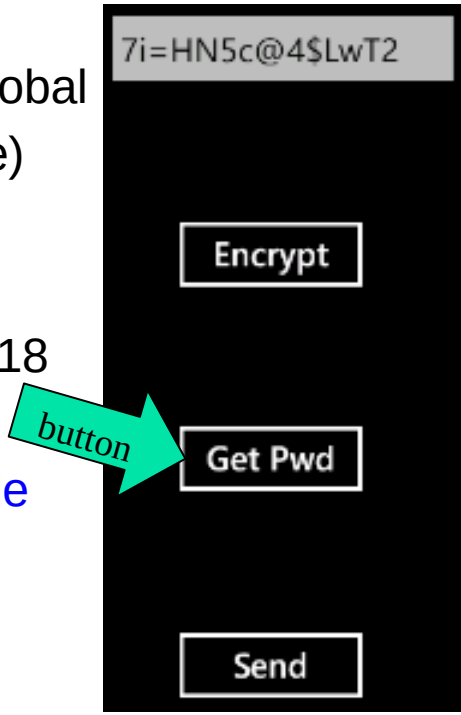
7i=HN5c@4$LwT2

</string>

# The Code Behind the Button "Get Pwd"

```
public static string spassword = "Waiting for call back"; //static-global
private void btnGetPwd_Click(object sender, RoutedEventArgs e)
    {
        Random rnd = new Random();
        Int32 length = rnd.Next(6, 18); random # between 6 and 18
        string toDisplay = spassword;
        GetPwd(length);  // It puts the result in spassword variable
        // Async! It will not have the value in the next statement
        if (spassword != "Waiting for call back")
        {
            XDocument xd = XDocument.Parse(spassword);
            XElement xe = xd.Root;
            toDisplay = xe.Value;
        }
        textBox1.Text = toDisplay;
    }
```

Continued on next Page




We will obtain the password from the previous call, this this call!

# The Code Behind the Button "Get Pwd"

```
public static void GetPwd(int length) {
    string baseUrl = "http://neptune.fulton.ad.asu.edu/WSRepository/Services/RandomString/
Service.svc/GetRandomString/";
    string len = Convert.ToString(length);
    string fullUrl = baseUrl + len;
    HttpWebRequest hwReq1 = (HttpWebRequest)HttpWebRequest.Create(new Uri(fullUrl));
   hwReq1.BeginGetResponse(new AsyncCallback(myCallbackFunc), hwReq1);
} // This method returns immediately
private static void myCallbackFunc(IAsyncResult  requestObj) {
    HttpWebRequest  hwReq2 = (HttpWebRequest)requestObj.AsyncState;
    HttpWebResponse hwResponse =
                        (HttpWebResponse)hwReq2.EndGetResponse(requestObj);
    using (StreamReader sReader =
                        new StreamReader(hwResponse.GetResponseStream())) {
        spassword = sReader.ReadToEnd().ToString();
    // Result save into this static variable spassword
}
```

RESTful service

Create request object

Call handler

Make request

Get response

Extract string

# The Code Explained

Create request object, but not calling

```
HttpWebRequest hwReq1 = (HttpWebRequest)HttpWebRequest.Create(new Uri(fullUrl));
hwReq1.BeginGetResponse(new AsyncCallback(myCallbackFunc), hwReq1);
} // This method returns immediately
private static void myCallbackFunc(IAsyncResult  requestObj) {
    HttpWebRequest  hwReq2 = (HttpWebRequest)requestObj.AsyncState;
    HttpWebResponse hwResponse =
                    (HttpWebResponse)hwReq2.EndGetResponse(requestObj);
    using (StreamReader sReader =
                    new StreamReader(hwResponse.GetResponseStream())) {
    spassword = sReader.ReadToEnd().ToString();
    // Result save into this static variable spassword
}
```

Call handler, with callback func and the object

Object from caller

Make the request in async

Get the response from the request

Extract string

Save string into global variable

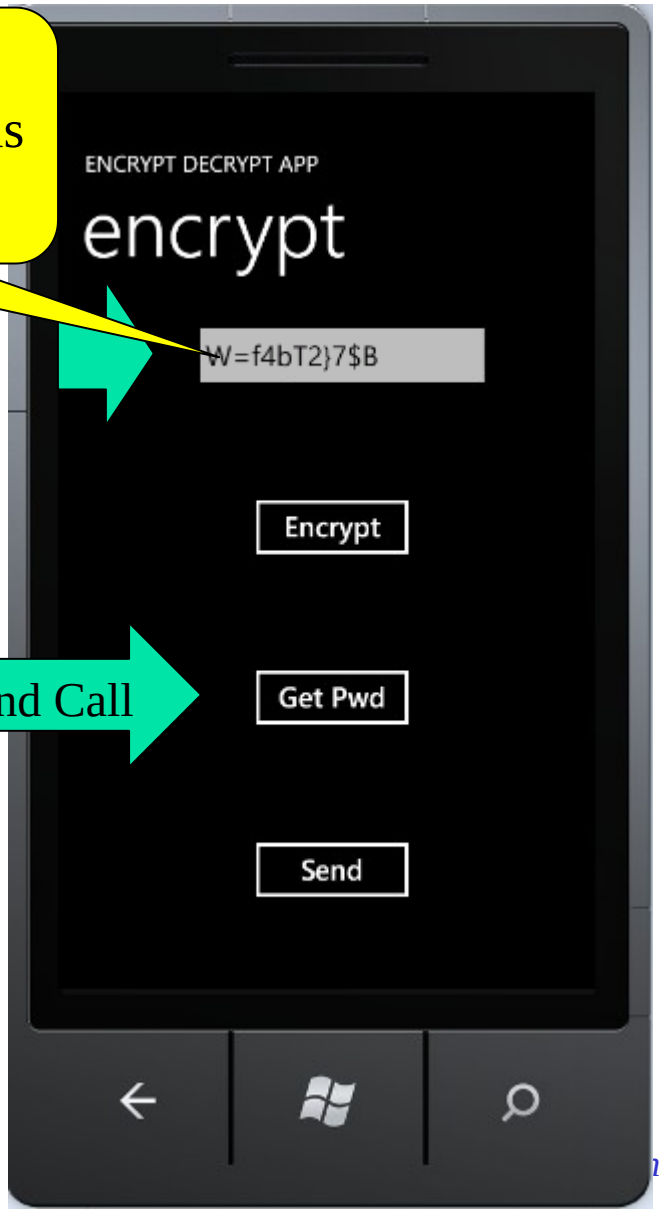
# Demonstration First Call and Second Call

http://www.public.asu.edu/~ychen10/teaching/cse446sie/AsyncRest.mp4

We obtain the password from the previous call, this this call!


First Call

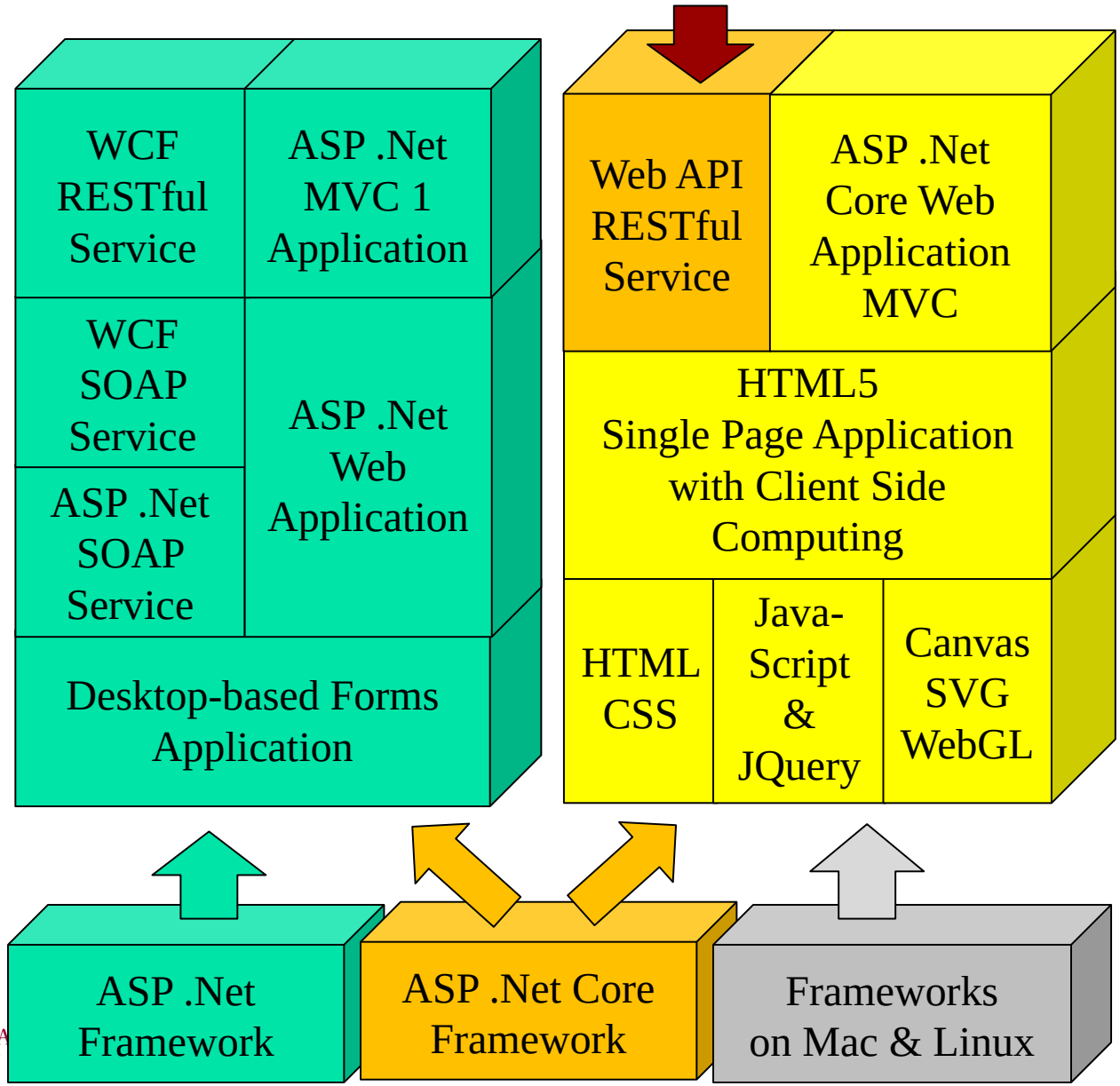
Second Call



# Lecture Roadmap

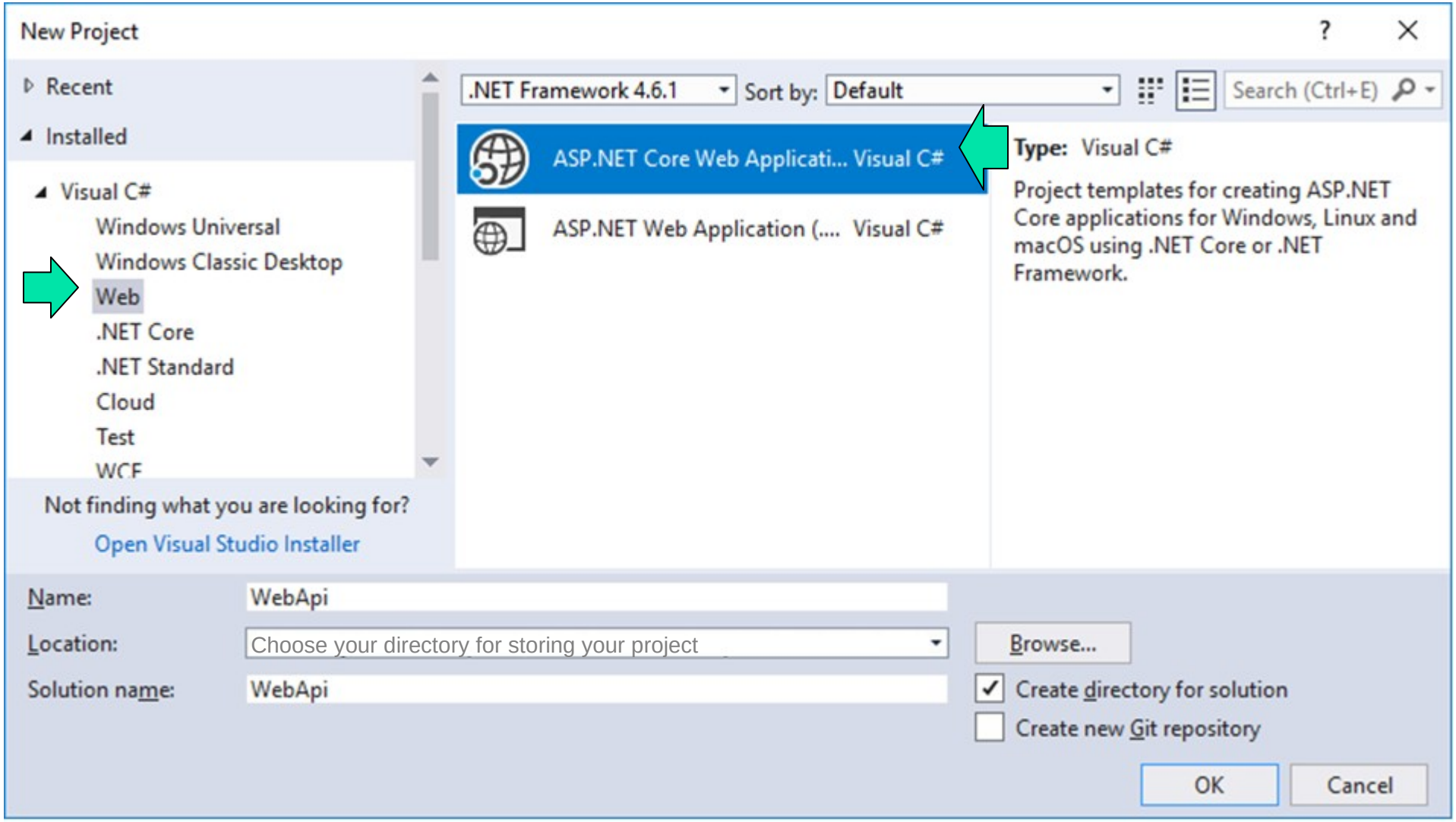
- **Developing REST Services:**
  - Developing RESTful Service
  - Defining Input and Output Formats
- **Image Verifier in RESTful Service**
  - RESTful Service of a Random String Generator
  - RESTful Service of an Image Verifier
  - Synchronous RESTful Service Calls
- **Consuming Services in Phone Apps**
  - Asynchronous SOAP Calls
  - Asynchronous RESTful Calls
- **RESTful services in ASP .Net Core**

# ASP .Net Core Framework: The 2nd Generation

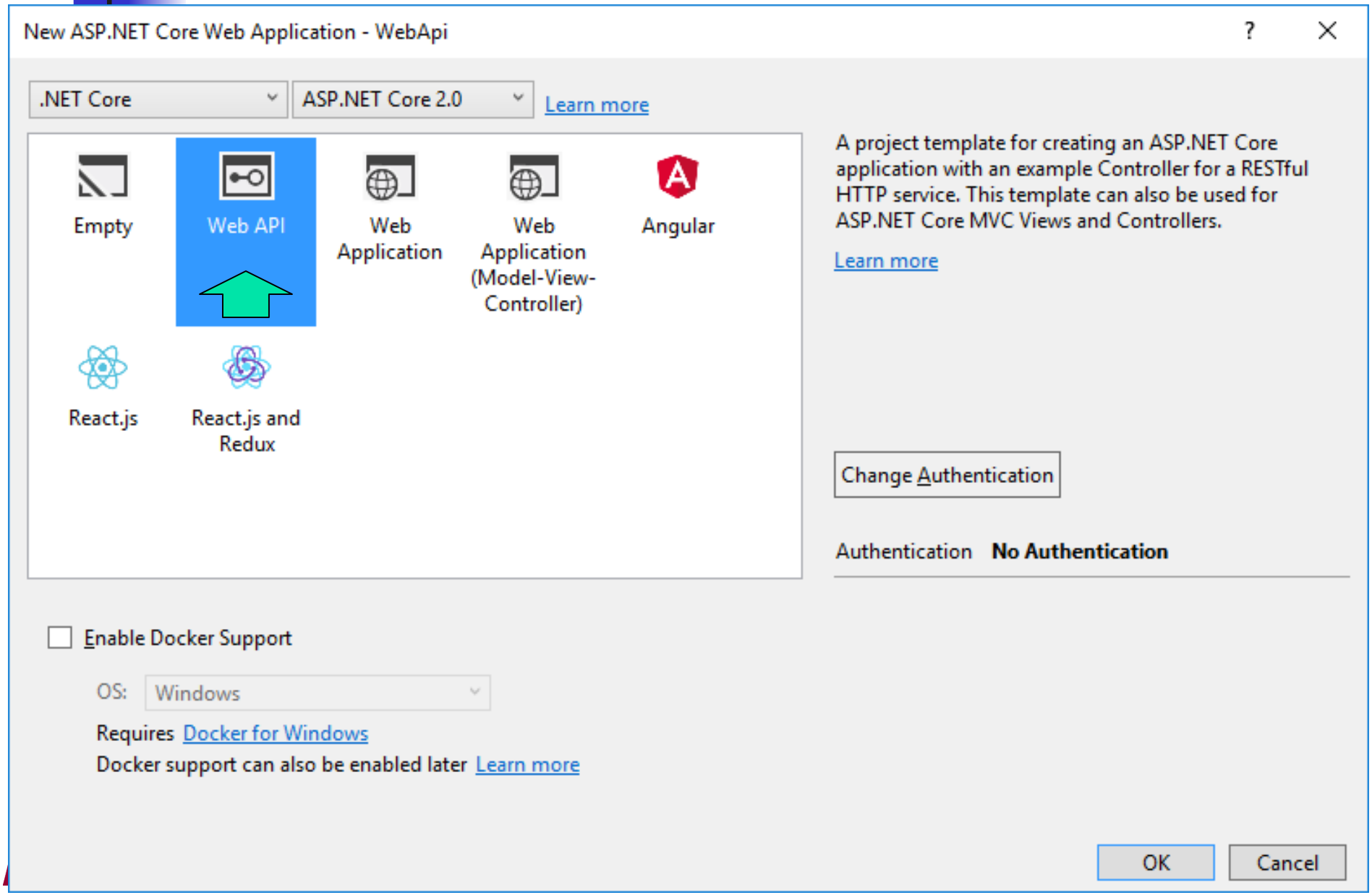


# Using ASP .Net Core Web Application

- New  Project …



# Choose Web API Template



# Service Templates, Add Your Code …

WebApi | ValuesController.cs

WebApi | WebApi.Controllers.Valu | Put(int id, string value)

```
namespace WebApi.Controllers
{
    [Route("api/[controller]")]
    0 references
    public class ValuesController : Controller
    {
        // GET api/values
        [HttpGet]
        0 references
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/values/5
        [HttpGet("{id}")]
        0 references
        public string Get(int id)
        {
            return "value";
        }
```

Solution Explorer

Search Solution Explorer (Ctrl+

- Solution 'WebApi' (1 project)
  - WebApi
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
      - ValuesController.cs
        - ValuesController
    - appsettings.json
    - Program.cs
    - Startup.cs

# Lecture Summary

- **Developing REST Services:**
  - Developing RESTful Service
  - Defining Input and Output Formats
- **Image Verifier in RESTful Service**
  - RESTful Service of a Random String Generator
  - RESTful Service of an Image Verifier
  - Synchronous RESTful Service Calls
- **Consuming Services in Phone Apps**
  - Asynchronous SOAP Calls
  - Asynchronous RESTful Calls
- **RESTful services in ASP .Net Core**

# Next Lecture