# Practical Lecture 5 Creating a Web Service and Client Website

## Practical Session Structure

1. Introduction
2. Building a business component
3. Building an admin GUI
4. Introducing .NET remoting
5. **Creating a web service and client website**
6. Developing a Java client

2

## Overview

- In order to start this session, you need to have completed all of the practical lecture 4
- In this lecture we will take the business component built in practical lecture 2 and add a web service façade to it.
- We will then build a website that uses the web service

3

## Learning Objectives

- Understand the concepts involved in Web Services
- Create a web service that acts as a façade to an existing business component
- Implement an ASP.NET website that communicates with the web service

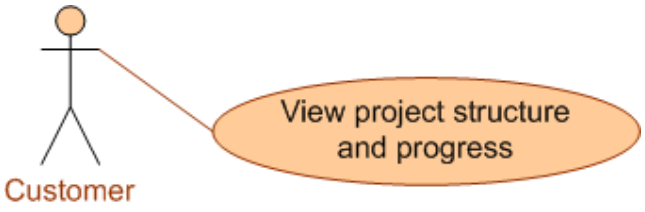
4

# Introduction

- In this practical session we will:
  - Add code to the PTSLibrary component to expose some of its methods as a web service. We will in fact create two web services, one for a customer client application and one for a team leader application
  - Implement a website using ASP.NET that uses the customer web service to allow a user to log in and view project details

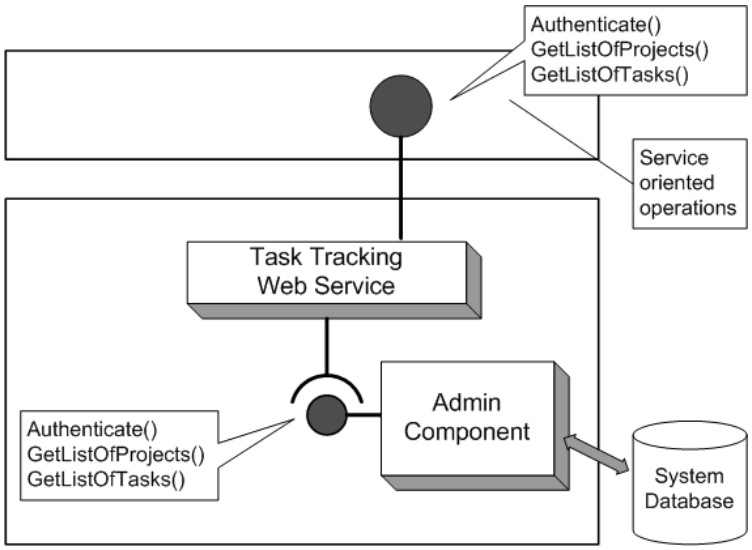
5

# What are we building?




6

## What are we building?




7

## Distributed Systems in .NET

- As mentioned in the previous lecture, web services work across platforms, so can be used to provide services to clients that are not under your control and could be written in any language that supports web services

8

# Web services

- A new breed of Web application
- Self-contained, self-describing, modular applications that can be published, located, and invoked across the Web
- Perform functions: anything from simple requests to complicated business processes
- Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service

*IBM web service tutorial*

9

# Web Services

- Distributed computing model based on asynchronous messaging (XML)
  - Support dynamic application integration over the Web
  - Web Services connect computers and devices with each other using the Internet to exchange data and access services
  - On-the-fly software creation through the use of loosely coupled, reusable software components
  - Business services can be distributed over the Internet

10

## Designing Web Services

- Requirements
  - Based on standards (e.g. HTTP, SOAP)
  - Minimal amount of required infrastructure is assumed
    - Only a minimal set of standards must be implemented
  - Very low level of application integration is expected
    - But may be increased in a flexible way
  - Focuses on messages and documents, not on APIs

11

## Web Services Framework

- Framework can be described in terms of
  - What goes “on the wire”:
    Formats and protocols
  - What describes what goes on the wire:
    Description languages
  - What allows us to find these descriptions:
    Discovery of services

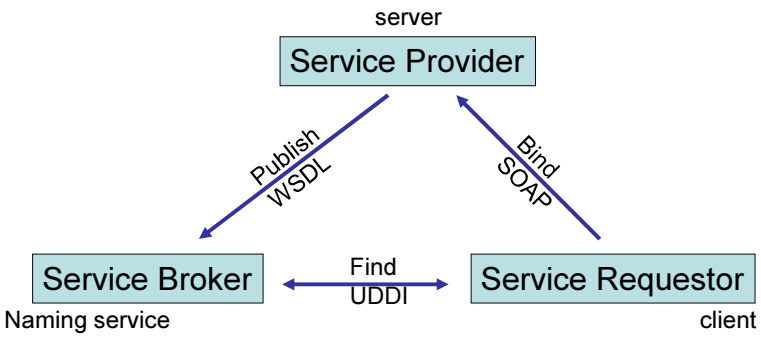
12

## Web Service Architecture. Components

- *Service providers* :
  - publish available services and offer bindings for services
- *Service brokers*
  - allow service providers to publish their services
  - provide mechanisms to locate services and their providers
- *Service requestor*
  - uses the service broker to find a service and then
  - invokes (or binds) the service offered by a service provider

13

## Service-oriented architecture




14

## XML Messaging: SOAP

- SOAP 1.1 defined:
  - An XML envelope for XML messaging,
    - Headers + body
  - An HTTP binding for SOAP messaging.
    - SOAP is “transport independent”.
  - A convention for doing RPC.
  - An XML serialization format for structured data
- SOAP Attachments adds
  - How to carry and reference data attachments using in a MIME envelope and a SOAP envelope.

15

## Web Services Description Language WSDL

- Defines services as collections of network endpoints or *ports*
- Provides functional description of network services:
  - IDL description
  - Protocol and deployment details
  - Platform independent description.
  - Extensible language.

16

## Using WSDL

1. As extended IDL: WSDL allows tools to generate compatible client and server stubs.
   - Tool support for top-down, bottom-up and “meet in the middle” development.
2. Allows industries to define standardized service interfaces.
3. Allows advertisement of service descriptions, enables dynamic discovery and binding of compatible services.
   - Used in conjunction with UDDI registry
4. Provides a normalized description of heterogeneous applications.

17

## UDDI Overview

- Universal Description Discovery and Integration protocol
- UDDI defines the operation of a service registry:
  - Data structures for registering
    - Businesses
    - Technical specifications: tModel is a keyed reference to a technical specification.
    - Service and service endpoints: referencing the supported tModels
  - SOAP Access API
  - Rules for the operation of a global registry
    - “private” UDDI nodes are likely to appear, though.

18

# For more information

- SOAP

http://www.w3c.org/TR/soap

- WSDL

http://www.w3c.org/TR/wsdl

- UDDI

http://www.uddi.org

- WSFL

http://www.ibm.com/software/webservices

19

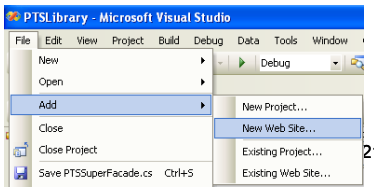
# Visual Studio Development Server

- Visual Studio .NET 2005 comes with an in-built ASP.NET development server
- This is great for development as you can test your web services and websites without having to deploy to a web server
- The web server starts automatically when you run a web service or website project
- The address is http://localhost:*port*
  - Where port is a port number chosen by Visual Studio

20

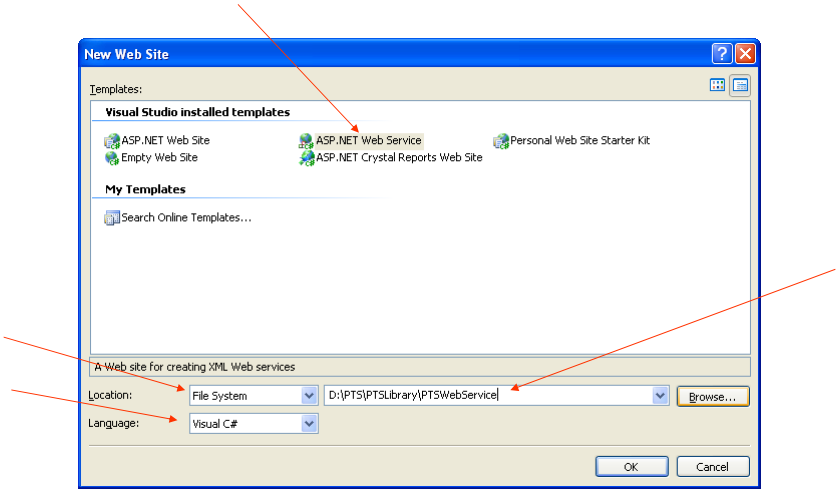
# Creating the Web Service

- We will create the web service as an *ASP.NET Web Service* in the PTSLibrary solution
- Open Visual Studio 2005
- Open the PTSLibrary solution created in lecture 2
- Go to *File -> Add -> New WebSite*
- Select *ASP.NET Web Service* as the template, location to *File System* and *Visual C#* as the language
- Name the project PTSWebService and save it in a suitable location




21

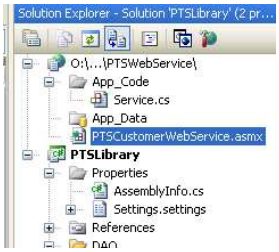
# Creating the Web Service /2




22

# Creating the Web Service /3

- Visual Studio will automatically create a web service file (Service.asmx)
- Rename this to *PTSCustomerWebService.asmx*
  - This will be our web service to be used by the Customer browser website
  - The only functionality this needs to offer is authentication and retrieval of project details




23

# Web Service Skeleton Code

- The following code is generated:

```
1  using System;
2  using System.Web;
3  using System.Web.Services;
4  using System.Web.Services.Protocols;
5
6  [WebService(Namespace = "http://tempuri.org/")]
7  [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
8  public class Service : System.Web.Services.WebService
9  {
10     public Service () {
11
12         //Uncomment the following line if using designed components
13         //InitializeComponent();
14     }
15
16     [WebMethod]
17     public string HelloWorld() {
18         return "Hello World";
19     }
20
21 }
22
```

Web Service related namespaces are imported

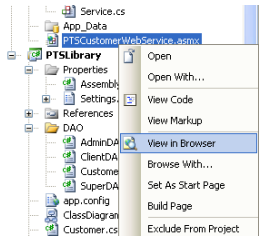
Inherits from WebService

Any method you want to make available through the web service needs to be marked as a ***[WebMethod]***

24

# Run the Web Service

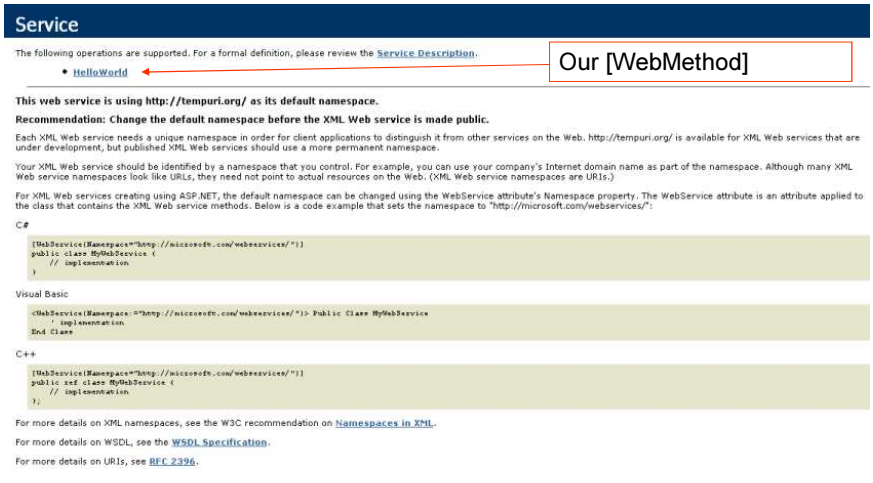
- Right-click on the *PTSCustomerWebService* file and select *View in Browser*
- This should start the development server and open a web browser displaying the operations supported by the web service (see next slide)





25

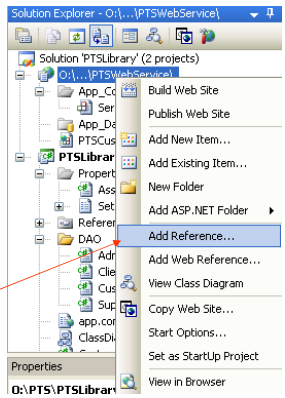
# Running the Web Service




26

# Adding References

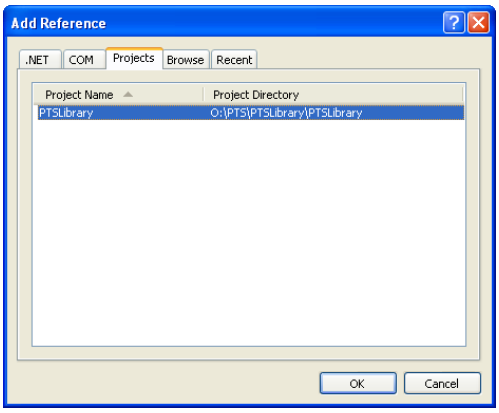
- As the web service just provides a façade for access to functionality in PTSLibrary, this project needs to be added as a reference
- Righ-click on the PTSWebService project and select *Add Reference…*




27

# Adding References /2

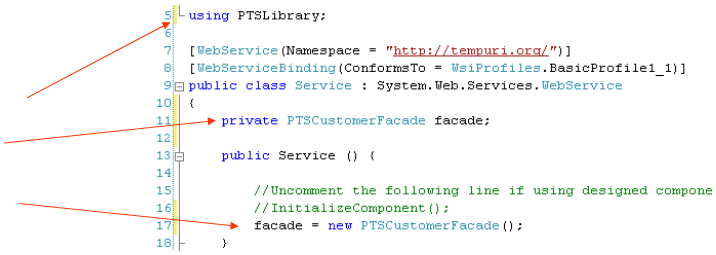
- Because the PTSLibrary project is in the same project, we can just select it from the *Projects* tab




28

# Accessing PTSLibrary

- Add the code to access the PTSCustomerFacade:



```
using PTSLibrary;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    private PTSCustomerFacade facade;

    public Service () {

        //Uncomment the following line if using designed compone
        //InitializeComponent();
        facade = new PTSCustomerFacade();
    }
```

29

# Calling the Façade Methods

- The methods defined here will only act as a level of indirection of the methods in the PTSCustomerFacade class

Both methods will be available to call on the web service

```
    [WebMethod]
    public int Authenticate(string username, string password)
    {
        return facade.Authenticate(username, password);
    }

    [WebMethod]
    public Project[] GetListOfProjects(int customerId)
    {
        return facade.GetListOfProjects(customerId);
    }
```

30

# No-Argument Constructors

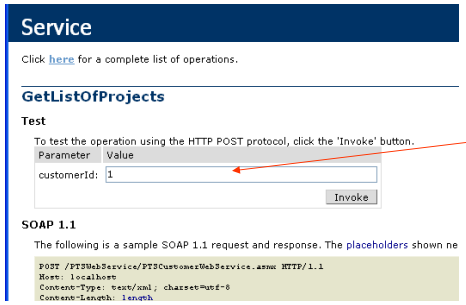
- Our business classes will be serialised and deserialised by the web service
- This process requires them to have no-argument constructors
- Introduce empty no-argument constructors for the business classes:
  - Project
  - Team
  - Task
  - Customer
  - TeamLeader

```
using System;
using System.Collections.Generic;
using System.Text;

namespace PTSLibrary
{
    [Serializable]
    public class Customer : User
    {
        public Customer() { }

        public Customer(string name, int id)
        {
            this.name = name;
            this.id = id;
        }
    }
}
```

31

# Testing the Web Service

- Make sure the database server is running and the database is available
- Run the web service in a browser and invoke the Authenticate and GetListOfProjects web methods

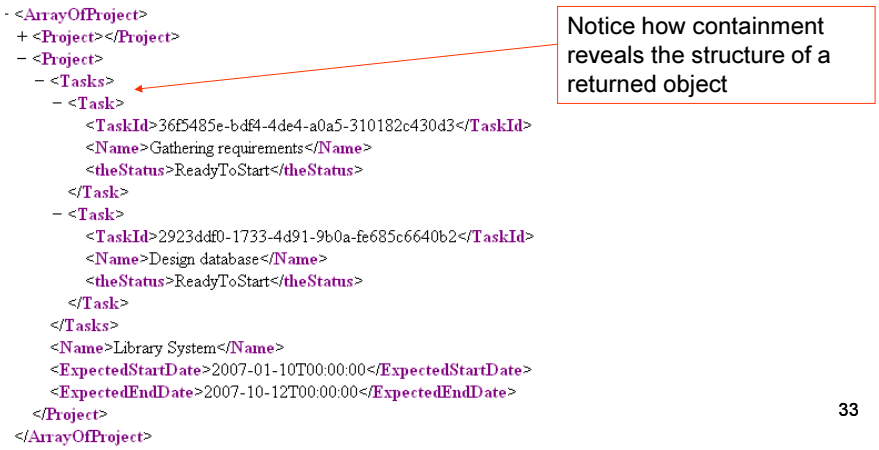



Notice how expected parameters are passed into the web method

32

# Testing the Web Service /2

- When a web method is invoked, the message is returned in XML (SOAP)




33

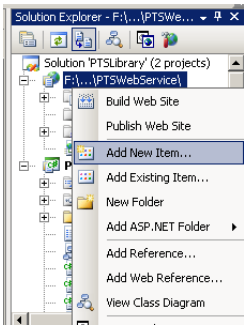
# Creating a Second Web Service

- Now that you have seen how easy it is to create a web service, let's create a second web service, to be used by the Java clients (the team leader's application, that we will build in the next session)
- The same two methods as for the customer web service will be created
  - The only difference is that the GetListOfProjects takes the teamId as a parameter, not the customerId

34

# Creating a Second Web Service /2

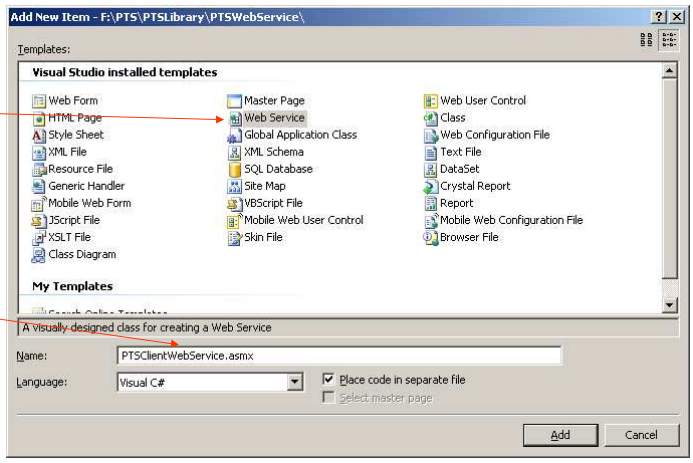
- Right-click on the PTSWebService project and select *Add New Item…*




35

# Creating a Second Web Service /3

- Select the *Web Service* template and name it *PTSClientWebService*




36

# Creating a Second Web Service /4

- Add the code to access PTSClientFacade

```
6   using PTSLibrary;
7
8   /// <summary>
9   /// Summary description for PTSClientWebService
10  /// </summary>
11  [WebService(Namespace = "http://tempuri.org/")]
12  [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
13  public class PTSClientWebService : System.Web.Services.WebService
14  {
15      private PTSClientFacade facade;
16
17      public PTSClientWebService()
18      {
19
20          //Uncomment the following line if using designed components
21          //InitializeComponent();
22          facade = new PTSClientFacade();
23      }
```

Remember, it's PTSClientFacade now, not PTSCustomerFacade

37

# Creating a Second Web Service /5

- Add the two web methods as shown here:

```
[WebMethod]
public TeamLeader Authenticate(string username, string password)
{
    return facade.Authenticate(username, password);
}

[WebMethod]
public Project[] GetListOfProjects(int teamId)
{
    return facade.GetListOfProjects(teamId);
}
```

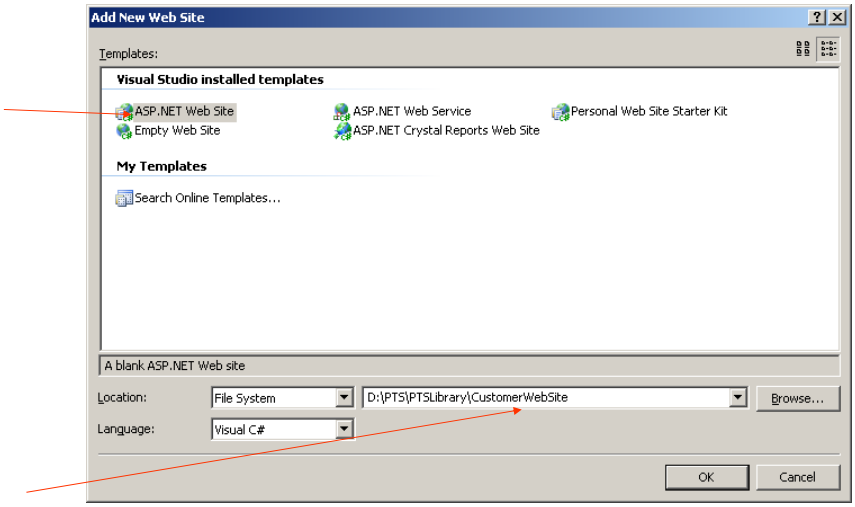
38

## Building the Customer Website

- Now we want to build a website that will allow customers to log in and view the details of the project(s) they commissioned
- The website will be built as an *ASP.NET Website* project (still in the PTSLibrary solution)
- Go to *File -> Add -> New Website…*
- Select the *ASP.NET Web Site* template and name it *CustomerWebSite*

39

## Building the Customer Website /2




40

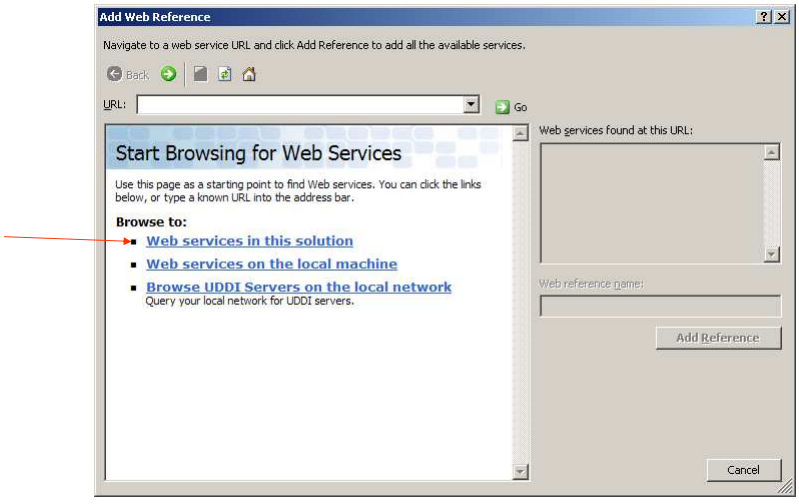
# Adding a Web Reference

- In order to access our newly created web service from the website, it has to be added as a web reference
- Righ-click on the *CustomerWebSite* project in the solution explorer and select *Add Web Reference…*

41

# Adding a Web Reference /2

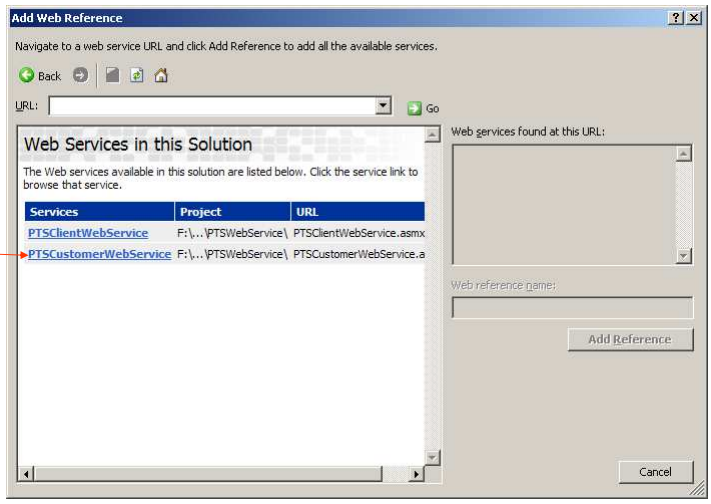
- Click on Web services in this solution




42

# Adding a Web Reference /3

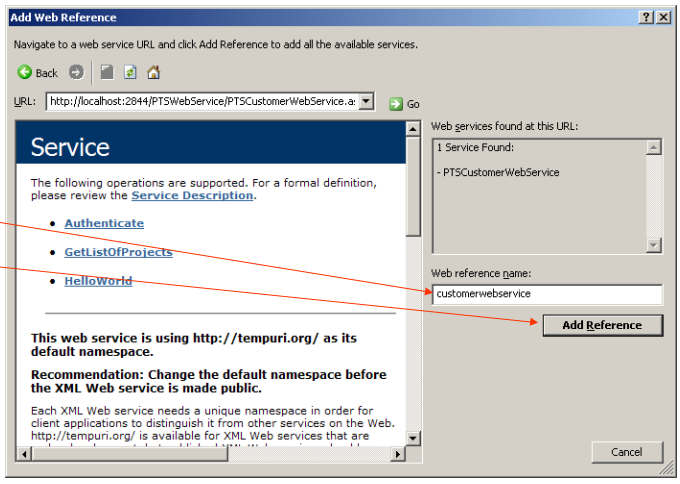
- Select the PTSCustomerWebService:




43

# Adding a Web Reference /4

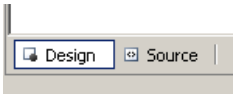
- Once the web service has been found, name the reference *customerwebservice* and click *Add Reference*




44

## Source and Design

- When we created the web site project, a default web form called Default.aspx was created
  - Double-click on this file to open it
- You will notice that you can toggle between Design and Source view




45

## Structure of an ASP.NET Web Form

- An ASP.NET web form actually consists of two files:
  - *XXX*.aspx – this file contains HTML-like markup code, which is used to design the layout of the web form (you can see the code when you are in Source view)
  - *XXX*.aspx.cs – this is the C# code behind the *XXX*.aspx page. It contains the business logic driving the display laid out in the visual design surface and mark-up.
  - Code behind is a direct attempt to separate display from business logic

46

## Design of Default.aspx

- Change to Design view and add the following controls:
  - 2 Labels saying *Username* and *Password*
  - 2 TextBoxes (txtUsername, txtPassword)
  - 1 Button (btnLogin), saying *Login*
  - 1 Label (lblMessage) with text set to nothing
- Have a look at the next slide to see what it should look like

47


## Design of Default.aspx /2

48

# Adding Code

- Double-click on the btnLogin button
  - This will take you to the code-behind file
  - Skeleton code for handling the event will automatically be generated
- Start by declaring an instance of the web service class

```
10
11  public partial class _Default : System.Web.UI.Page
12  {
13      private customerwebservice.Service service;
14
15      protected void Page_Load(object sender, EventArgs e)
16      {
17          service = new customerwebservice.Service();
18      }
```

49

# The btnLogin_Click Method

- This code is run when the user tries to login

Call on the web service

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    int id = service.Authenticate(txtUsername.Text, txtPassword.Text);
    if (id > 0)
    {
        Session["id"] = id;
        Response.Redirect("ProjectDetails.aspx");
    }
    else
    {
        lblMessage.Text = "Incorrect login details, please try again";
    }
}
```

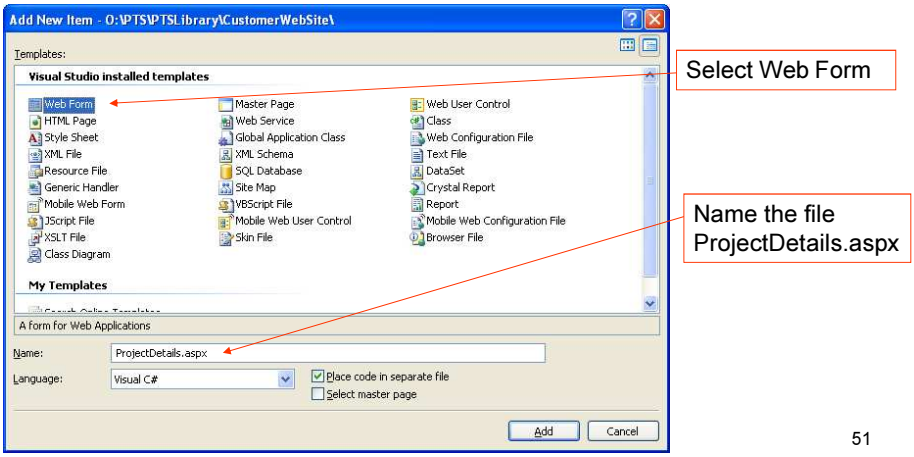
Saves id in session

Successful authentication redirects the user to the ProjectDetails.aspx page

50

# Add ProjectDetails.aspx Web Form

- Right-click on the CustomerWebSite project and select *Add New Item*




51

# ProjectDetails.aspx

- On this web form we will display the name of each of the customer's projects. For each project, we will list all tasks and display the task's status
- To do this we will write a method that obtains all this information, concatenates it as a string and writes it on the page
- Right-click on ProjectDetails.aspx and choose *View Code*, which will open the code-behind file
  - Add the using directive
    - using customerwebservice;

52

# ProjectDetails.aspx /2

```
public void ShowProjectDetails()
{
    if (Session["id"] == null)
    {
        Response.Redirect("Default.aspx");
    }
    else
    {
        Service service = new Service();

        Project[] projects = service.GetListOfProjects((int)Session["id"]);

        for (int i = 0; i < projects.Length; i++)
        {
            Project p = projects[i];
            Response.Write("<p>Project: <b>" + p.Name + "</b><br>");
            Task[] tasks = p.Tasks;
            for (int j = 0; j < tasks.Length; j++)
            {
                Task t = tasks[j];
                Response.Write("Task: <i>" + t.Name + " - " + t.theStatus + "</i><br>");
            }
            Response.Write("</p>");
        }

    }
}
```

If the page is accessed without being logged in, redirect to login page

List each project

List tasks for each project

Response.Write writes to current HTTP output

53

# ProjectDetails.aspx /3

- Double-click on the ProjectDetails.aspx form and change to Source view

```
1  <%@ Page Language="C#" AutoEventWireup="true" CodeF
2
3  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transi
4
5  <html xmlns="http://www.w3.org/1999/xhtml" >
6  <head runat="server">
7      <title>Untitled Page</title>
8  </head>
9  <body>
10     <form id="form1" runat="server">
11     <div>
12         <% ShowProjectDetails(); %>
13     </div>
14     </form>
15 </body>
16 </html>
17
```

This is all the code that needs to be added. Just a call to our method

54

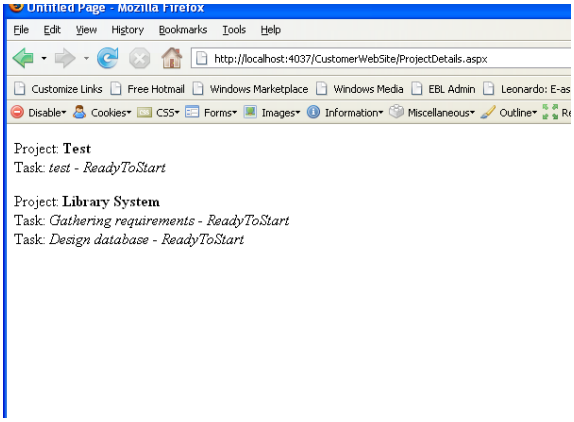
# Testing the Customer Website

- Right-click on ProjectDetails.aspx and select *View in Browser*
- If everything goes well then you should be redirected to *Default.aspx*
- Enter wrong login details and check that an error message is displayed
- Then enter valid login details for a customer you entered in the database
  - Make sure that there is a project created for this customer and that there are tasks created for the project

55

# Testing the Customer Website /2

- If all goes well, you should see something like this:




56

## Summary

- In this session we have looked at what web services are and how they work
- We built two web services to be used by the customer website and the Java client
- No-argument constructors were introduced in the business classes of the PTSLibrary
- A customer website was created
- In the next session we will create a Java client that uses the client web service

57