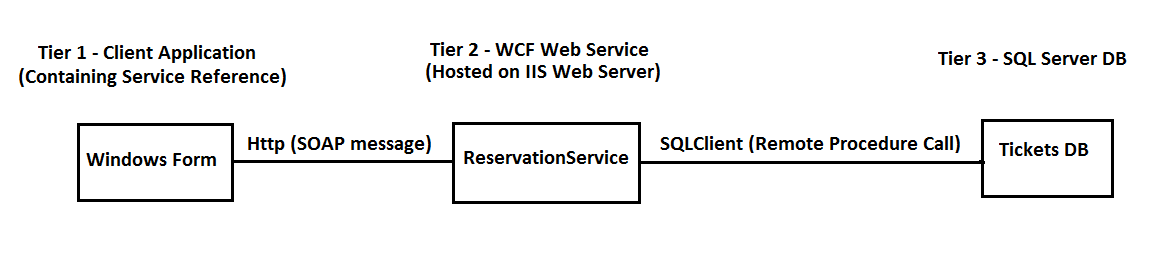
**IT7307: Creating an N-Tier Distributed System**

**Introduction**

In this tutorial we will be creating an N-Tier distributed system that will connect a Windows Forms client to a WCF Web Service, which will in turn invoke a stored procedure running on a SQL Server database. Although the functionality provided by this distributed system will be basic, its architecture and the middleware communication protocols it uses are quite sophisticated. The Windows Form client uses a Service Reference to access the functionality provided by the Web Service. Communication between this client and Web Service is implemented using SOAP (Simple Open Access Protocol). SOAP is a standardised XML-based protocol describing how requests and responses are handled between the client and web server. These SOAP messages are encapulated within the HTTP protocol, which allows the messages to pass unimpeded through firewall security. The Web Service processes these messages by invoking methods that implement the functionality of the Web Service. In our system, these methods will contain Remote Procedures Calls (RPC) that invoke stored procedures running on a SQL Server database. The communication between the Web Service and SQL Server will be implemented using SQLClient, a middleware component that allows machines on a network to connect to a remote SQL Server database and invoke its stored procedures. The components of our distrubuted system and the means by which they communicate are illustrated below.

**Figure 1**



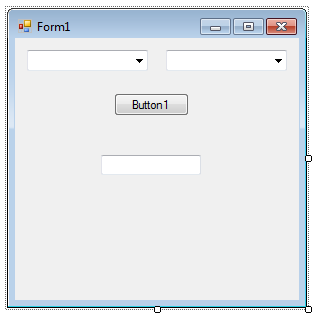
**Approach**

Our web service needs to connect to a SQL Server Database and execute a stored procedure that has previously been created. Rather than developing this functionality within a web service directly, a good first step is to create a simple prototype. The prototype will involve a VB forms application that will use the SQLClient protocol to connect to SQL Server and invoke a stored procedure running in a database called TICKETS. This prototyping approach allows us to focus on functional requirements without having to worry about the additional complexity of the web service itself. Once we have a working application in this environment, we can simply copy and paste our code into our web service application.

The form in figure 2 will be used to supply values to two input parameters that are required to run a stored procedure called ‘Reserve’, previously created in the TICKETS database. The top left combo box has been named **seatType** and the combo box to the right has been named **classType**. Using the **Items** property of these controls, the values **‘Window’**, **‘Middle’** and **‘Aisle’** are added to the seatType combo box. The values **‘Economy’** and **‘First’** are added to the Items property of the classType combo box.

When the user runs the application, they simply select the desired value from each combo box and click the the button named ‘Button 1’. The code attached to the click event of this button (described in Step 2, p4) connects to the TICKETS database and calls the Reserve stored procedure (described in Step 1, p3), passing the values selected in the combo boxes as parameters. The textbox (named **TextBox1**) below the button control is simply used to display the value returned by the stored procedure, which indicates whether or not the desired seat has been booked. If the value ‘True’ is displayed, this indicates that a seat is available and has been booked. If the value ‘False’ is displayed, this indicates that no seat is available for the selected class and seat type.

**Figure 2**

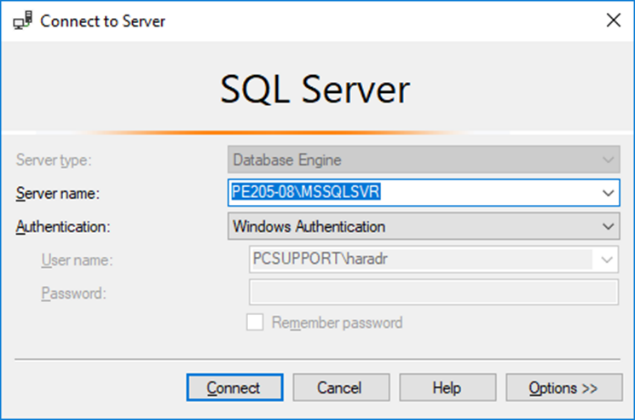


**Step 1 – Configuring SQL Server**

The first step is to attach a database called TICKETS to SQL Server. The database can be found in the WCF Web Services section on Moodle. Download the database and copy it to the following folder:

C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSVR\MSSQL\DATA

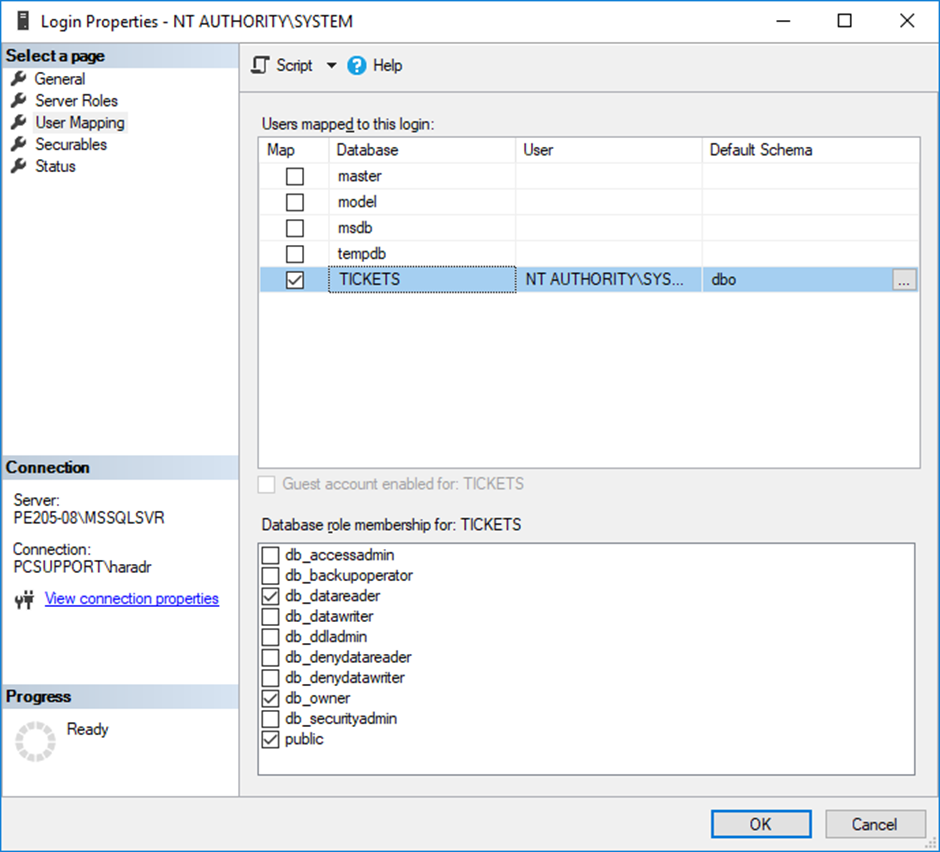
Start SQL Server, ensuring that you connect to the MSSQLSVR instance, rather than SQLEXPRESS, as shown below.



In the **Object Explorer** panel on the left, right-click on the **Database** node and select **Attach** from the short-cut menu. In the dialog box that appears, click on the **Add** button and select the TICKETS database. If a log file for the database is listed in the bottom panel, select it and click on the **Remove** button. In the **Attach As** column in the top panel, delete the path details and extension for the database so that only the name **TICKETS** is left in this column entry. Now click on OK button to attach the database.

The next step is to create a stored procedure called **Reserve.** When this procedure is executed, it will modify the Seats table in the TICKETS database to record that a seat has been reserved by the user. You will find the code to create this stored procedure on Moodle. Click on the link and copy the code. In SQL Server, click on the **New Query** button and paste the code into the query window. Press **F5** to execute the query. I will demonstrate how we can execute the stored procedure we have just created to ensure that it is functioning correctly. The stored procedure simple counts the number of seats that meet the criteria specified in the WHERE clause of the first SELECT statement i.e. the desired type and class of seat requested by the user, as defined by the input parameters seatType and classType. If the variable @Count is non-zero, at least one seat meets the specified criteria. In which case the primary key (named Number), of the first row that meets this criteria has the column ‘Taken’ set to 1 – indicating that this seat is now been reserved. Finally, the output parameter ‘@Response’ is set to 1 and this value is returned to the program that invoked this stored procedure. If no seat meets the criteria, the database is not updated and the stored procedure returns a value of 0.

Because we will be accessing this database from an external program, we need to adjust the security settings to prevent authentication problems when our application connects to SQL Server to execute the stored procedure we have just created. Expand the **Security** node in the Object Explorer panel (located below the Databases node). Now expand **Logins** node and double-click on the login called **NT AUTHORITY\SYSTEM.** This will cause the Login Properties dialog box to appear, as shown below.

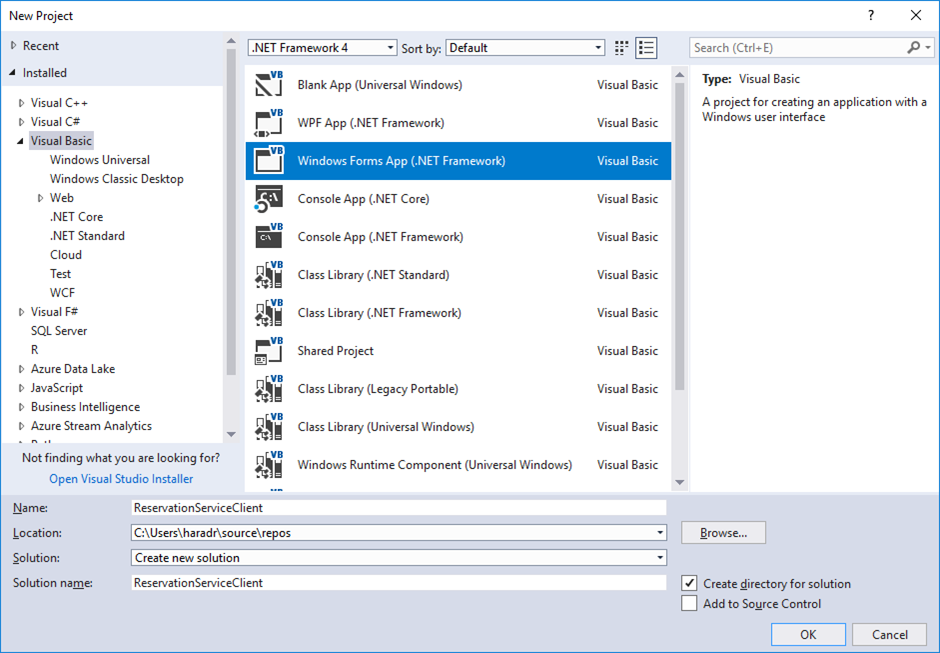


In the **Select a page** panel click on the **User Mappings** page and locate the TICKETS database. Place a tick in the **Map** column adjacent the TICKETS database row, as shown above. In the lower panel, place a tick in **db\_reader** and **db\_owner**, as shown above. Finally, click on the OK button. Performing these steps informs SQL Server that we will be using our network credentials (the username and password we provide when first logging on to a lab machine) to authenticate the execution of the stored procedure we created earlier. This will prevent authentication problems when we run our application.

**Step 2 – Create VB Form Prototype Application**

Now that we have set up our dataset, we will create a simple prototype in Visual Studio that will use the SQLClient namespace of the .NET framework to connect to SQL Server and execute the stored procedure we created earlier. We will use this prototype to define the functionality required by our Web Service. Once working correctly, we will be able to use the code in our Web Service.

Start Visual Studio and create a new project with the properties shown below. Ensure that you select a **Windows Form App** from the Visual Basic templates and **.NET Framework 4**, as shown below. Name the application **ReservationServiceClient**. You can use the default location details as the destination for this application.



Add the controls to the form, as shown in figure 2, ensuring that you use the same names for them as specified in the Approach section of these notes. You should also populate the combo boxes with the values specified in that section. In Moodle, click on the link called ‘Code used in Reservation Prototype’ and copy the code. Double-click on the form Button and paste the code previously copied.

In this code, **SqlConnection** is an object that connects to our SQL Server database. The **Data Source** property specifies the name of the machine we wish to connect to (i.e. the machine on which our database exists). **You will need to change this to the name of your machine**. The **SqlCommand** object is used to specify the name of the stored procedure we wish to invoke from our VB forms application – ‘Reserve’ in our case. The **SqlParameter** object is used to define any parameters expected by the stored procedure. In our case, we have 2 input parameters (@seatType and @classType) and 1 output parameter (@Response). The names and data types of all these parameters must match the names and data types specified in our stored procedure. The **ParameterDirection** property of the two input parameters (@seatType and @classType) has been set to Input. The output parameter ‘@Response’ has its direction set to Output.

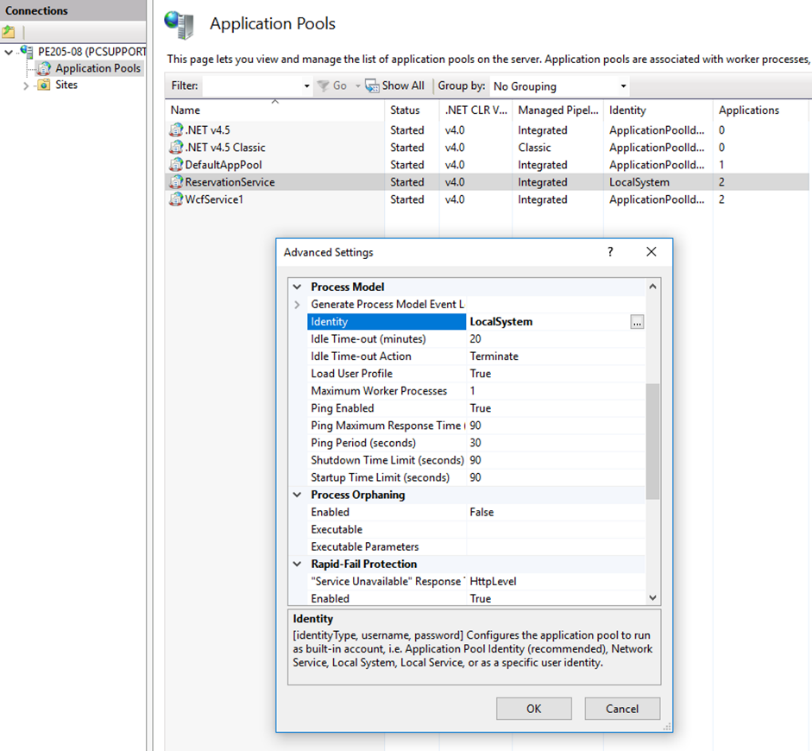
Once these details have been specified, we are ready to open a connection to our TICKETS database using the ‘**objConnection.Open()**’ command and to execute our query using the command ‘**objCommand.ExecuteNonQuery()**’. After execution, the output parameter’s value property is populated with the value returned by our stored procedure (either a 1 or a 0). This value is converted to a string and displayed in the textbox control in the form.

Once we have ensured that the code executes as expected, we can copy and paste it into a web service application. With a minor modification, the code can be used to implement the desired functionality in the web service.

**Step 3 – Configuring IIS**

Now that we have a working prototype, we are ready to create our web service. The first step is to create a web site in IIS to host our WCF Web Service. To begin with, create a folder in the **inetpub** folder of the C: Windows drive that we will use to contain the files for our web service. We will call this folder **ReservationService.** It’s always a good idea to check that the web site we create in IIS has been defined correctly, so copy the htm and png files within the **wwwroot** subfolder of inetpub and then paste them into the new folder we have just created. Now fire up IIS Management Console and stop all the other web sites – remember IIS can only run one web site at a time. In the **Connections** panel in IIS, right-click on the **Sites** node and create a new web site called **ReservationService**. The final step in configuring IIS is to change a setting for the Application Pool that contains our web site to allow our web service to interact with a SQL Server database. Do this by clicking on the **Application Pools** node in the Connections panel and select the application pool called ReservationService. Right-click on this application pool and select **Advanced Settings** from the short-cut menu. Scroll down the setting **Identity** in the **Process Model** section (see figure 3 overleaf) and change this setting from ApplicationPoolID to **LocalSystem**. To test that the basic functionality of our web site has been defined correctly, click on the new web site in the Connections panel and then browse to it by clicking on **Browse \*:80 (http)** in the **Actions** panel. If all is well, the IIS Splash form will appear in Internet Explorer.

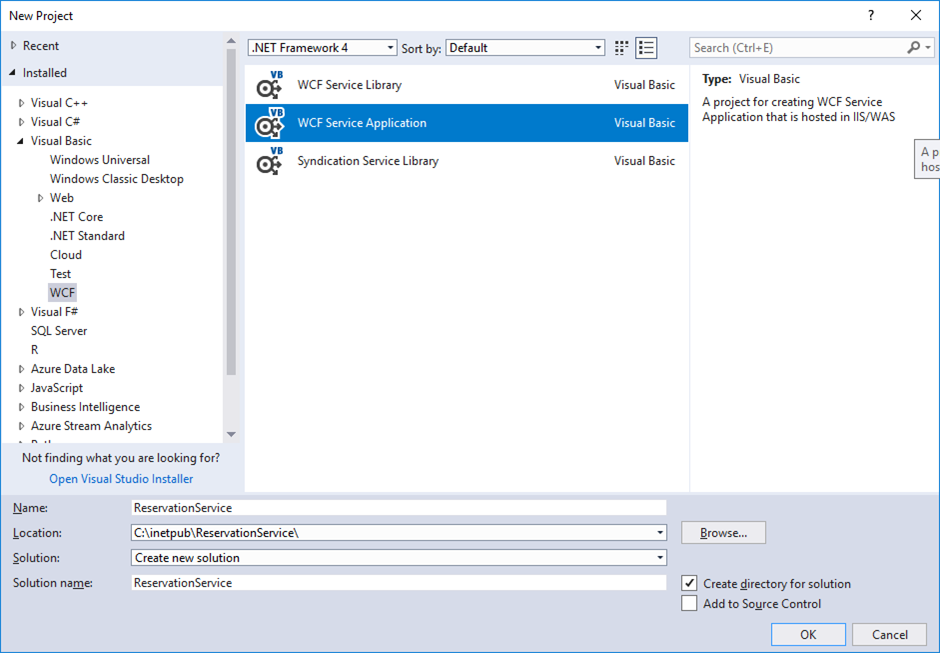
**Figure 3**



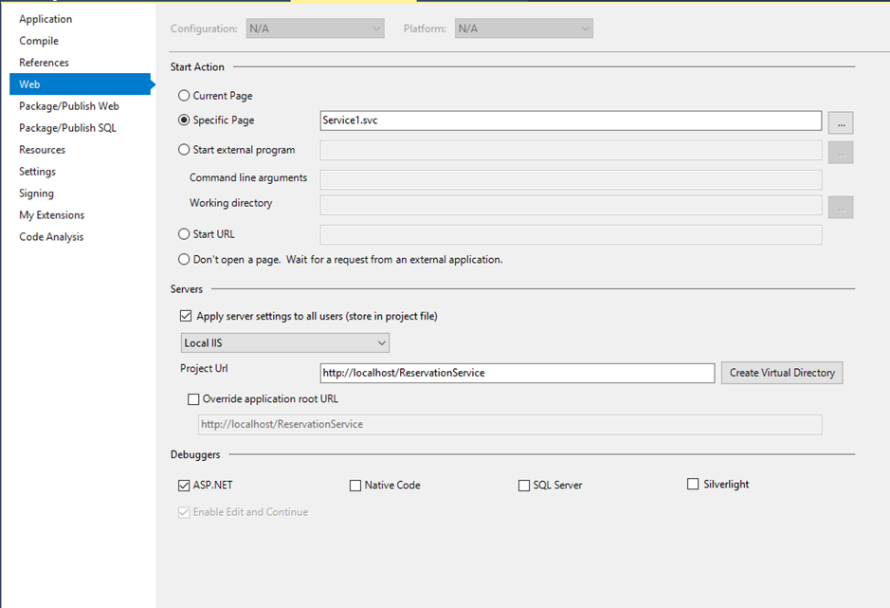
**Step 4 – Creating a Web Service**

In Step 2, we created a simple prototype using the Visual Basic Window Forms App template to ensure that we can access and modify the Tickets database in SQL Server using the SQLClient communication protocol. This approach allows us to debug any problems we may have encountered, without the additional complexity of a web service to worry about. In other words, if a problem had occurred it may have been difficult to determine whether the problem was related to our SQLClient code or some configuration problem associated with the web service itself. However, now that we know that our prototype functions correctly, we can use the code attached to the button control to allow our web service to access and modify our Ticket database. The code will require a minor modification, which I will highlight in my demonstration.

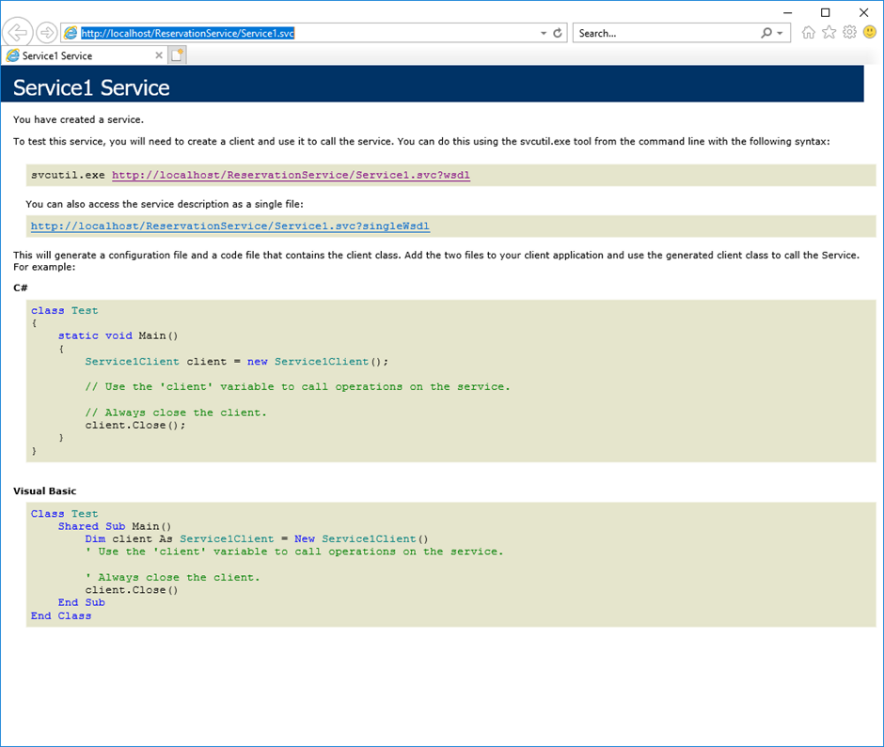
To create web-based applications, we must run Visual Studio as an Administrator. Re-start Visual Studio if necessary and specify the properties below when you create a new project. From the **Visual Basic** templates, select **WCF** from the Web templates and **WCF Service Application.** Ensure that **.NET Framework 4** has been selected and name the project **ReservationService.** Use the Browse button to set the location for the project in **C:\inetpub\ReservationService** – the folder we created in Step 3.

****

Once the project has been created based on the details we specified above, we need to adjust the properties of our application so that it will be hosted by IIS. From the **Project** menu, select **ReservationService Properties** and click of the Web option, as shown below. We need to change the **Start Action** property to **Specific Page** and use the ellipses button to specify **Service1.svc** as the page to start. For the **Service** properties, select Local IIS from the combo box and then click on the **Create Virtual Directory** button on the right. It is important to ensure that the Project URL does not contain any port number details – i.e. it should be specified as <http://localhost/ReservationService>, as shown below. It may be necessary to Build the application or even re-load the project before the project URL is specified in this format.



Once these details have been defined, set the Service1.svc file listed in the Solution Explorer panel as the start page. Right-click on this file and select **View in Browser (Internet Explorer)** option from the short-cut menu. It may be necessary to set Internet Explorer as the default browser. If all is well, you should see the web service displayed, as shown below. Ensure that the project URL is the same as that seen in this screen shot.

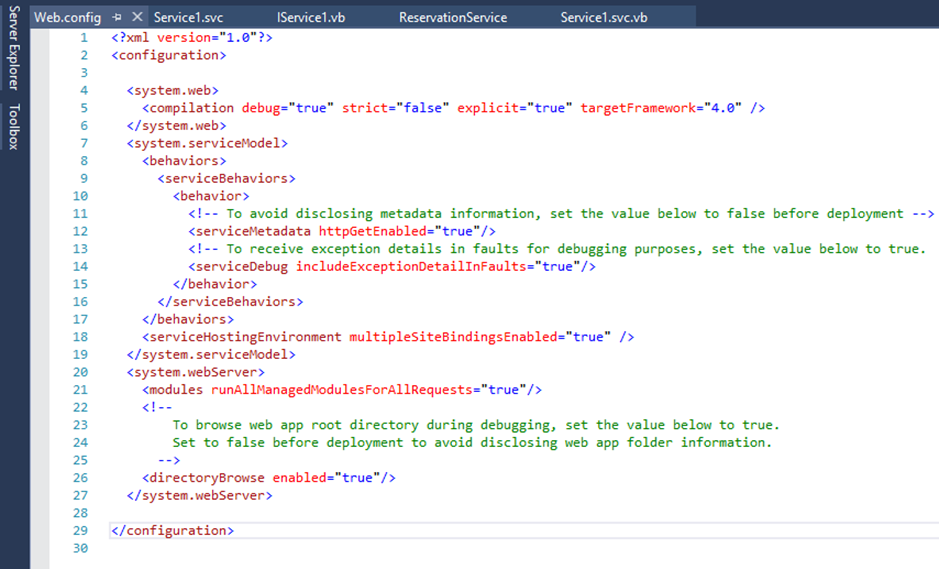


Now that we know the basic functionality of our web service has been defined correctly, we need to specify the actual functionality that our application will provide. Using the link in Moodle called **‘Code used in Reservation Web Service’**, copy the code and paste over the existing code listed after right-clicking on Service1.svc and selecting **View Code** from the short-cut menu. You will need to modify this code so that the Data Source details specified for the object **objConnection** matches the **Server name** displayed when you first connect to SQL Server Management Studio.

We now need to modify the public interface for the web service so that it reflects the details we specified in the Service1.svc file. Double-click on the file IService1.vb in the Solution Explorer panel and modify the code so that the only function listed below the <OperationContract()> is as follows:

Function Reserve(ByVal seatType As String, ByVal classType As String) As Boolean

The final step in the creation of our web service is to adjust the **Web.config** file listed in Solution Explorer so that meaningful error messages are generated when we test our application. Double-click on the Web.config file and change the value for **includeExceptionDetailInFaults** from false to true, as shown below.



Build and save the application and then select View in Browser (Internet Explorer) after right-clicking in the Service1.svc file. The web service should display in Internet Explorer as before. Copy the URL of the web service so that we can use it when we add a service reference to our client application in Step 5. Keep this web site open – closing it shuts down the web service.

**Step 5 – Modifying the Prototype Client**

Now that we have our web service up and running, we need to re-visit our client prototype form so that is uses our new web service. The first task is to add a service reference to the project so that the client application can access the functionality with have defined in Step 4. In the **Project** menu select the **Add Service Reference** option. In the **Address** text box in the dialog box that now appears, paste the URL of the web service we copied at the end of Step 4. Click on the **Go** button to retrieve details of the functionality provided by our web service. The Reserve function should be displayed.

On the form, remove the Textbox1 control – we will use a message box control to provide feedback to the user. Double-click in the button control and overwrite all of the existing code with the code found on the link called ‘**Code for a** **Form Based We Service Client’** on Moodle. Now save and build the project. Finally, start the application and attempt to reserve seats using the combo box controls on the client form. A message box will appear to inform you as to whether your reservation was successful. As a final check, refresh the **Database** node in SQL Server and examine the Seats table to ensure that your reservation was recorded.