Distributed Computing

Tutorial 6, Marks: 30,
Due date: 15 Oct 23:59 AWST

What We're Doing Today

1. Replacing your business and tier (from Tutorial 4 and 5) with a Web Service
2. Replacing your Data Tier with a Web Service too!

An Intro to MVC

MVC is a RESTful system that uses a Model View Controller pattern to manage the website (thus the name). For those of you not familiar with an MVC model, it has three major parts:

- Controllers: Control actions and behaviours in the system (program logic)
- Models: Represent entities in the system (data objects representing things)
- Views: Ways of presenting the information (in this case web output)

Controllers control how Models are viewed through Views.

In ASP.NET MVC, Controllers define a service interface and the actions you can perform, Models represent the information present in the system (and in our case, the data tier), and Views represent templates for web pages that the system can then create based on what you ask it to do.

Obviously… we don't care about web pages! So for this tutorial we're not going to bother with them. We're going to build an API! .

Anything you return from the API controller methods will be converted in the backend into JSON as best as .NET can manage it.

Please Follow Lecture 6 - WebAPI which is your guide to build a simple WEBAPI. I have also added the source codes for your convenience. It will help you to create your business tier web service.

In tutorial four and five, you already have data servers where we can search a Bank database. We have a client that connects to a WCF business tier and the business server connects with the data server and get the result back to the Client.

First replace the WCF data server to WEB API Data Server !!
Then, we replace the WCF Business tier to a WEB API Business tier. And the client will call REST APIs, so we will update the client as well.

Note, from the visual output's perspective, it is exactly same like Tutorial 2, just different technologies.

Quick Interlude: Making data classes

So you want to send stuff across the internet using JSON. There is a really dumb method in which you encode and decode the JSON manually yourself, but nobody wants to do that. Nobody wants to do that so much for that matter that most API's for web services on the internet already have Python, C#, and Java interfaces pre built for integration with the web service.

We're going to build one of those for C#. Sort of anyway.

Go and create a new Class Library project for the .NET Core framework. Call it something like API Classes, or Biz-GUI Classes. These are going to be the go-betweens for our data.

Create two classes, I've called mine DataIntermed and SearchData. These are simply data constructs, all public, no functions. For me, DataIntermed looks like this

```
public class DataIntermed
    {
        public int bal;
        public uint acct;
        public uint pin;
        public string fname;
        public string lname;
    }
```

And SearchData looks like this

```
 public class SearchData
    {
        public string searchStr;
    }
```

These objects represent the format of the data we're going to be passing through from the Biz tier to the GUI, so if you have profile pictures implemented, make sure you've added those too!

Basically what we're going to do is use these objects as templates for the .NET JSON serializers, so we don't have to do any work with the JSON itself. Makes it easy for us!

Back to Data and Business

In your API Controller out on your business tier web service, modify the default get request so it returns the number of entries in the Data Tier. Use Restsharp and newtonsoft.json library to make a request from Business WebAPI to Data WebAPI. Attached sourced code has an example.

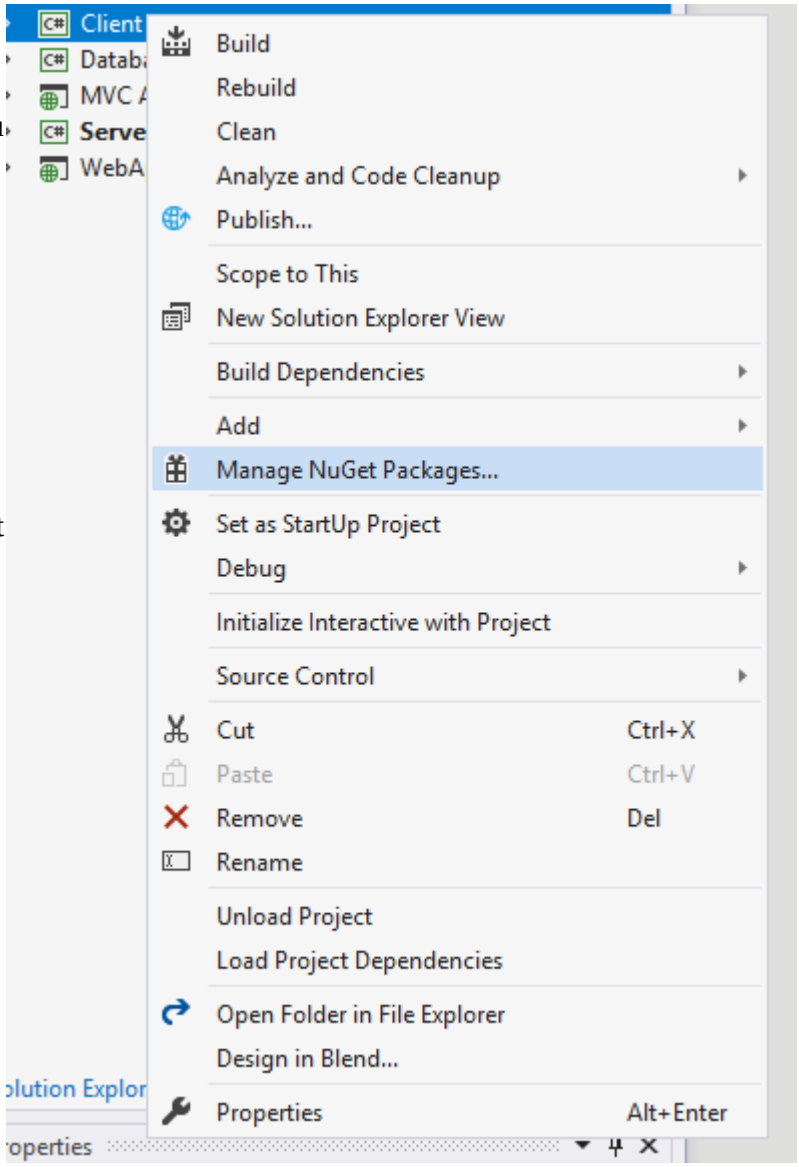
Then, add another API Controller (maybe name it "GetValuesController" or "GetAllController"), and remove all functions but the GET request that takes in a value. This will be the index value, and you can use it to return the values from the Data Tier. Use your new Data Intermediary object to return the data (just return a DataIntermed object populated with the record you want to send to the GUI).

Next, add *another* API Controller (call it "SearchController"), and remove everything but the post function. Make sure it takes in an object of type SearchData, and returns an object of type DataIntermed. You'll note you don't have to do anything special, as the MVC backend will convert these objects to JSON for you.

At this point…. You should be done. Try starting up the Business Web Service tier alongside the Data tier and go to /api/values/2. If you get a json object representing the second person in the database, congrats! You have a functioning web service!

GUI Time

Right click on your GUI and go to Manage NuGet Packages. NuGet is kind of like Python's Pip package manager, it allows you to borrow entire libraries of functionality from other developers on the web and add them to your project.



If you build any really cool code, you can put it up on NuGet too! It's a big community project, and it's really useful.

Click on the browse tab and search for "RestSharp". Install it. RestSharp is a framework for making RESTful interface requests from inside an application. This greatly reduces the work we're going to have to do to get the GUI working with the new Web Service Business Tier.

You'll also want to find the "Newtonsoft.Json" addon. This is the gold standard of JSON integration libraries for C#, and it's what we're going to use for this unit.

Once these are both installed, pop on over to your GUI's main window, and add `using Newtonsoft.Json;` and `using RestSharp;` to your used namespaces at the top of your C# code.

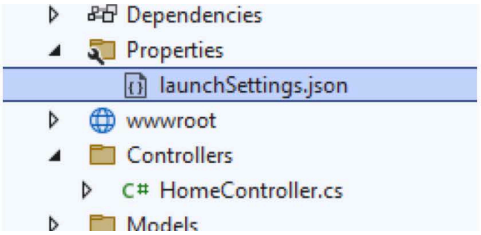
Remove your .NET remoting code from your GUI code. What you'll want to do instead is something like the following:

```
URL = "your base URI here";

client = new RestClient(URL);
RestRequest request = new RestRequest("api/values");

IRestResponse numOfThings = client.Get(request);
TotalNum.Text = numOfThings.Content;
```

This is a *lot* simpler than the .NET remoting system, as you're basically using the .NET web engine backend to just make web requests.

To get your Base URI for the Web Service, click on the launchSettings.json. The Base URI *should* be the application URL in the profiles of the project.

```
"profiles": {
  "SimpleWebAPI": {
    "commandName": "Project",
    "dotnetRunMessages": true,
    "launchBrowser": true,
    "applicationUrl": "http://localhost:5076",
    "environmentVariables": {
      "ASPNETCORE_ENVIRONMENT": "Development"
    }
  },
```



Next, you need to change the code for your button that retreives info for a given index. You're going to use the DataIntermed class, so make sure a reference to it exists for your GUI.

Next, you'll want to do something like the following:

```
        private void GoButton_Click(object sender, RoutedEventArgs e)
        {

            //On click, Get the index....
            int index = Int32.Parse(IndexNum.Text);
            //Then, set up and call the API method...
            RestRequest request = new RestRequest("api/getall/" + index.ToString());
            IRestResponse resp = client.Get(request);
            //And now use the JSON Deserializer to deseralize our object back to the class
we want
            API_Classes.DataIntermed dataIntermed =
JsonConvert.DeserializeObject<API_Classes.DataIntermed>(resp.Content);
            //And now, set the values in the GUI!
            FNameBox.Text = dataIntermed.fname;
            LNameBox.Text = dataIntermed.lname;
            BalanceBox.Text = dataIntermed.bal.ToString("C");
            AcctNoBox.Text = dataIntermed.acct.ToString();
            PinBox.Text = dataIntermed.pin.ToString("D4");
        }
```

Make sure you've matched the Web service URI to the name of the controller you set earlier.

Next, change your search button function to something similar to the following:

```
        private void Searchbut_Click(object sender, RoutedEventArgs e)
        {
            //Make a search class
            API_Classes.SearchData mySearch = new API_Classes.SearchData();
            mySearch.searchStr = Searchyboi.Text;
            //Build a request with the json in the body
            RestRequest request = new RestRequest("api/search/");
            request.AddJsonBody(mySearch);
            //Do the request
            IRestResponse resp = client.Post(request);
            //Deserialize the result
            API_Classes.DataIntermed dataIntermed =
JsonConvert.DeserializeObject<API_Classes.DataIntermed>(resp.Content);
            //aaaaand input the data
            FNameBox.Text = dataIntermed.fname;
            LNameBox.Text = dataIntermed.lname;
            BalanceBox.Text = dataIntermed.bal.ToString("C");
            AcctNoBox.Text = dataIntermed.acct.ToString();
            PinBox.Text = dataIntermed.pin.ToString("D4");
        }
```

Now you should be able to just start all three parts up, and…. It should work! Congrats! You've built a web service! Wasn't that so much nicer than .NET remoting?

The marks for Business and Data Web API project is 10. The marks for GUI rest client is 5.

We're not done yet, here's some stuff for you to play around with:

- First up, I've not implemented threading with these API requests… you may want to do that :) [5 Marks]
- Secondly, try to send image from the business Web API over http [5 marks]

[Total 5 marks for next 2 tasks]

- Thirdly, Exceptions! They need to work somehow. Fun fact, you *can* serialize exceptions! I'll let you figure out how to conditionally deserialize them.
- If you've implemented your Data Tier like I did, every new Data Tier connection created at the Business Tier will recreate the Database….. this isn't good. You'll need to make the Data Tier a singleton. Here's a hint (there's a lot more to this, but I'll let you figure it out):

```
//Bind server to the **instance** of DataServer

DataServer bigServer = new DataServer();
host = new ServiceHost(bigServer);
```

And as always, if you get stuck, find your tutor!