Restful Web Services through ASP.Net Web API 2

John F. Gnazzo

Introduction

This Blog will discuss ASP.Net Web API 2, and the architecture and details of using it to work with data.

ASP.Net Web API 2

ASP.NET Web API 2 is a framework for building Restful web services that can be accessed through the internet using the HyperText Transfer Protocol (HTTP). Client applications such as browsers, mobile and desktop devices can consume and upload data through these services using standard HTTP Methods including GET, PUT, POST, and DELETE.

ASP.Net Web API 2 is a Microsoft server side technology. However, any client technology that can perform web requests and can consume data through the Web API, may be used.

The Web API 2 web services may be hosted in Internet Information Services (IIS), or self-hosted as a service or console application, typically under the Windows Server operating system.

The Web API 2 web services are an alternative, and a newer technology, than web services created using the Microsoft Windows Communication Framework (WCF). Web API 2 is the preferred technology to use when creating HTTP-based services that need to be accessible from the widest variety of clients.

Architecture

The architecture for a complete application includes the Server side components, and exposed Restful services, and clients.

Server Side Components

Server side components of interest include the Data Access Layer, Controllers and the WebApiConfig.cs, file.

Data Access Layer

The source of the data for the application is typically a relational database, but could be anything from a document database, to data stored in an XML or JSON formatted file.

For simplicity purposes, the Entity Frameworks architecture is used to map data base tables and stored procedures through a **dbContext** object.

The **dbContext** object is typically setup by the creation of an **ADO.Net Entity Data Model**.

The following are the model classes used for the examples below:

BusinessInfo Class

```
[Serializable]
[DataContract(Name = "BusinessInformation", Namespace = "WebAPIs.Controllers")]
public class BusinessInfo
{
  [DataMember]
  List<BusinessInformation_Result> BusinessInformation;
```

```
    public UserInfo(List<BusinessInformation_Result> result)
    {
      BusinessInformation = result;
    }
  }
```

BusinessInformation_Result Class

```
  public partial class BusinessInformation_Result
  {
    public int UserID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Street { get; set; }
    public string City { get; set; }
    public string Phone { get; set; }
    public int OrderID { get; set; }
    public int ItemID { get; set; }
    public string ItemName { get; set; }
    public decimal BaseCost { get; set; }
  }
```

Controllers

The access to the Web API services, are through Controllers, distinct modules of code providing methods mapping to Restful web requests. Controllers are usually mapped to one or more tables and/or related stored procedures. Each controller usually contains the following elements:

- Creation of a database context.
- Get method to select all objects of a certain type
- Get method to select a specific object
- Put method to update data
- Post method to add data
- Delete method to delete data
- 0 or more custom methods to call stored procedures.

An example controller is shown below:

```
public class BusinessDataController : ApiController
  {
    // creation of database context
    private readonly BusinessEntities _db = new BusinessEntities ();

    // Get Method to select all objects of a certain type
    public IEnumerable<BusinessData> GetBusinessDatas()
    {
      return _db.BusinessDatas.ToList();
    }

    // Get Method to select a specific object
    [ResponseType(typeof(BusinessData))]
    public IHttpActionResult GetBusinessData(int id)
    {
      BusinessData BusinessData = _db.BusinessDatas.Find(id);
      if (BusinessData == null)
      {
```

```
        return NotFound();
      }

      return Ok(BusinessData);
    }

    // Put Method to update data in database
    public IHttpActionResult PutBusinessData(int id, BusinessData BusinessData)
    {
      if (!ModelState.IsValid)
      {
        return BadRequest(ModelState);
      }

      if (id != BusinessData.UserId)
      {
        return BadRequest();
      }

      _db.Entry(BusinessData).State = EntityState.Modified;

      try
      {
        _db.SaveChanges();
      }
      catch (DbUpdateConcurrencyException)
      {
        if (!BusinessDataExists(id))
        {
          return NotFound();
        }
        throw;
      }

      return StatusCode(HttpStatusCode.NoContent);
    }

    // Post Method to add data to database
    [ResponseType(typeof(BusinessData))]
    public IHttpActionResult PostBusinessData(BusinessData BusinessData)
    {
      if (!ModelState.IsValid)
      {
        return BadRequest(ModelState);
      }

      _db.BusinessDatas.Add(BusinessData);
      _db.SaveChanges();

      return CreatedAtRoute("DefaultApi", new { id = BusinessData.UserId },
BusinessData);
    }

    // Delete Method to delete data from database
    [ResponseType(typeof(BusinessData))]
    public IHttpActionResult DeleteBusinessData(int id)
    {
      BusinessData BusinessData = _db.BusinessDatas.Find(id);
```

```
      if (BusinessData == null)
      {
        return NotFound();
      }

      _db.BusinessDatas.Remove(BusinessData);
      _db.SaveChanges();

      return Ok(BusinessData);
    }

    // Method which calls GetUserInformation stored procedure
    [ActionName("GetUserInformation")]
    [HttpGet]
    public IHttpActionResult GetUserInformation(int userId)
    {
      var userInfo = _db.GetUserInformation(userId);

      if (userInfo == null)
      {
        return NotFound();
      }

      var userInformation = new UserInfo(userInfo.ToList());

      return Ok(userInformation);
    }

    private bool BusinessDataExists(int id)
    {
      return _db.BusinessDatas.Count(e => e.UserId == id) > 0;
    }
  }
```

WebApiConfig.cs

The WebAPIConfig.cs file is used to map routes, i.e. web requests to the framework, and to setup allowable output formats, and consists of the following sections:

- Web API configuration
- Standard Routes
- Custom Routes for action methods
- Setting up allowable output formats

An example WebApiConfig.cs file is shown below:

```
  public static class WebApiConfig
  {
    public static void Register(HttpConfiguration config)
    {
      // Web API configuration and services

      // Web API routes
      config.MapHttpAttributeRoutes();

      // Standard Routes
       config.Routes.MapHttpRoute(
          name: "DefaultApi",
```

```
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );

        // Custom Routes for action methods
        config.Routes.MapHttpRoute(
          name: "Action",
          routeTemplate: "api/{controller}/{action}/{userid}"
        );

        // Setting up allowable output format(s)
         var json = config.Formatters.JsonFormatter;
                    json.SerializerSettings.PreserveReferencesHandling =
                Newtonsoft.Json.PreserveReferencesHandling.None;

        config.Formatters.Remove(config.Formatters.XmlFormatter);

    }
  }
```

Consuming Data at the client

The following code is used to make each request described in the Controllers section above. The following code is written in C#, but other technologies such as Java can be used.

```
using (var client = new HttpClient())
        {
          client.BaseAddress = new Uri("http://localhost/WebAPIs/");

          // HTTP Get all Business Data
          HttpResponseMessage response = await client.GetAsync("api/BusinessData");
          if (response.IsSuccessStatusCode)
          {
                List<BusinessData> BusinessDatas =
                response.Content.ReadAsAsync<IEnumerable< BusinessData
                >>().Result.ToList();
          }

          // GET a business data with UserID of 7
          response = await client.GetAsync("api/ BusinessData /7");
          if (response.IsSuccessStatusCode)
          {
            var BusinessData = await response.Content.ReadAsAsync<BusinessData>();
            System.Console.WriteLine("{0}\t{1}\t{2}\t{3}\t{4}",
BusinessData.BusinessDataID, BusinessData.UserID, BusinessData.CreatedDate,
BusinessData.BusinessDataDetails, BusinessData.UserAddress);
          }

          // HTTP POST - Add BusinessData
          var BusinessDataPost = new BusinessData { UserID = 9,CreatedDate= DateTime.Now };
          response = await client.PostAsJsonAsync("api/BusinessData", BusinessDataPost);

          if (response.IsSuccessStatusCode)
          {
            var url = response.Headers.Location;

            // HTTP PUT - Modify BusinessData
            BusinessDataPost.CreatedDate = DateTime.Now.AddHours(-1);   // Update date
            response = await client.PutAsJsonAsync(url, BusinessDataPost);
```

```
    // HTTP DELETE - Delete BusinessData
    response = await client.DeleteAsync(url);
  }

// HTTP GET – Calling the GeBusinessInfomation Action Method.

HttpResponseMessage response = await client.GetAsync
("api/BusinessData/GetBusinessInformation/9");
  if (response.IsSuccessStatusCode)
  {
    var businessStuff = response.Content.ReadAsAsync<BusinessInfo>().Result;
  }
```

The configuration of the allowable web request output format is discussed in the WebApiConfig.cs section above. In this example, all data will be returned in JSON format.

Below is an example of the JSON output from calling the GetBusinessInformation Action method above.

{"BusinessInformation":[{"UserID":9,"FirstName":"paul","LastName":"casey","Street":"Test1","City":"Test City1","Phone":"952-239-0000","OrderID":7,"ItemID":1,"ItemName":"Meat Lovers Pizza","BaseCost":12.0000},{"UserID":9,"FirstName":"paul","LastName":"casey","Street":"Test1","City":"Test City1","Phone":"952-239-0000","OrderID":8,"ItemID":5,"ItemName":"Cheesy Bread","BaseCost":8.9900},{"UserID":9,"FirstName":"paul","LastName":"casey","Street":"Test1","City":"Test City1","Phone":"952-239-0000","OrderID":9,"ItemID":6,"ItemName":"Hot Wings","BaseCost":8.9900},{"UserID":9,"FirstName":"paul","LastName":"casey","Street":"Test1","City":"Test City1","Phone":"952-239-0000","OrderID":9,"ItemID":5,"ItemName":"Cheesy Bread","BaseCost":8.9900},{"UserID":9,"FirstName":"paul","LastName":"casey","Street":"Test1","City":"Test City1","Phone":"952-239-0000","OrderID":11,"ItemID":5,"ItemName":"Cheesy Bread","BaseCost":8.9900},{"UserID":9,"FirstName":"paul","LastName":"casey","Street":"Test1","City":"Test City1","Phone":"952-239-0000","OrderID":17,"ItemID":1,"ItemName":"Meat Lovers Pizza","BaseCost":12.0000},{"UserID":9,"FirstName":"paul","LastName":"casey","Street":"Test1","City":"Test City1","Phone":"952-239-0000","OrderID":18,"ItemID":1,"ItemName":"Meat Lovers Pizza","BaseCost":12.0000},{"UserID":9,"FirstName":"paul","LastName":"casey","Street":"Test1","City":"Test City1","Phone":"952-239-0000","OrderID":18,"ItemID":3,"ItemName":"Hawaiian Pizza","BaseCost":12.9900},{"UserID":9,"FirstName":"paul","LastName":"casey","Street":"Test1","City":"Test City1","Phone":"952-239-0000","OrderID":19,"ItemID":1,"ItemName":"Meat Lovers Pizza","BaseCost":12.0000},{"UserID":9,"FirstName":"paul","LastName":"casey","Street":"Test1","City":"Test City1","Phone":"952-239-0000","OrderID":20,"ItemID":12,"ItemName":"pickle","BaseCost":2.9900}]}

Conclusion

ASP.Net Web API 2 provides a framework for building Restful web services that can be accessed through the internet using the HyperText Transfer Protocol (HTTP). This Blog discussed the ASP.Net Web API 2 technology as well as architecture and details for building and using it.