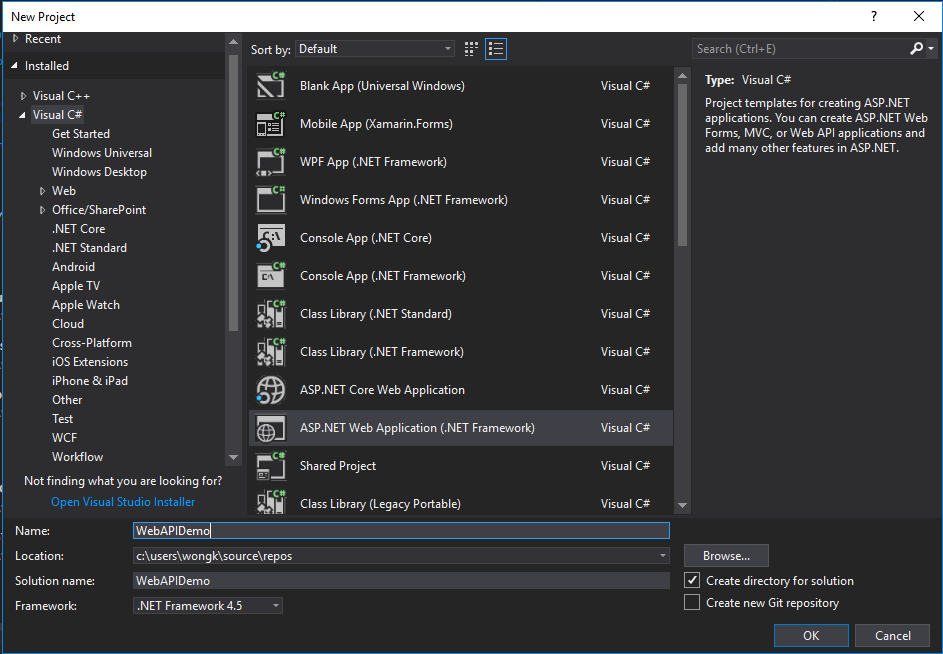
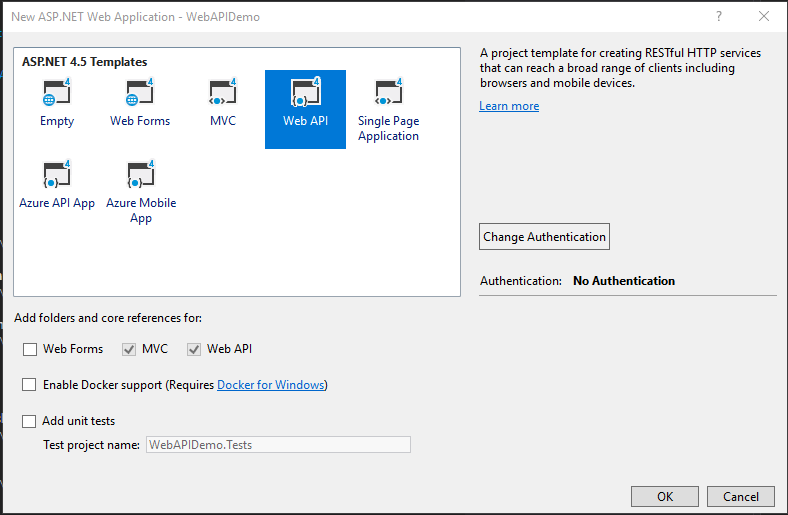
**LECTURE 2 – CREATING A WEB API PROJECT**

* In Visual Studio, create a new project.
* This project will be an **ASP.NET Web Application** project and will be called **WebAPIDemo**.



* Select **Web API** and then click **OK**.



**LECTURE 3 – HTTP GET PUT POST DELETE**

* In the ValuesController, add the following code and make the following changes.

namespace WebAPIDemo.Controllers

{

public class ValuesController : ApiController

{

static List<string> strings = new List<string>()

{

"value0", "value1", "value2"

};

// GET api/values

public IEnumerable<string> Get()

{

return strings;

}

// GET api/values/5

public string Get(int id)

{

return strings[id];

}

// POST api/values

public void Post([FromBody]string value)

{

strings.Add(value);

}

// PUT api/values/5

public void Put(int id, [FromBody]string value)

{

strings[id] = value;

}

// DELETE api/values/5

public void Delete(int id)

{

strings.RemoveAt(id);

}

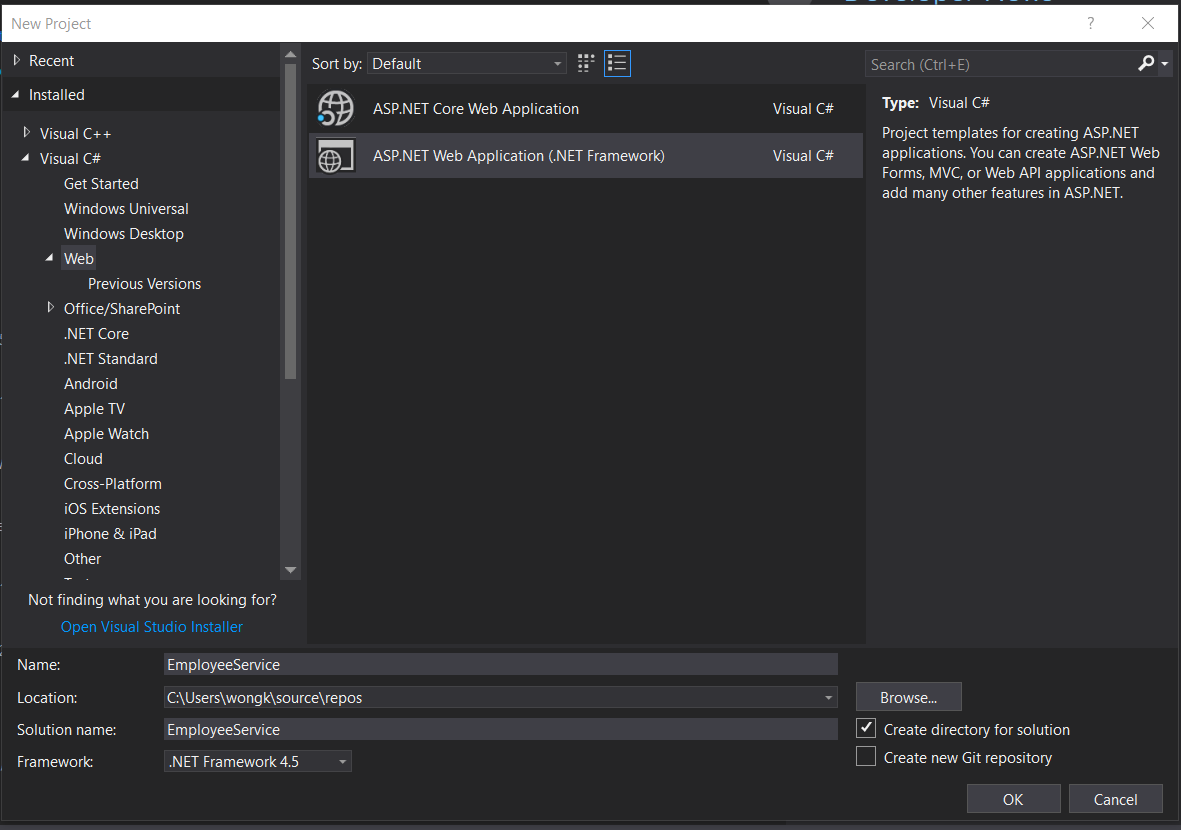
}

}

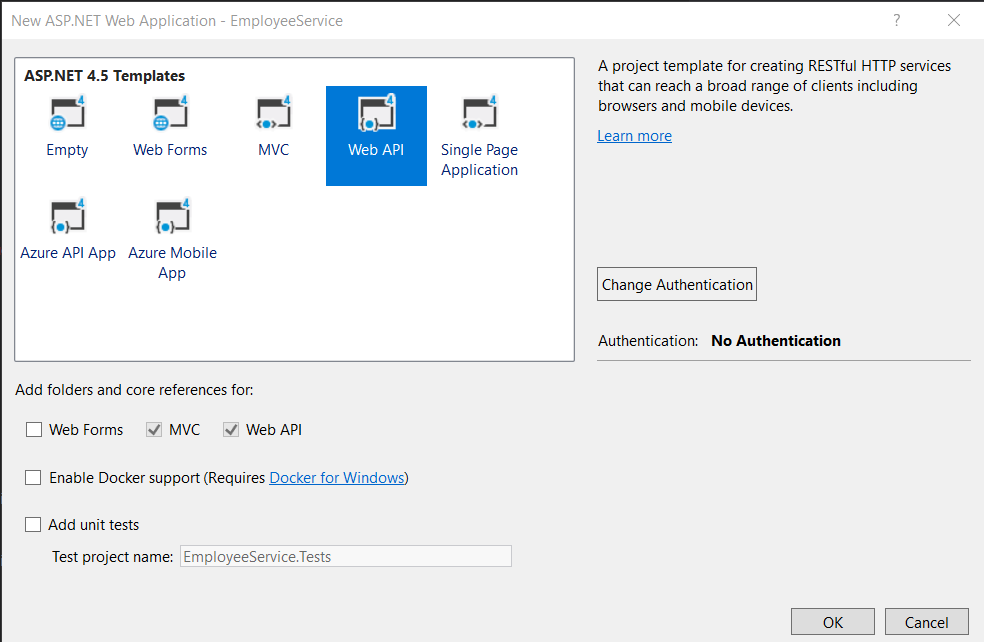
* Run the API.

**LECTURE 4 – ASP NET WEB API AND SQL SERVER**

* We will create a new Web Api Project.
* Go **File > New > Project**.
* Select **Visual C# > Web > ASP.NET Web Application (.NET Framework)** and name the project **EmployeeService** then click **OK**.



* Select **Web API** then click **OK**.



* Now go to **SQL Server** and run the following script. This script will create the **EmployeeDB** database and will also create a **Employees** table.

Create Database EmployeeDB

Go

Use EmployeeDB

Go

Create table Employees

(

ID int primary key identity,

FirstName nvarchar(50),

LastName nvarchar(50),

Gender nvarchar(50),

Salary int

)

Go

Insert into Employees values ('Mark', 'Hastings', 'Male', 60000)

Insert into Employees values ('Steve', 'Pound', 'Male', 45000)

Insert into Employees values ('Ben', 'Hoskins', 'Male', 70000)

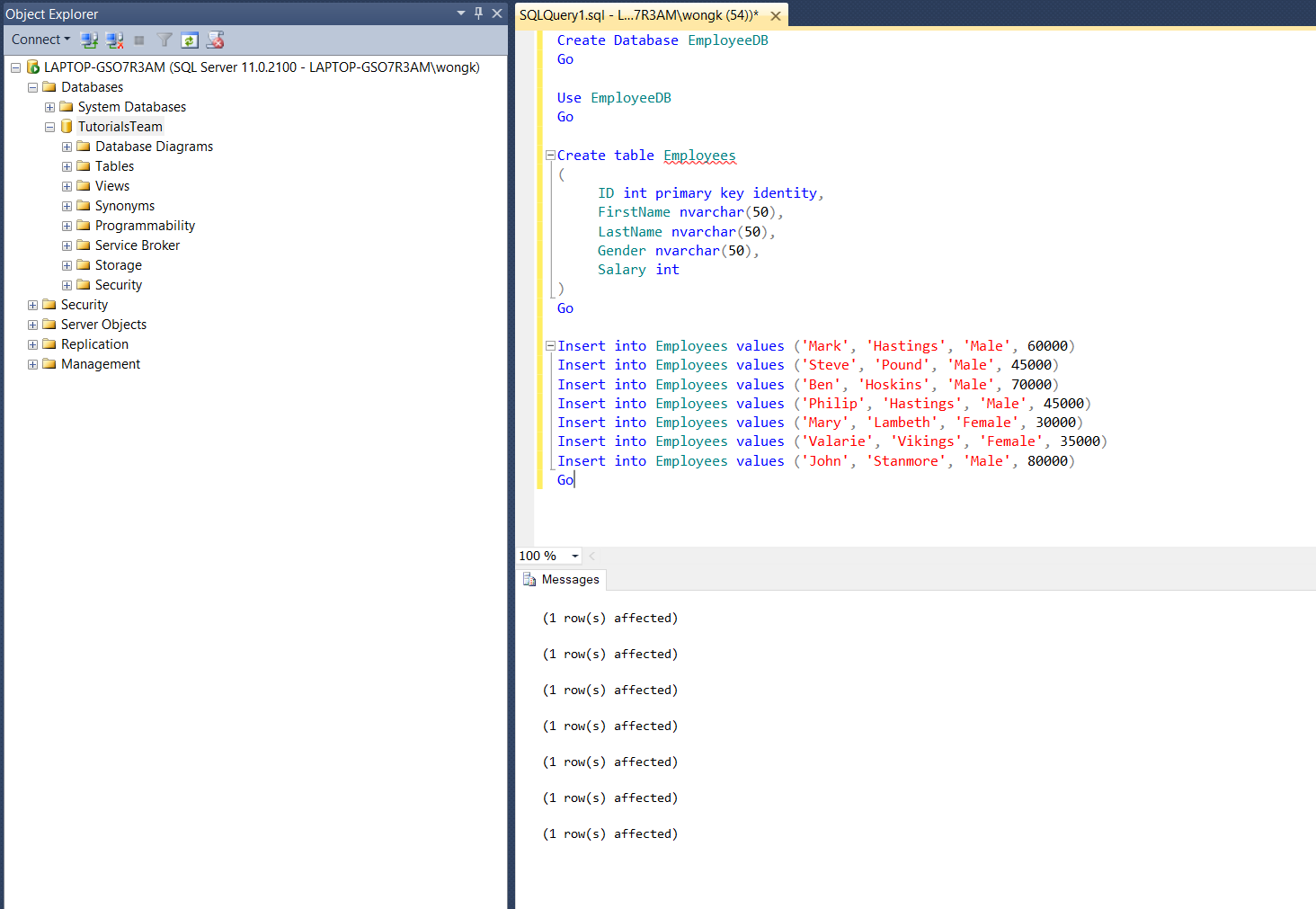
Insert into Employees values ('Philip', 'Hastings', 'Male', 45000)

Insert into Employees values ('Mary', 'Lambeth', 'Female', 30000)

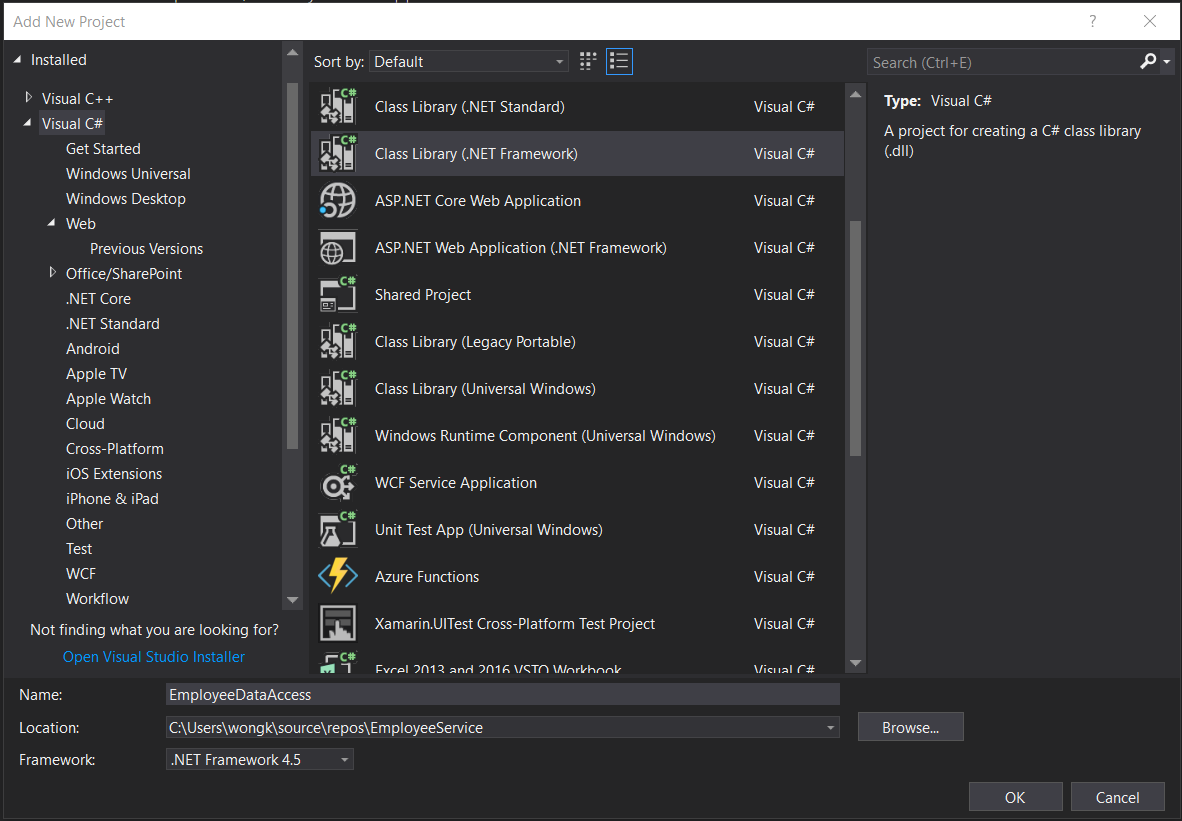
Insert into Employees values ('Valarie', 'Vikings', 'Female', 35000)

Insert into Employees values ('John', 'Stanmore', 'Male', 80000)

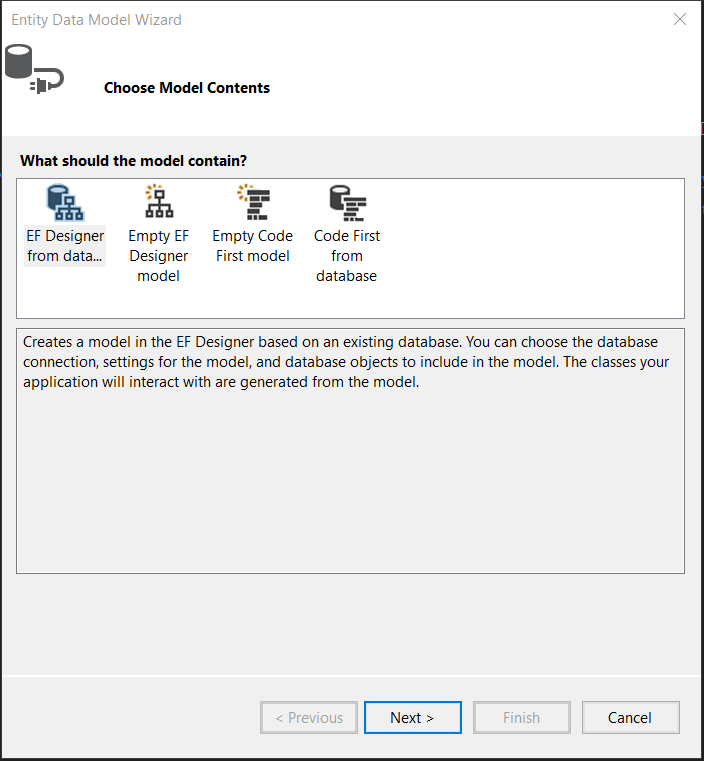
Go



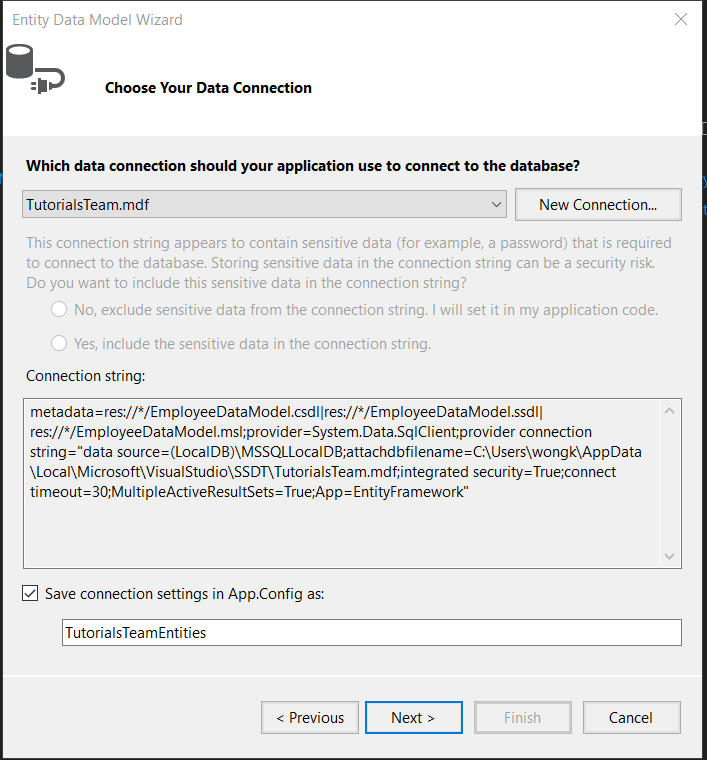
* Now go back to Visual Studio. We will now create another class that we will use to connect to SQL Server using Entity Framework.
* In VS Studio, right-click on the solution and select **Add > New Project**.
* Next, select **Visual C# > Class Library (.NET Framework)**.



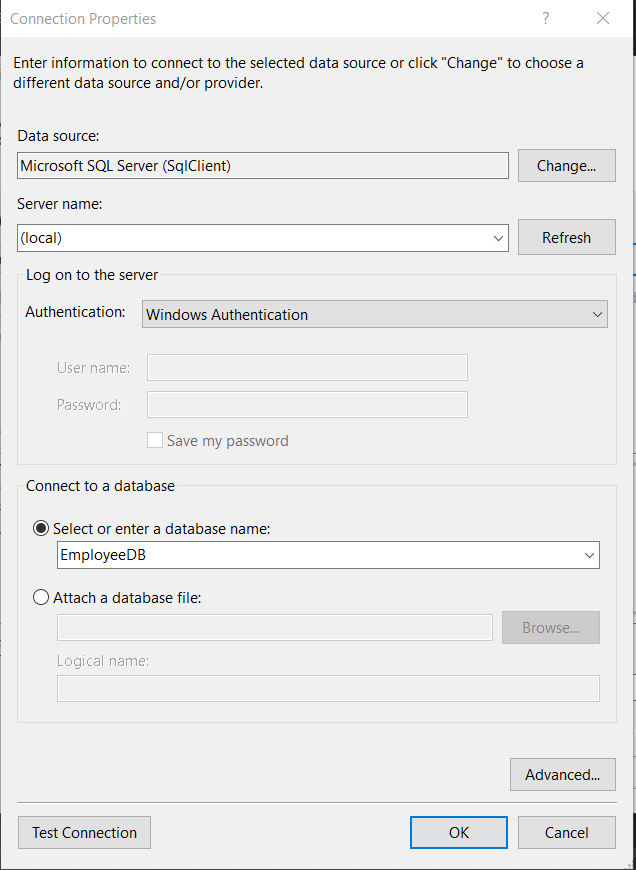
* In the **EmployeeDataAccess** project, we no longer need the **Class1.cs** file so you can delete it.
* We are now going to add the ADO.NET Entity Data Model.
* Right-click on the **EmployeeDataAccess** project and select **Add > New Item**.
* Select **Visual C# Items > Data > ADO.NET Entity Data Model** and name the Model **EmployeeDataModel** then click **Add**.
* Next select **EF Designer from data** and click **Next**.



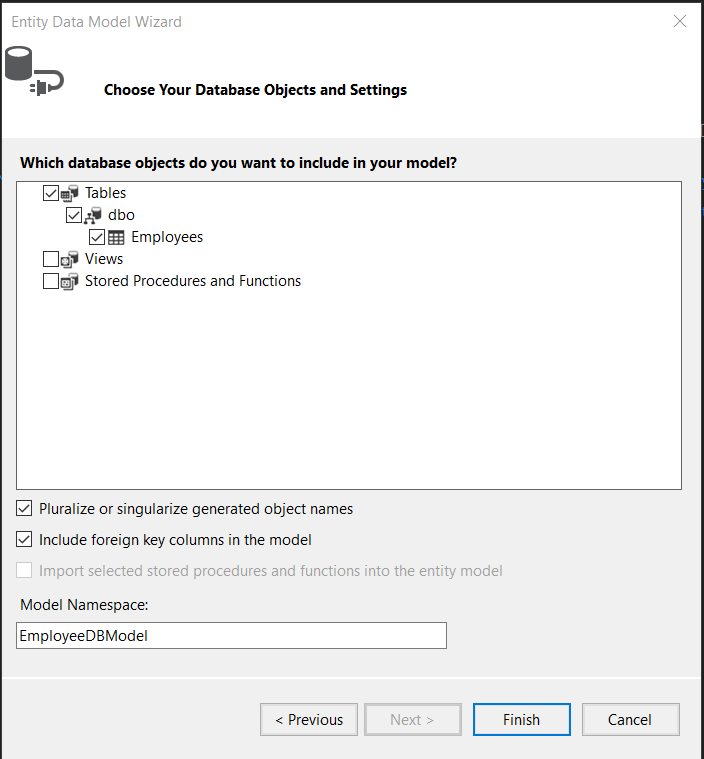
* Click on **New Connection**.



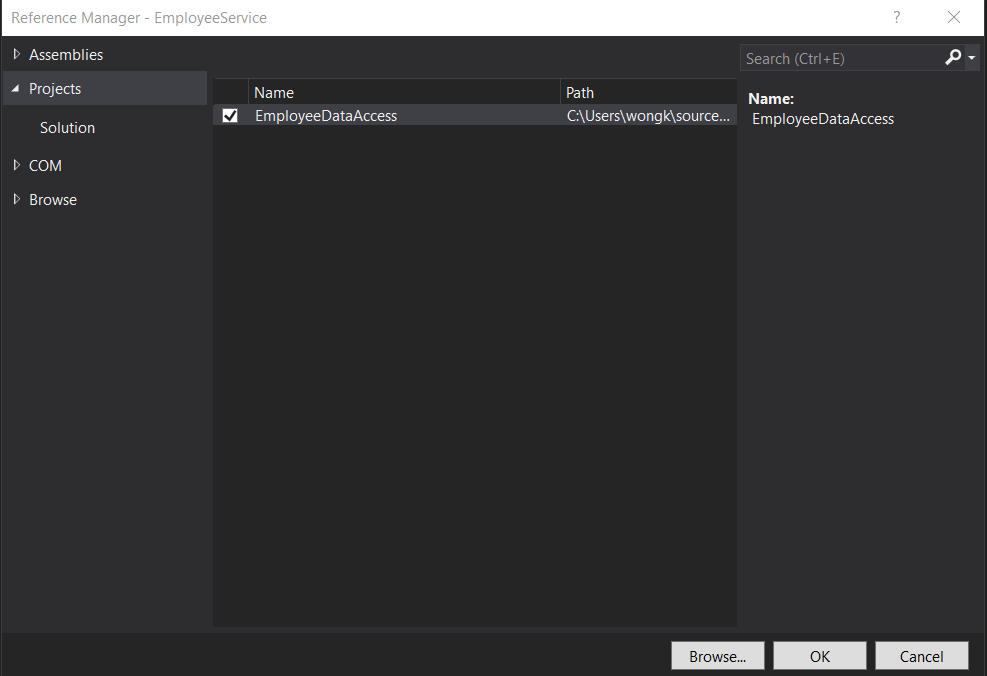
* As the Data source choose **Microsoft SQL Server (SqlClient)**.
* We are going to connect to the local database so type in **(local)** in the Database file name.
* In the Connect to a database dropdown select **EmployeeDB** and click **OK**.



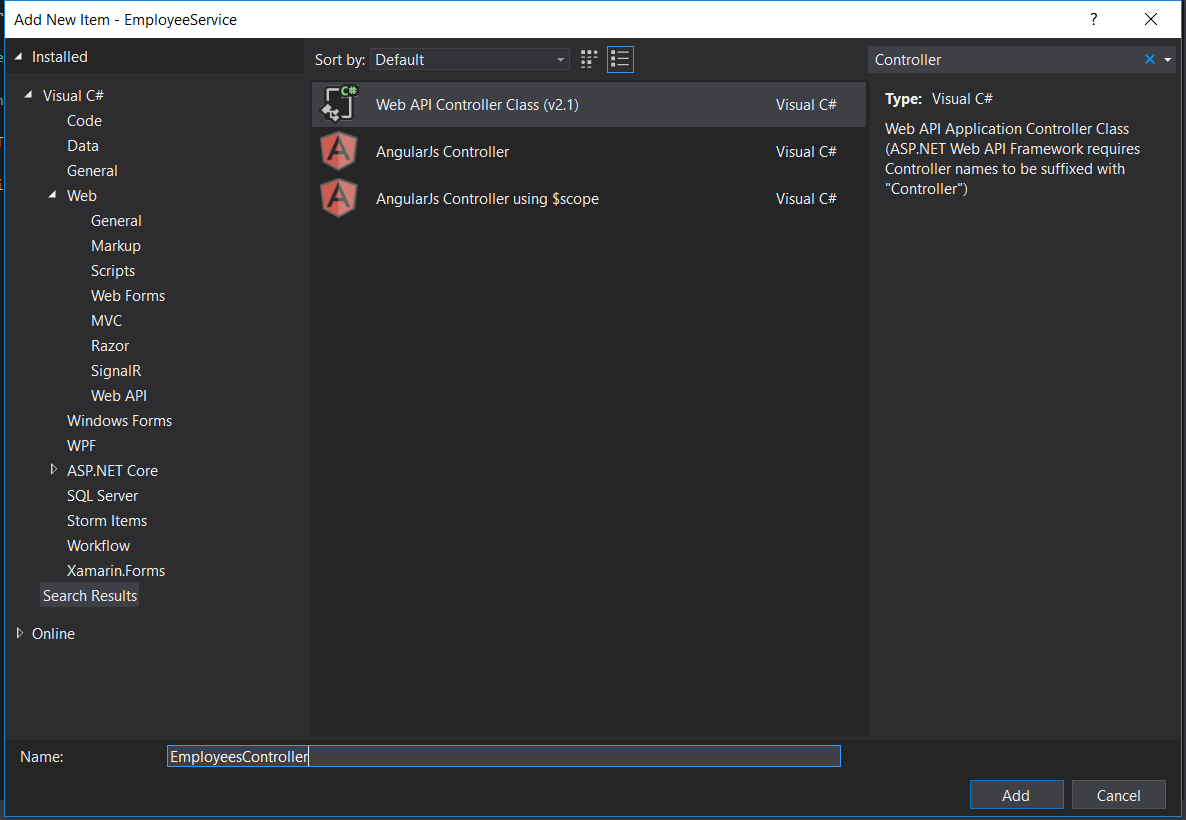
* Then click on **Next**.
* On the next screen choose **Entity Framework 6.x**.
* Select the following options as follows and leave the Model NameSpace to **EmployeeDBModel**. Click on **Finish** when done.



* We now want to add a reference to the data model from the **EmployeeService** project.
* Within the EmployeeService project right-click on the *References* and click on **Add Reference**.
* Select the Projects tab and then tick **EmployeeDataAccess** and press **OK**.



* Now we will add a Controller.
* Right-click on **Controllers** and select **Add > New Item**.
* Search for Controller in the search bar and then select **Web API Controller Class (v2.1)** and name the Controller **EmployeesController** and then click on **Add**.



* Go the **EmployeesController.cs** and delete all the code within the class.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

namespace EmployeeService.Controllers

{

public class EmployeesController : ApiController

{

// GET api/<controller>

public IEnumerable<string> Get()

{

return new string[] { "value1", "value2" };

}

// GET api/<controller>/5

public string Get(int id)

{

return "value";

}

// POST api/<controller>

public void Post([FromBody]string value)

{

}

// PUT api/<controller>/5

public void Put(int id, [FromBody]string value)

{

}

// DELETE api/<controller>/5

public void Delete(int id)

{

}

}

}

* Import the **EmployeeDataAccess** reference.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using EmployeeDataAccess;

namespace EmployeeService.Controllers

{

public class EmployeesController : ApiController

{

}

}

* Write the following boilerplate code for the GET method.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using EmployeeDataAccess;

namespace EmployeeService.Controllers

{

public class EmployeesController : ApiController

{

public IEnumerable<Employee> Get()

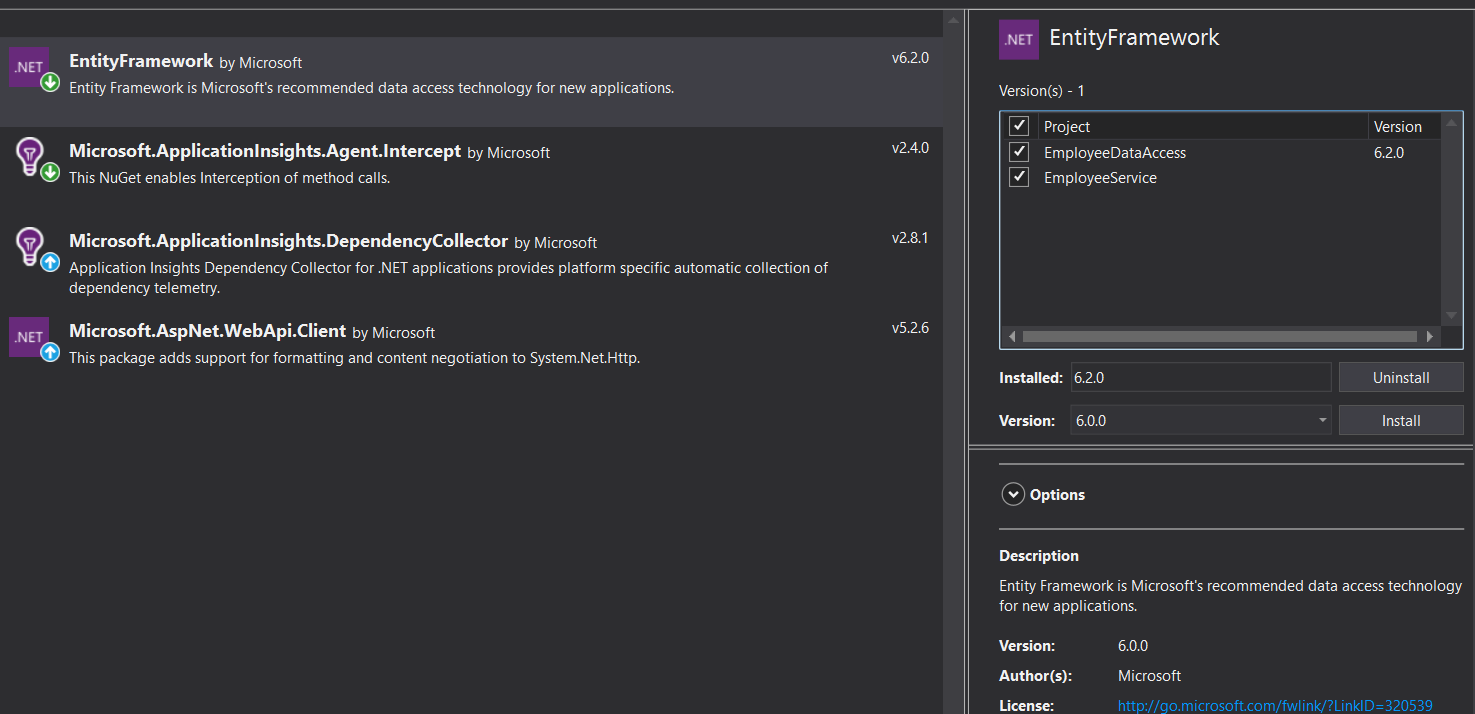
{

}

}

}

* Now we will need Entity Framework 6.0.0 so use the Nuget Package to install that version of Entity Framework.



* Write the following code within the GET Method.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using EmployeeDataAccess;

namespace EmployeeService.Controllers

{

public class EmployeesController : ApiController

{

public IEnumerable<Employee> Get()

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.ToList();

}

}

}

}

* Now we will write some code that will extract the data of one employee based on the id specified.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using EmployeeDataAccess;

namespace EmployeeService.Controllers

{

public class EmployeesController : ApiController

{

public IEnumerable<Employee> Get()

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.ToList();

}

}

public Employee Get(int id)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.FirstOrDefault(e => e.ID == id);

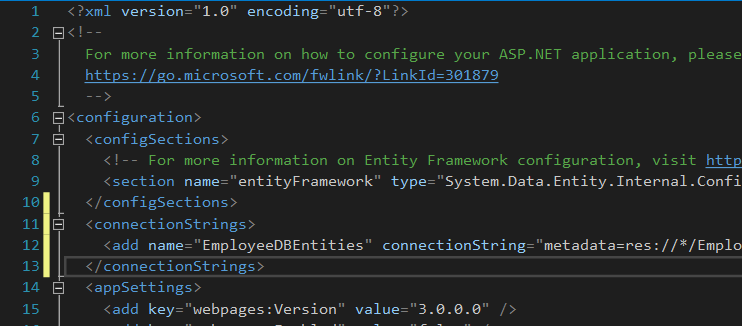
}

}

}

}

* We need to add the EmpoloyeesDBEntities connectionString in the Web.config of the **EmployeeService** project.
* Go to the App.config of the **EmployeeDataAccess** project and copy the **EmployeeDBEntities** connectionString.
* Paste the connectionString right under the **configSections**.



* Now when you run the API and type **localhost:12345/api/Employees** you should get all the Employees listed.



**LECTURE 5 – CONTENT NEGOATIATION**

* Content Negotiation is when the API tells the browser what format to send the data back in.
* The Web API Controller generates the data that we want to send to the client.
* We are now going to use a MediaTypeFormatter to change the format of the data.
* In the EmployeeService project go to **WebApiConfig.cs** and write the following code.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web.Http;

namespace EmployeeService

{

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

config.Formatters.JsonFormatter.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;

config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new Newtonsoft.Json.Serialization.CamelCasePropertyNamesContractResolver();

}

}

}

**LECTURE 6 – MEDIATYPEFORMATTER**

* Go to the **WebApiConfig.cs** file and comment out the previous 2 lines of MediaTypeFormatters that we wrote.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web.Http;

namespace EmployeeService

{

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

//config.Formatters.JsonFormatter.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;

//config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new Newtonsoft.Json.Serialization.CamelCasePropertyNamesContractResolver();

}

}

}

* What we are going to do now is remove the Xml Formatter.
* In WebApiConfig.cs write the following code to remove the Xml Formatter.

namespace EmployeeService

{

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

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

//config.Formatters.JsonFormatter.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;

//config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new Newtonsoft.Json.Serialization.CamelCasePropertyNamesContractResolver();

}

}

}

* Now we will remove the json formatter so that the response will be formatted only in xml.
* Comment out the remove xml formatter.

//#########################################################################################

// The below MediaTypeFormatter removes the Xml Formatter resulting in an only Json

// response.

//#########################################################################################

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

//#########################################################################################

// The below MediaTypeFormatter formats Json data by indenting the raw data response

//#########################################################################################

//config.Formatters.JsonFormatter.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;

//#########################################################################################

// The below MediaTypeFormatter formats Json data by turning the Response into Camel Case

//#########################################################################################

//config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new Newtonsoft.Json.Serialization.CamelCasePropertyNamesContractResolver();

}

}

}

* Write the following code to remove the Xml Formatter.

namespace EmployeeService

{

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

//#########################################################################################

// The below MediaTypeFormatter removes the Json Formatter resulting in an only Xml

// response.

//#########################################################################################

config.Formatters.Remove(config.Formatters.JsonFormatter);

//#########################################################################################

// The below MediaTypeFormatter removes the Xml Formatter resulting in an only Json

// response.

//#########################################################################################

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

//#########################################################################################

// The below MediaTypeFormatter formats Json data by indenting the raw data response

//#########################################################################################

//config.Formatters.JsonFormatter.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;

//#########################################################################################

// The below MediaTypeFormatter formats Json data by turning the Response into Camel Case

//#########################################################################################

//config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new Newtonsoft.Json.Serialization.CamelCasePropertyNamesContractResolver();

}

}

}

* Now we are going to add a MediaTypeFormatter that enables us to format the response into one type when receiving the data in the browser, but receives the response in a different format when receiving the data in a Web Tool like Postman.
* Comment out the previous json formatter remover.
* Type in the following code.

//#########################################################################################

// The below MediaTypeFormatter formats the response data as json when the Header Value

// is in text/html (meaning it's being received in the browser)

//#########################################################################################

config.Formatters.JsonFormatter.SupportedMediaTypes.Add(new System.Net.Http.Headers.MediaTypeHeaderValue("text/html"));

//#########################################################################################

// The below MediaTypeFormatter removes the Json Formatter resulting in an only Xml

// response.

//#########################################################################################

//config.Formatters.Remove(config.Formatters.JsonFormatter);

//#########################################################################################

// The below MediaTypeFormatter removes the Xml Formatter resulting in an only Json

// response.

//#########################################################################################

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

//#########################################################################################

// The below MediaTypeFormatter formats Json data by indenting the raw data response

//#########################################################################################

//config.Formatters.JsonFormatter.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;

//#########################################################################################

// The below MediaTypeFormatter formats Json data by turning the Response into Camel Case

//#########################################################################################

//config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new Newtonsoft.Json.Serialization.CamelCasePropertyNamesContractResolver();

}

}

}

* We are now going to create a custom formatter.
* First comment out all the previous MediaTypeFormatters.
* Ensure you have the following using statements.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web.Http;

using Newtonsoft.Json.Serialization;

using System.Net.Http.Formatting;

* Within the WebApiConfig.cs file create the following class.

namespace EmployeeService

{

public class CustomJsonFormatter : JsonMediaTypeFormatter

{

public CustomJsonFormatter()

{

this.SupportedMediaTypes.Add(new System.Net.Http.Headers.MediaTypeHeaderValue("text/html"));

}

public override void SetDefaultContentHeaders(Type type, HttpContentHeaders headers, MediaTypeHeaderValue mediaType)

{

base.SetDefaultContentHeaders(type, headers, mediaType);

headers.ContentType = new MediaTypeHeaderValue("application/json");

}

}

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

* Within the WebApiConfig class, write the following code to impolement our new custom formatter.

namespace EmployeeService

{

public class CustomJsonFormatter : JsonMediaTypeFormatter

{

public CustomJsonFormatter()

{

this.SupportedMediaTypes.Add(new System.Net.Http.Headers.MediaTypeHeaderValue("text/html"));

}

public override void SetDefaultContentHeaders(Type type, HttpContentHeaders headers, MediaTypeHeaderValue mediaType)

{

base.SetDefaultContentHeaders(type, headers, mediaType);

headers.ContentType = new MediaTypeHeaderValue("application/json");

}

}

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

//#########################################################################################

// Using our new custom formatters

//#########################################################################################

config.Formatters.Add(new CustomJsonFormatter());

**LECTURE 8 – IMPLEMENTING POST METHOD IN ASPNET WEB API**

* First comment out the CustomJsonFormatter class we created in the WebApiConfig.cs file and also the calling of the custom formatter within the WebApiConfig class.

namespace EmployeeService

{

//public class CustomJsonFormatter : JsonMediaTypeFormatter

//{

// public CustomJsonFormatter()

// {

// this.SupportedMediaTypes.Add(new System.Net.Http.Headers.MediaTypeHeaderValue("text/html"));

// }

// public override void SetDefaultContentHeaders(Type type, HttpContentHeaders headers, MediaTypeHeaderValue mediaType)

// {

// base.SetDefaultContentHeaders(type, headers, mediaType);

// headers.ContentType = new MediaTypeHeaderValue("application/json");

// }

//}

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

//#########################################################################################

// Using our new custom formatters

//#########################################################################################

//config.Formatters.Add(new CustomJsonFormatter());

//#########################################################################################

// The below MediaTypeFormatter formats the response data as json when the Header Value

// is in text/html (meaning it's being received in the browser)

//#########################################################################################

//config.Formatters.JsonFormatter.SupportedMediaTypes.Add(new System.Net.Http.Headers.MediaTypeHeaderValue("text/html"));

* Go to the EmployeesController.cs and write the following POST method.

namespace EmployeeService.Controllers

{

public class EmployeesController : ApiController

{

public IEnumerable<Employee> Get()

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.ToList();

}

}

public Employee Get(int id)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.FirstOrDefault(e => e.ID == id);

}

}

public void Post([FromBody] Employee employee)

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

entities.Employees.Add(employee);

entities.SaveChanges();

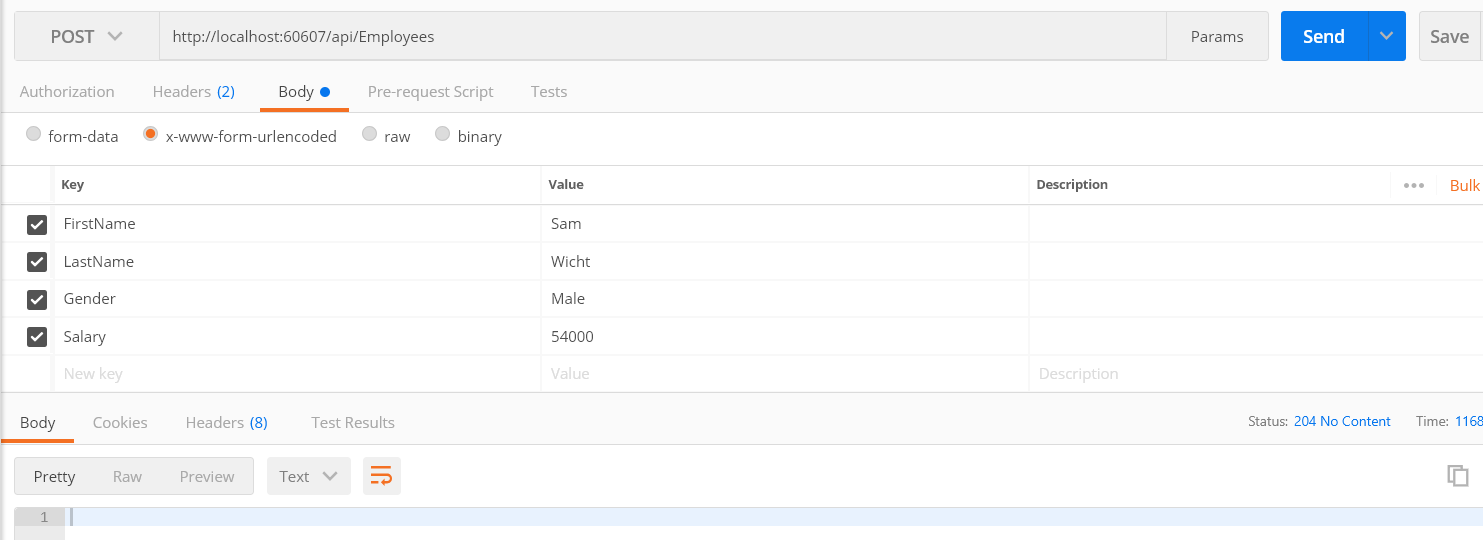
}

}

}

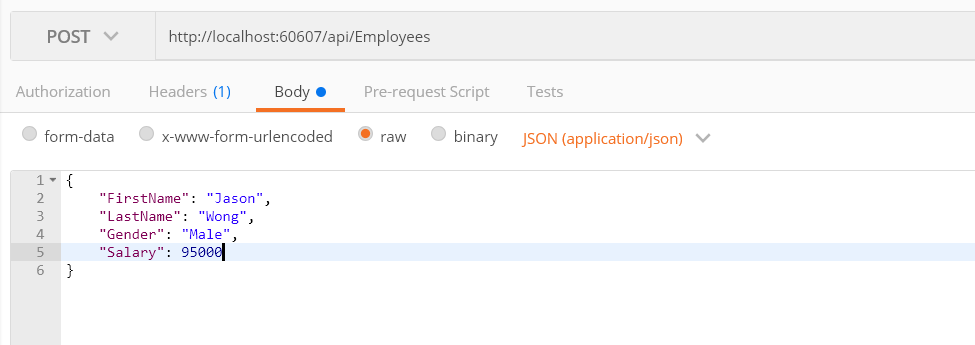
}

* Now in Postman, send a Post Request with the following details.



* If we want to specify that the type of data we are going to send is in json format.
* In the Header type **Content-Type: application/json**.





* Currently we are returning no content even when the Post Request was successful.
* In order to return proper Http Responses we use the HttpResponseMessage as the return type.
* Make the following amendments to the POST method.

public class EmployeesController : ApiController

{

public IEnumerable<Employee> Get()

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.ToList();

}

}

public Employee Get(int id)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.FirstOrDefault(e => e.ID == id);

}

}

public HttpResponseMessage Post([FromBody] Employee employee)

{

try

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

entities.Employees.Add(employee);

entities.SaveChanges();

var message = Request.CreateResponse(HttpStatusCode.Created, employee);

// This will return the Uri where the newly created object is located

// Typing this Uri in the browser will allow you to see the newly created data.

message.Headers.Location = new Uri(Request.RequestUri + “/” + employee.ID.ToString());

return message;

}

}

catch (Exception ex)

{

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);

}

}

}

* Using Postman send another Post Request.



* If you look at the Header Response you should see the location of the new item that you just posted.



* Now we are going to modify the code so that instead of returning a response of null (as well as a successful 200 response) when you do a GET method on an item that does not exist you will now get a 404 not found response.
* Modify the Get(int id) method so that it looks like the following.

namespace EmployeeService.Controllers

{

public class EmployeesController : ApiController

{

public IEnumerable<Employee> Get()

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.ToList();

}

}

public HttpResponseMessage Get(int id)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

if (entity != null)

{

return Request.CreateResponse(HttpStatusCode.OK, entity);

}

else

{

return Request.CreateErrorResponse(HttpStatusCode.NotFound, "Employee with id = " + id.ToString() + " not found");

}

}

}

**LECTURE 8 – IMPLEMENTING DELETE METHOD IN ASPNET WEB API**

* Go to the EmployeesController.cs file and write the following DELETE method.

public HttpResponseMessage Post([FromBody] Employee employee)

{

try

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

entities.Employees.Add(employee);

entities.SaveChanges();

var message = Request.CreateResponse(HttpStatusCode.Created, employee);

// This will return the Uri where the newly created object is located

// Typing this Uri in the browser will allow you to see the newly created data.

message.Headers.Location = new Uri(Request.RequestUri + "/" + employee.ID.ToString());

return message;

}

}

catch (Exception ex)

{

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);

}

}

public void Delete(int id)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

entities.Employees.Remove(entities.Employees.FirstOrDefault(e => e.ID == id));

entities.SaveChanges();

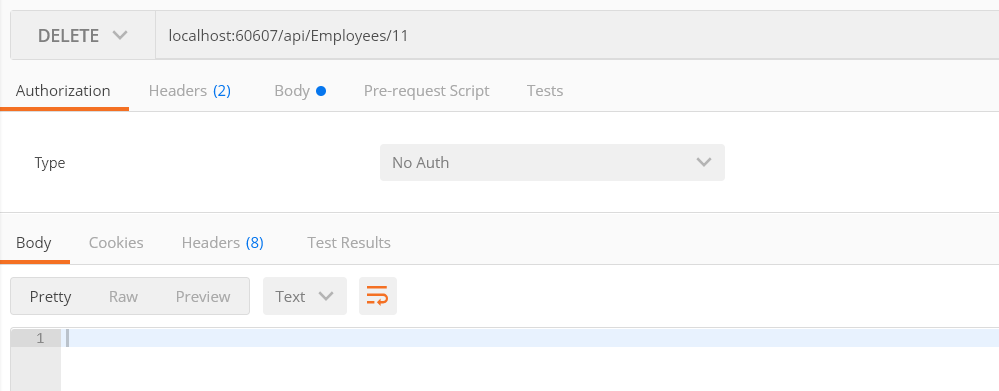
}

}

}

}

* Go to Postman and send the following DELETE Request.



* We are now going to modify the code so that it returns a HttpResponse instead of void.
* In the EmployeesController make the following changes.

public HttpResponseMessage Post([FromBody] Employee employee)

{

try

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

entities.Employees.Add(employee);

entities.SaveChanges();

var message = Request.CreateResponse(HttpStatusCode.Created, employee);

// This will return the Uri where the newly created object is located

// Typing this Uri in the browser will allow you to see the newly created data.

message.Headers.Location = new Uri(Request.RequestUri + "/" + employee.ID.ToString());

return message;

}

}

catch (Exception ex)

{

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);

}

}

public HttpResponseMessage Delete(int id)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

try

{

var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

if (entity == null)

{

return Request.CreateErrorResponse(HttpStatusCode.NotFound, "Employee with id = " + id.ToString() + " not found to delete");

}

else

{

entities.Employees.Remove(entity);

entities.SaveChanges();

return Request.CreateResponse(HttpStatusCode.OK);

}

}

catch (Exception ex)

{

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);

}

}

}

}

}

**LECTURE 9 – IMPLEMENTING PUT METHOD IN ASPNET WEBAPI**

* Write the following PUT method in WebApiConfig.cs.

public HttpResponseMessage Delete(int id)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

try

{

var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

if (entity == null)

{

return Request.CreateErrorResponse(HttpStatusCode.NotFound, "Employee with id = " + id.ToString() + " not found to delete");

}

else

{

entities.Employees.Remove(entity);

entities.SaveChanges();

return Request.CreateResponse(HttpStatusCode.OK);

}

}

catch (Exception ex)

{

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);

}

}

}

public void Put(int id, [FromBody] Employee employee)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

entity.FirstName = employee.FirstName;

entity.LastName = employee.LastName;

entity.Gender= employee.Gender;

entity.Salary = employee.Salary;

entities.SaveChanges();

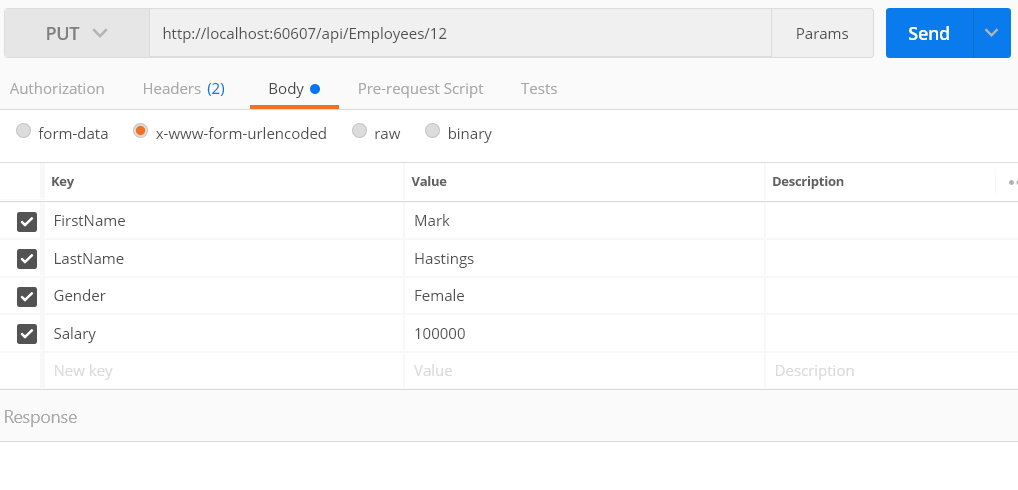
}

}

}

}

* Make the following PUT Request.



* Make the following amendments to the PUT method.

public HttpResponseMessage Delete(int id)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

try

{

var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

if (entity == null)

{

return Request.CreateErrorResponse(HttpStatusCode.NotFound, "Employee with id = " + id.ToString() + " not found to delete");

}

else

{

entities.Employees.Remove(entity);

entities.SaveChanges();

return Request.CreateResponse(HttpStatusCode.OK);

}

}

catch (Exception ex)

{

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);

}

}

}

public HttpResponseMessage Put(int id, [FromBody] Employee employee)

{

try

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

if (entity == null)

{

return Request.CreateErrorResponse(HttpStatusCode.NotFound, "Employee with id = " + id.ToString() + " not found to update");

}

else

{

entity.FirstName = employee.FirstName;

entity.LastName = employee.LastName;

entity.Gender = employee.Gender;

entity.Salary = employee.Salary;

entities.SaveChanges();

return Request.CreateResponse(HttpStatusCode.OK, entity);

}

}

}

catch (Exception ex)

{

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);

}

}

}

}

**LECTURE 10 – CUSTOM METHOD NAMES IN ASPNET WEB API**

* Now we are going to rename the HTTP Methods from their standard GET, POST, PUT, DELETE to different names.
* Rename the GET all employees method from the simple **Get()** to **LoadAllEmployees()**.
* In order to ensure that all HTTP Requests know to refer to this method when sending a GET Request add the **HttpGet** attribute on top of the method.

namespace EmployeeService.Controllers

{

public class EmployeesController : ApiController

{

[HttpGet]

public IEnumerable<Employee> LoadAllEmployees()

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.ToList();

}

}

public HttpResponseMessage Get(int id)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

if (entity != null)

{

return Request.CreateResponse(HttpStatusCode.OK, entity);

}

else

{

return Request.CreateErrorResponse(HttpStatusCode.NotFound, "Employee with id = " + id.ToString() + " not found");

}

}

}

* Apply the same changes to the **Get(int id)** method. Rename the method **LoadEmployeeById**.

namespace EmployeeService.Controllers

{

public class EmployeesController : ApiController

{

[HttpGet]

public IEnumerable<Employee> LoadAllEmployees()

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.ToList();

}

}

[HttpGet]

public HttpResponseMessage LoadEmployeeById(int id)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

if (entity != null)

{

return Request.CreateResponse(HttpStatusCode.OK, entity);

}

else

{

return Request.CreateErrorResponse(HttpStatusCode.NotFound, "Employee with id = " + id.ToString() + " not found");

}

}

}

**LECTURE 11 – ASPNET WEBAPI QUERY STRING PARAMETERS**

* We are now going to use query parameters in the HTTP Requests.
* With the Get method that we changed to LoadAllEmployees, rename the method back to **Get** and remove the HttpGet attribute.

namespace EmployeeService.Controllers

{

public class EmployeesController : ApiController

{

public IEnumerable<Employee> Get()

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.ToList();

}

}

* Amend the Get() method to look like the following.

namespace EmployeeService.Controllers

{

public class EmployeesController : ApiController

{

public HttpResponseMessage Get(string gender = "all")

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

switch (gender.ToLower())

{

case "all":

return Request.CreateResponse(HttpStatusCode.OK, entities.Employees.ToList());

case "male":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "male").ToList());

case "female":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "female").ToList());

default:

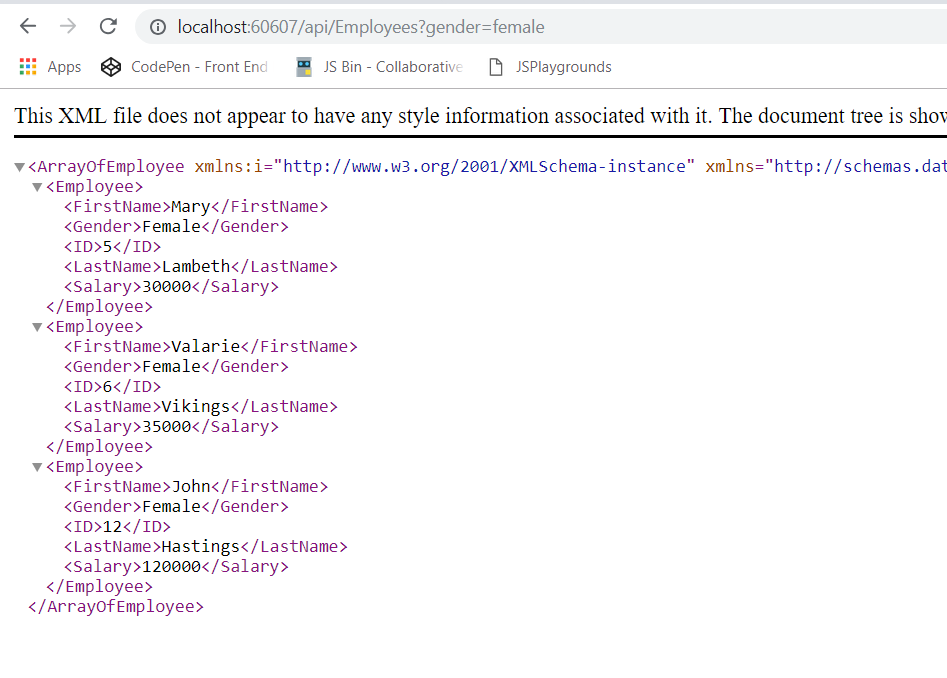
return Request.CreateErrorResponse(HttpStatusCode.BadRequest, "Value for gender must be All, Mall or Female. " + gender + " is invalid");

}

}

}

* Launch the API and send the following GET request with the following query string. With this you will receive all employees of gender female.



**LECTURE 12 – FROMBODY AND FROMURI IN WEB API**

* **Parameter Binding** is when the API tries to get Request data that is simple from the Request URI, while getting the more complex object data from the Request Body. This is the normal convention.
* Now we are going to use **FromUri** to force the PUT method to look in the Uri query string instead of the Request Body in order to perform the PUT Request.
* Make the following amendments to the PUT method.

public HttpResponseMessage Put(int id, [FromUri] Employee employee)

{

try

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

if (entity == null)

{

return Request.CreateErrorResponse(HttpStatusCode.NotFound, "Employee with id = " + id.ToString() + " not found to update");

}

else

{

entity.FirstName = employee.FirstName;

entity.LastName = employee.LastName;

entity.Gender = employee.Gender;

entity.Salary = employee.Salary;

entities.SaveChanges();

return Request.CreateResponse(HttpStatusCode.OK, entity);

}

}

}

catch (Exception ex)

{

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);

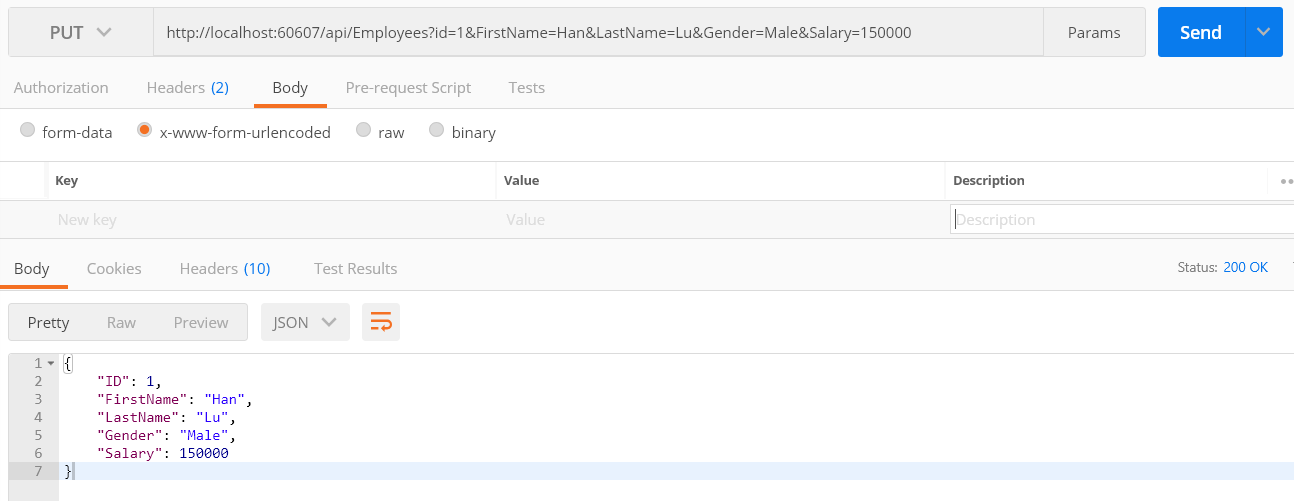
}

}

* Go to Postman and send the following PUT Request with the fields being located in the Query String.



* The records should be updated.



* Now what we’re going to do make changes to the PUT method so that the Employee Id will be extracted from the Request Body.

public HttpResponseMessage Put([FromBody]int id, [FromUri] Employee employee)

{

try

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

var entity = entities.Employees.FirstOrDefault(e => e.ID == id);

if (entity == null)

{

return Request.CreateErrorResponse(HttpStatusCode.NotFound, "Employee with id = " + id.ToString() + " not found to update");

}

else

{

entity.FirstName = employee.FirstName;

entity.LastName = employee.LastName;

entity.Gender = employee.Gender;

entity.Salary = employee.Salary;

entities.SaveChanges();

return Request.CreateResponse(HttpStatusCode.OK, entity);

}

}

}

catch (Exception ex)

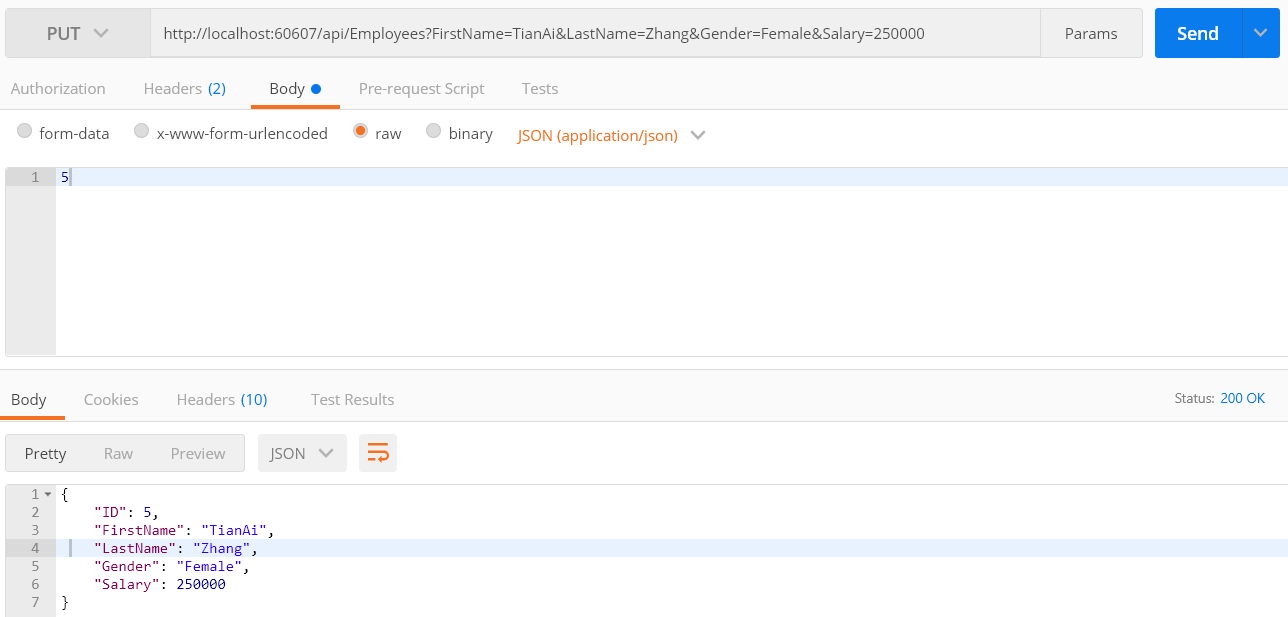
{

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ex);

}

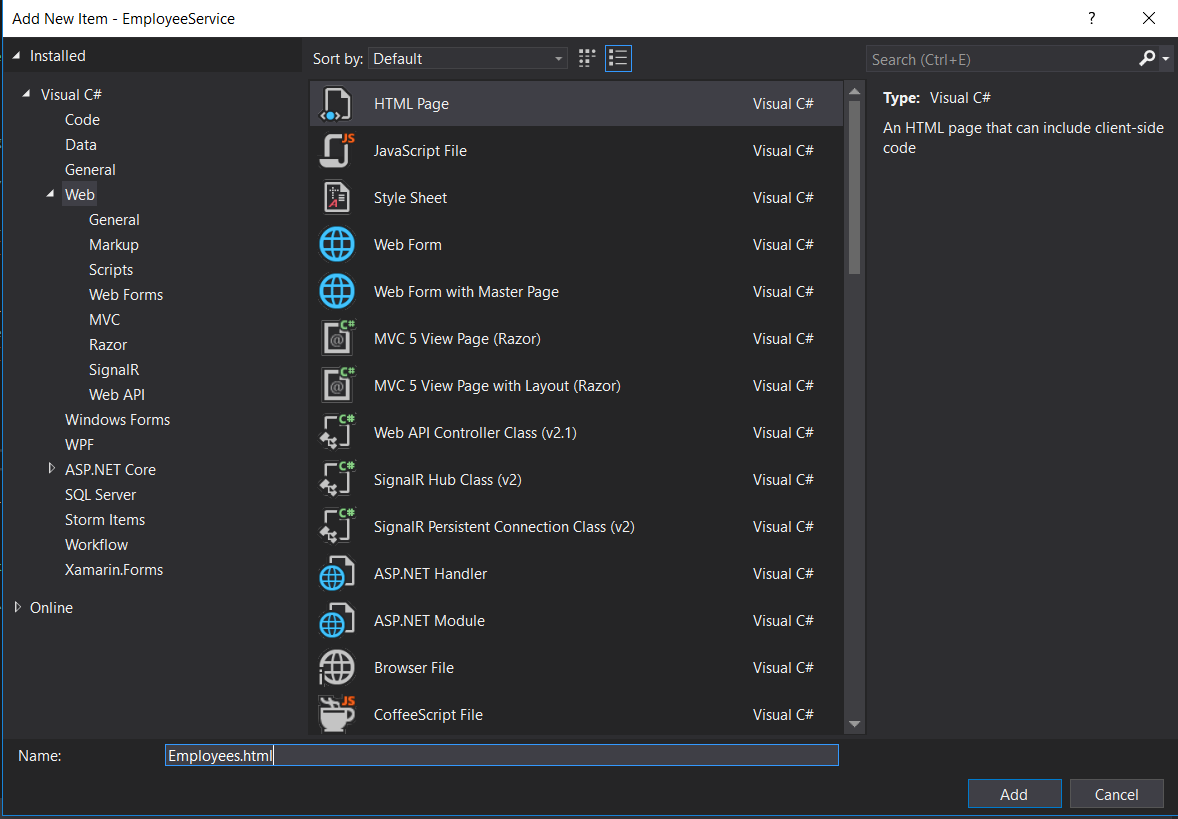
}

* Now send the following PUT Request with the id now in the Request Body.

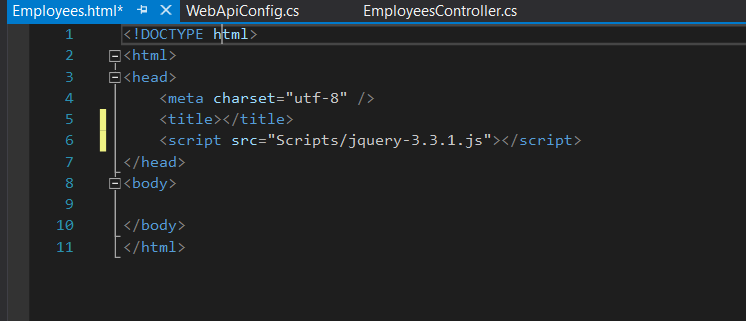


**LECTURE 13 – CALL ASPNET WEBAPI FROM JQUERY**

* Now we are going to create a simple HTML page that will enable us to call the WebApi and display the Employee names and also to delete them.
* Right-click on the **EmployeeService** project and select **Add > New Item**.
* Select **Visual C$ > Web > HTML Page** and name the page **Employees.html** and click on **Add**.



* We need to refer to jQuery in the html page so drag and drop **jquery-3.3.1.js** within the head tag.



* In the Employees.html amend the code to look like the following.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script src="Scripts/jquery-3.3.1.js"></script>

<script type="text/javascript">

$(document).ready(function () {

var ulEmployees = $('#ulEmployees');

$('#btn').click(function () {

$.ajax({

type: 'GET',

url: 'api/Employees',

dataType: 'json',

success: function (data) {

ulEmployees.empty();

$.each(data, function (index, val) {

var fullName = val.FirstName + ' ' + val.LastName;

ulEmployees.append('<li>' + fullName + '</li>')

});

}

});

});

$('#btnClear').click(function () {

ulEmployees.empty();

});

});

</script>

</head>

<body>

<input type="button" value="Get All Employees" id="btn" />

<input type="button" value="Clear" id="btnClear" />

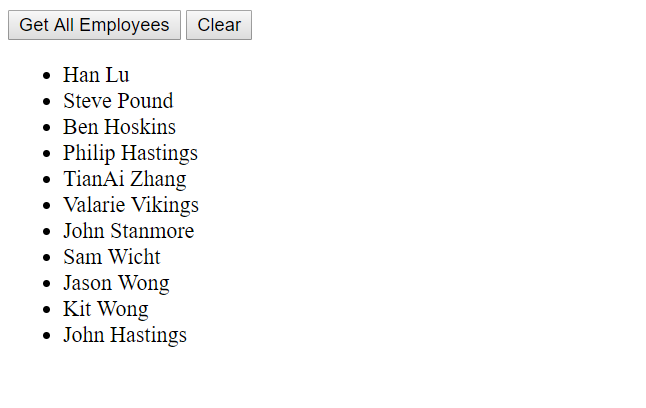
<ul id="ulEmployees">

</ul>

</body>

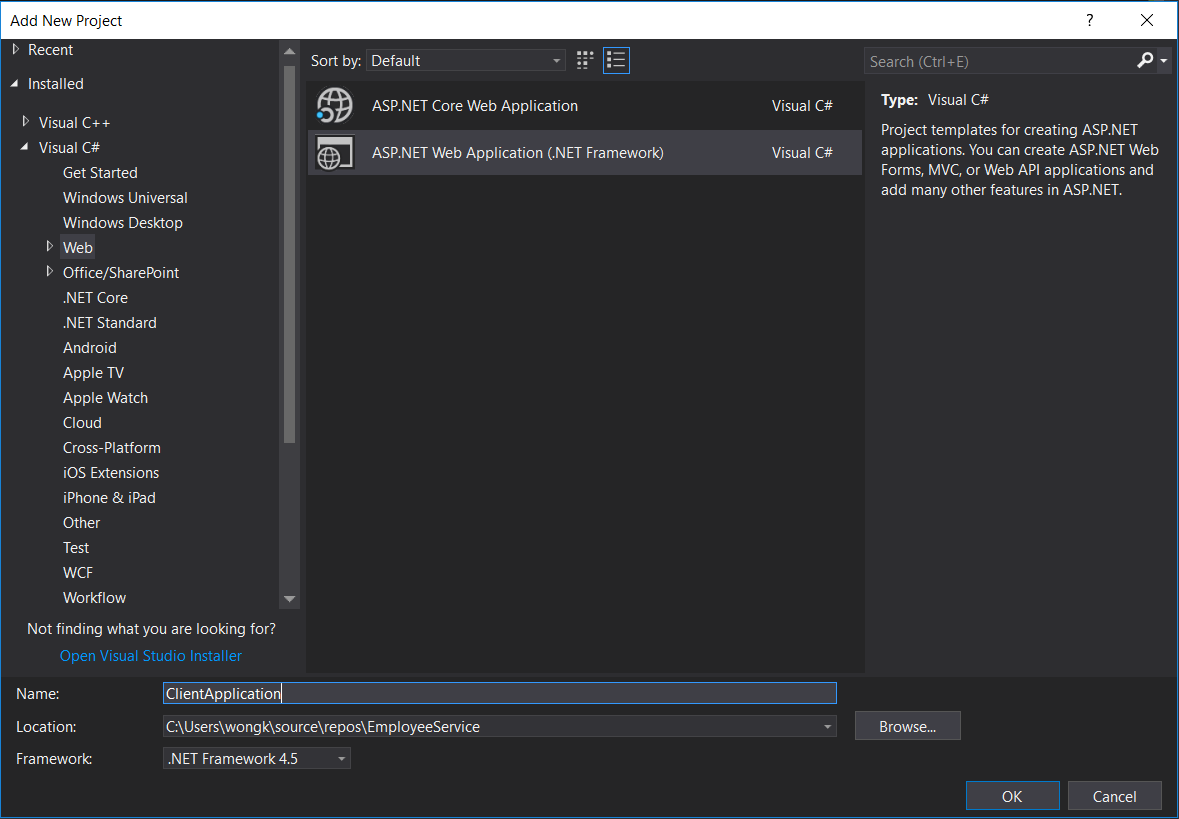
</html>

* Build the API.
* When you click on the **Get All Employees** button the names should be listed.

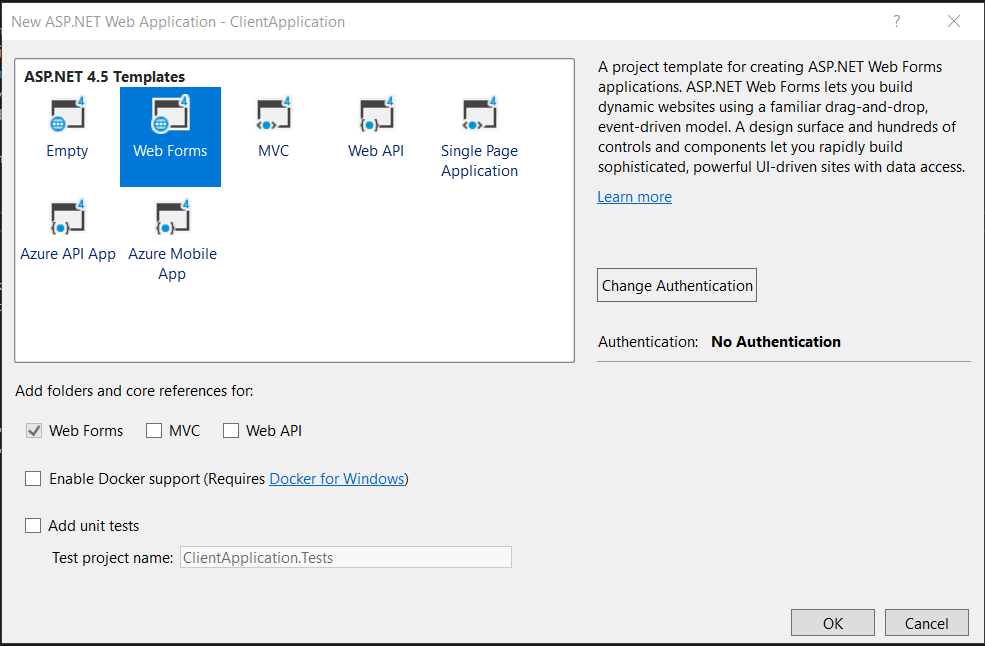


**LECTURE 14 – CALLING ASPNET WEBAPI SERVICE IN A CROSS DOMAIN USING JQUERY AJAX**

* **Same Origin Policy** means browsers will not allow you to make Ajax calls from one domain to another domain.
* We are going to add a new project on to the Solution.
* Right-click on the solution and select **Add > New Project**.
* Select **Visual C# > Web > ASP.NET Web Application (.NET Framework)** and name the application **ClientApplication** and click **OK**.



* Select **WebForms** and click **OK**.



* On the **ClientApplication** project, right-click and select **Add > HTML Page** and leave the default name of HtmlPage1 and click **OK**.
* Now under the **EmployeeService** Project copy all the code in the Employees.html page and paste it in the **HtmlPage1.html** under the **ClientApplication** project.
* In the HtmlPage1.html change the url so that it’s not just a relative url but the full url.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script src="Scripts/jquery-3.3.1.js"></script>

<script type="text/javascript">

$(document).ready(function () {

var ulEmployees = $('#ulEmployees');

$('#btn').click(function () {

$.ajax({

type: 'GET',

url: 'http://localhost:60607/api/Employees',

dataType: 'json',

success: function (data) {

ulEmployees.empty();

$.each(data, function (index, val) {

var fullName = val.FirstName + ' ' + val.LastName;

ulEmployees.append('<li>' + fullName + '</li>')

});

}

});

});

$('#btnClear').click(function () {

ulEmployees.empty();

});

});

</script>

</head>

<body>

<input type="button" value="Get All Employees" id="btn" />

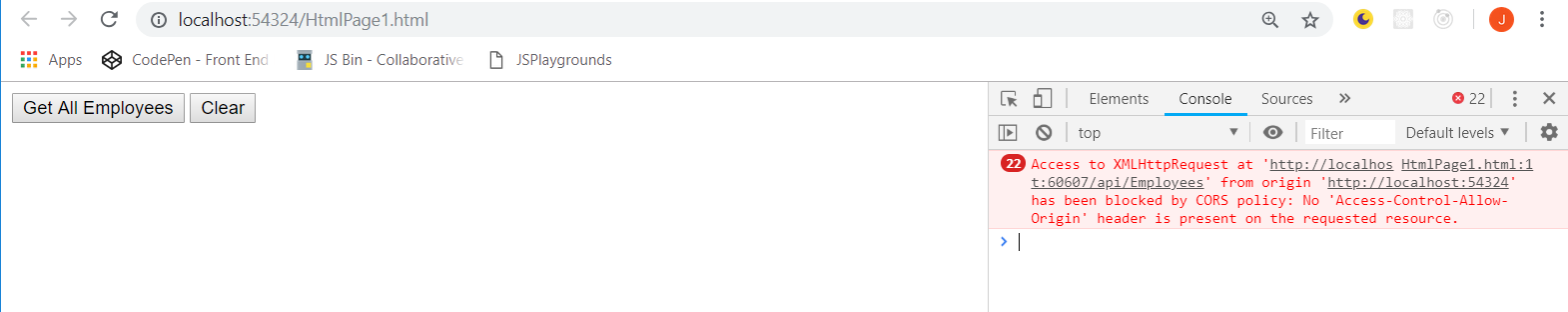
<input type="button" value="Clear" id="btnClear" />

<ul id="ulEmployees"></ul>

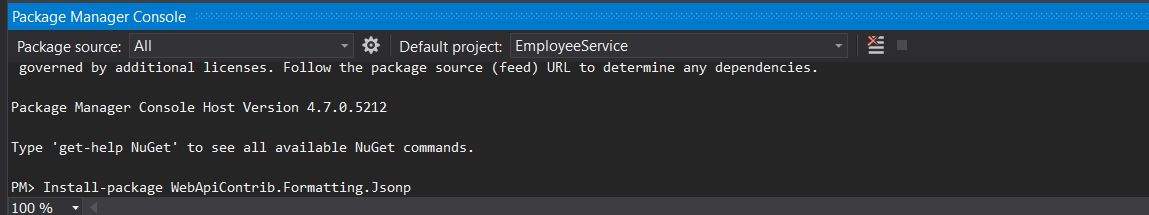
</body>

</html>

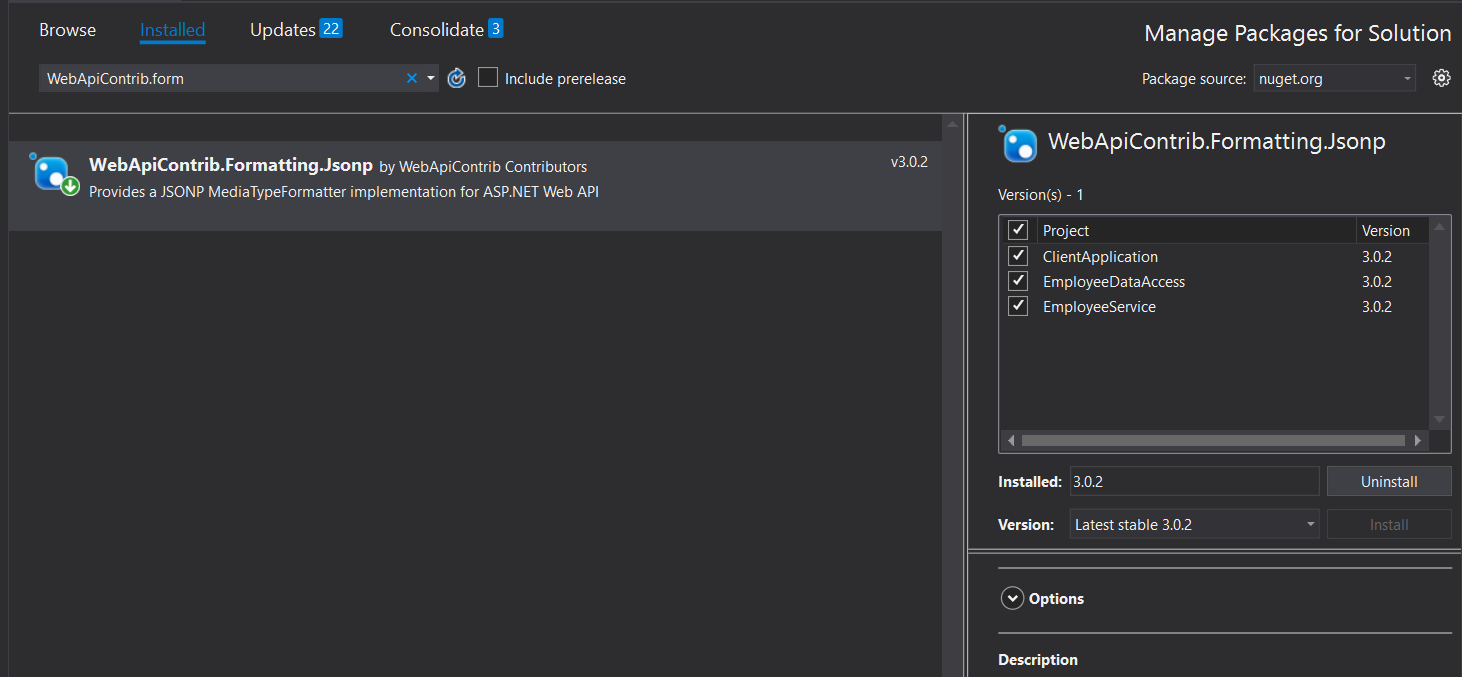
* Now run the API. Also right-click on HtmlPage1.html and select **View in Browser**. This way we have both the API running and the Client page running, both on different origins (In this case they should be in different port numbers).
* If you click on **Get All Employees** in the Client page you should get the following error response in the Console.



* There are 2 ways to get around this problem.
* Using **JSONP** (JSON with Padding)
* Enabling **CORS** (Cross Origin Resource Sharing)
* In this lecture we will be discussing **JSONP**.
* All JSONP does is wrap the data in a function.
* To use JSONP we must first install the **WebApiContrib.Formatting.Jsonp** nuget package.
* Go to **Tools > Nuget Package Manager > Package Manager Console.**
* In the Package Manager Console type **Install-package WebApiContrib.Formatting.Jsonp**.



* Alternatively, you can go to **Tools > Nuget Package Manager > Manage Nuget Packages for Solution** and search for **WebApiContrib.Formatting.Jsonp** and install the packages from there.



* Write the following using statement to bring in the Jsonp reference.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Web.Http;

using Newtonsoft.Json.Serialization;

using System.Net.Http.Formatting;

using System.Net.Http.Headers;

using WebApiContrib.Formatting.Jsonp;

* Go to **EmployeeService/App\_Start/WebApiConfig.cs** and write the following code.

namespace EmployeeService

{

//public class CustomJsonFormatter : JsonMediaTypeFormatter

//{

// public CustomJsonFormatter()

// {

// this.SupportedMediaTypes.Add(new System.Net.Http.Headers.MediaTypeHeaderValue("text/html"));

// }

// public override void SetDefaultContentHeaders(Type type, HttpContentHeaders headers, MediaTypeHeaderValue mediaType)

// {

// base.SetDefaultContentHeaders(type, headers, mediaType);

// headers.ContentType = new MediaTypeHeaderValue("application/json");

// }

//}

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

var jsonpFormatter = new JsonpMediaTypeFormatter(config.Formatters.JsonFormatter);

config.Formatters.Insert(0, jsonpFormatter);

* Now go back to **ClientApplication/HtmlPage1.html** and make the following change to the code.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script src="Scripts/jquery-3.3.1.js"></script>

<script type="text/javascript">

$(document).ready(function () {

var ulEmployees = $('#ulEmployees');

$('#btn').click(function () {

$.ajax({

type: 'GET',

url: 'http://localhost:60607/api/Employees',

dataType: 'jsonp',

success: function (data) {

ulEmployees.empty();

$.each(data, function (index, val) {

var fullName = val.FirstName + ' ' + val.LastName;

ulEmployees.append('<li>' + fullName + '</li>')

});

}

});

});

$('#btnClear').click(function () {

ulEmployees.empty();

});

});

</script>

</head>

<body>

<input type="button" value="Get All Employees" id="btn" />

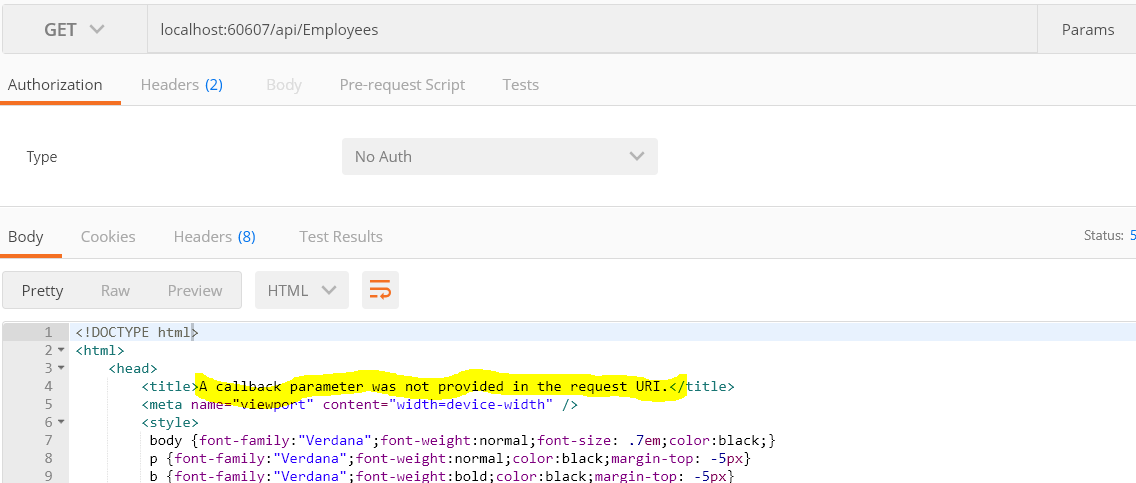
<input type="button" value="Clear" id="btnClear" />

<ul id="ulEmployees"></ul>

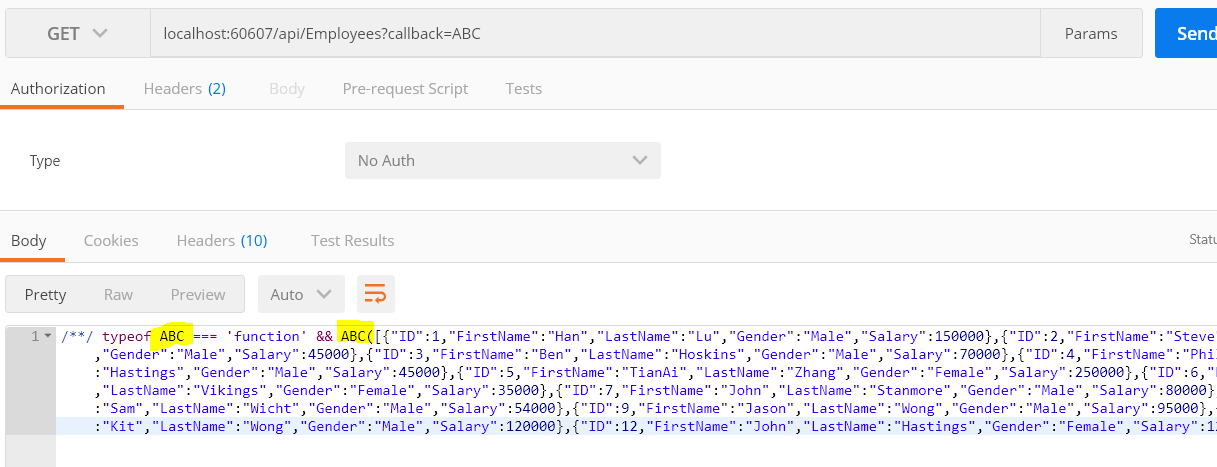
</body>

</html>

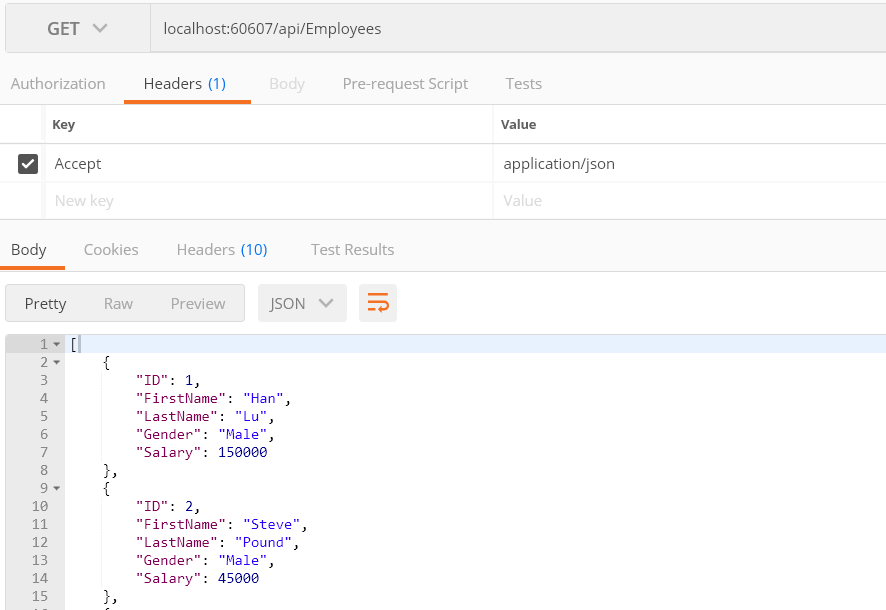
* Now run the API, and then View the HtmlPage1 in a browser.
* Now if you click on **Get All Employees** it should now work.
* Now if you go to Postman and then perform a GET Request to the API you will get the following error as the response data is in jsonp format.



* There are 2 ways you can fix this. You can wrap the data in a callback function by adding that in the query string as follows.



* Alternatively you can set the Accept Key to application/json in the Header.



**LECTURE 15 – CROSS ORIGIN RESOURCE SHARING ASPNET WEBAPI**

* We are now going to enable CORS to allow Cross Origin Ajax calls.
* Go to **EmployeeService/App\_Start/WebApiConfig.cs** and comment out the jsonp related code.
* Now go back to **ClientApplication/HtmlPage1.html** and change the dataTypes from jsonp to **json**.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script src="Scripts/jquery-3.3.1.js"></script>

<script type="text/javascript">

$(document).ready(function () {

var ulEmployees = $('#ulEmployees');

$('#btn').click(function () {

$.ajax({

type: 'GET',

url: 'http://localhost:60607/api/Employees',

dataType: 'json',

success: function (data) {

ulEmployees.empty();

$.each(data, function (index, val) {

var fullName = val.FirstName + ' ' + val.LastName;

ulEmployees.append('<li>' + fullName + '</li>')

});

}

});

});

$('#btnClear').click(function () {

ulEmployees.empty();

});

});

</script>

</head>

<body>

<input type="button" value="Get All Employees" id="btn" />

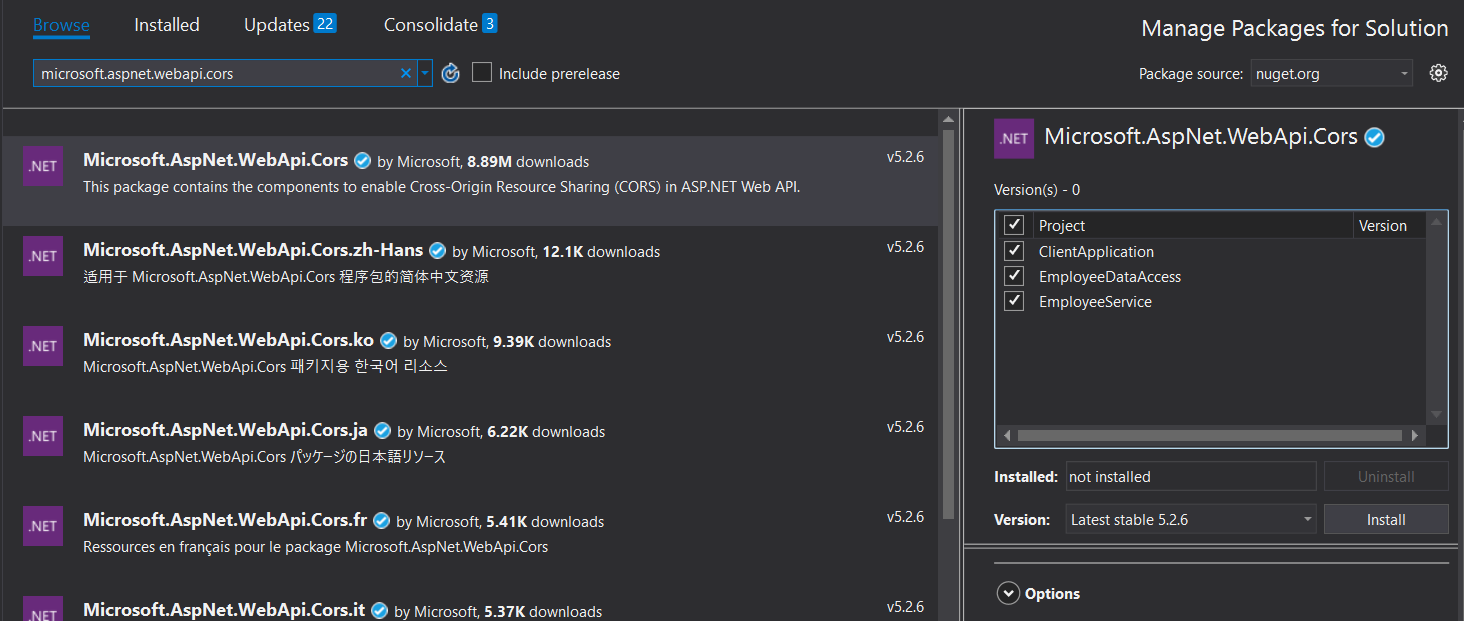
<input type="button" value="Clear" id="btnClear" />

<ul id="ulEmployees"></ul>

</body>

</html>

* We are going to enable CORS. First we need to install the **Microsoft.AspNet.WebApi.Cors** package.
* Go to **Tools > NuGet Package Manager > Manage Packages for Solution** and search for **Microsoft.AspNet.WebApi.Cors**, select all projects then click on **Install**.



* Now go to **EmployeeService/App\_Start/WebApiConfig.cs** and type in the using for the cors package we just downloaded.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Web.Http;

using Newtonsoft.Json.Serialization;

using System.Net.Http.Formatting;

using System.Net.Http.Headers;

using WebApiContrib.Formatting.Jsonp;

using System.Web.Http.Cors;

* Write the following code for the cors config. Ensure that the first argument in the EnableCorsAttribute is the **domain of HtmlPage1.html**.

namespace EmployeeService

{

//public class CustomJsonFormatter : JsonMediaTypeFormatter

//{

// public CustomJsonFormatter()

// {

// this.SupportedMediaTypes.Add(new System.Net.Http.Headers.MediaTypeHeaderValue("text/html"));

// }

// public override void SetDefaultContentHeaders(Type type, HttpContentHeaders headers, MediaTypeHeaderValue mediaType)

// {

// base.SetDefaultContentHeaders(type, headers, mediaType);

// headers.ContentType = new MediaTypeHeaderValue("application/json");

// }

//}

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

// This enable cors method takes in three parameters.

// FIRST: parameter is a comma separated list of the origns/domains

// that the API will take requests from.

// If you want to include all the websites then you just use "\*"

// SECOND: Is a comma separated list of all the headers that is supported

// by the resource. Use "\*" if you want to accept all headers.

// THIRD: A comma separated list of the methods that are supported by the

// resource (i.e. GET, POST ....)

EnableCorsAttribute cors = new EnableCorsAttribute("http://localhost:54324,http://www.google.com", "\*", "\*");

config.EnableCors(cors);

//var jsonpFormatter = new JsonpMediaTypeFormatter(config.Formatters.JsonFormatter);

//config.Formatters.Insert(0, jsonpFormatter);

* Now run API and view HtmlPage1.html in the browser.
* If you press the **Get All Employees** button now you should see all the names.
* We are also able to specify which Controllers or Methods we want to enable the CORS attribute on.
* If you want to enable CORS throughout the entire application then you need to only use **config.EnableCors()**.
* In the WebApiConfig.cs file make the following amendments.

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

// This enable cors method takes in three parameters.

// FIRST: parameter is a comma separated list of the origns/domains

// that the API will take requests from.

// If you want to include all the websites then you just use "\*"

// SECOND: Is a comma separated list of all the headers that is supported

// by the resource. Use "\*" if you want to accept all headers.

// THIRD: A comma separated list of the methods that are supported by the

// resource (i.e. GET, POST ....)

//EnableCorsAttribute cors = new EnableCorsAttribute("http://localhost:54324,http://www.google.com", "\*", "\*");

config.EnableCors();

* Now we are going to enable CORS only for the EmployeesController.
* Go to EmployeesController.cs and make the following amendments.

namespace EmployeeService.Controllers

{

[EnableCors("\*", "\*", "\*")]

public class EmployeesController : ApiController

{

public HttpResponseMessage Get(string gender = "all")

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

switch (gender.ToLower())

{

case "all":

return Request.CreateResponse(HttpStatusCode.OK, entities.Employees.ToList());

case "male":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "male").ToList());

case "female":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "female").ToList());

default:

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, "Value for gender must be All, Mall or Female. " + gender + " is invalid");

}

}

}

* To disable cors on certain methods use the following attributes.

namespace EmployeeService.Controllers

{

[EnableCors("\*", "\*", "\*")]

public class EmployeesController : ApiController

{

[DisableCors]

public HttpResponseMessage Get(string gender = "all")

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

switch (gender.ToLower())

{

case "all":

return Request.CreateResponse(HttpStatusCode.OK, entities.Employees.ToList());

case "male":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "male").ToList());

case "female":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "female").ToList());

default:

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, "Value for gender must be All, Mall or Female. " + gender + " is invalid");

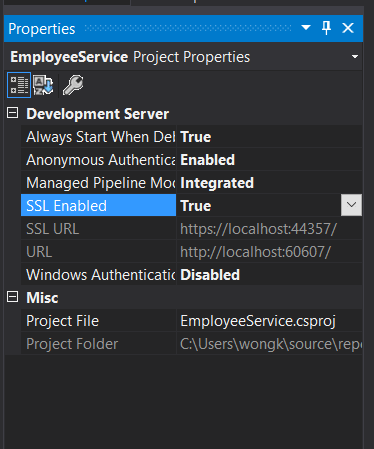
}

}

}

**LECTURE 16 – ENABLE SSL IN VISUAL STUDIO DEVELOPMENT SERVER**

* We are now going to implement SSL (https).
* In the Solution Explorer, select the **EmployeeService** project and press F4.
* Set SSL Enabled to **True** and then an SSL URL should appear.



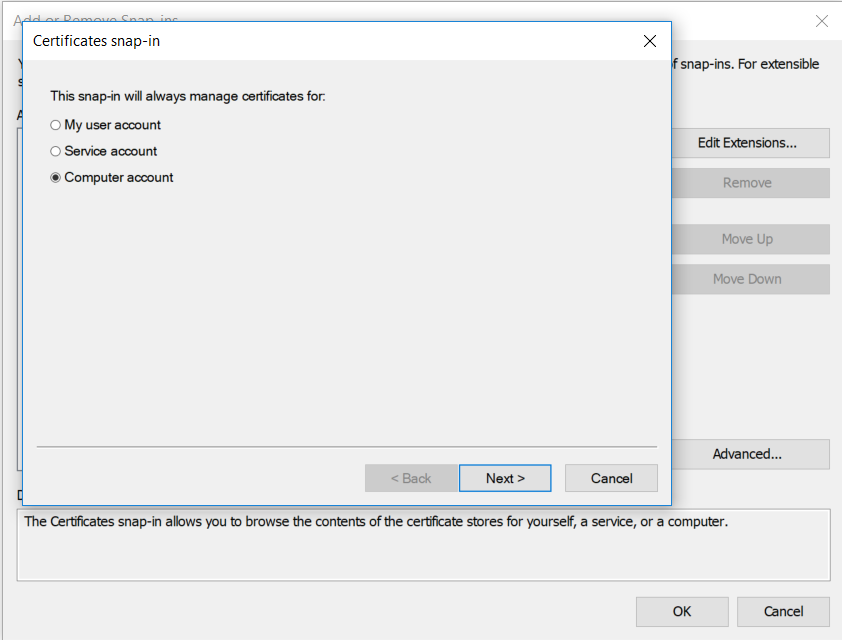
* **To enable trust for the certificate issued by Visual Studio**, launch the **run** command on Windows.
* Type **mmc.exe** in the run window and press **OK**.



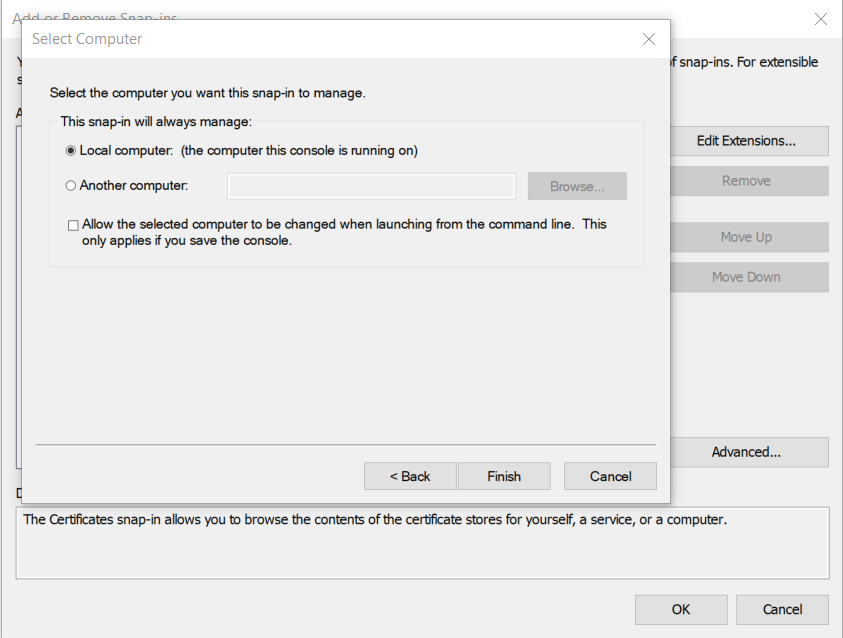
* On the Microsoft Management Console go to **File > Add/Remove Snap In.**



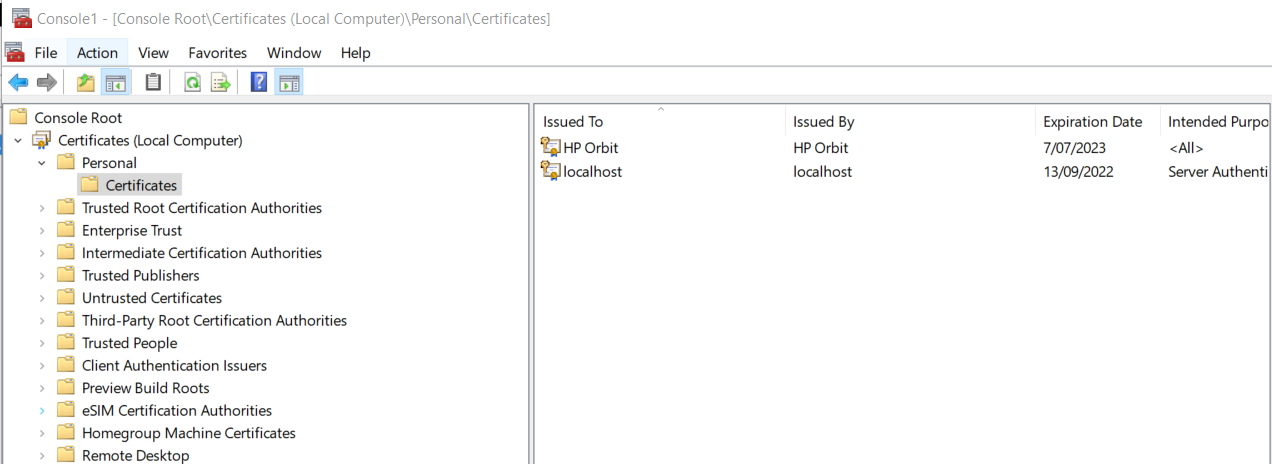
* Select **Certificates** then click **Add**.
* Select **Computer account** and click on **Next**.



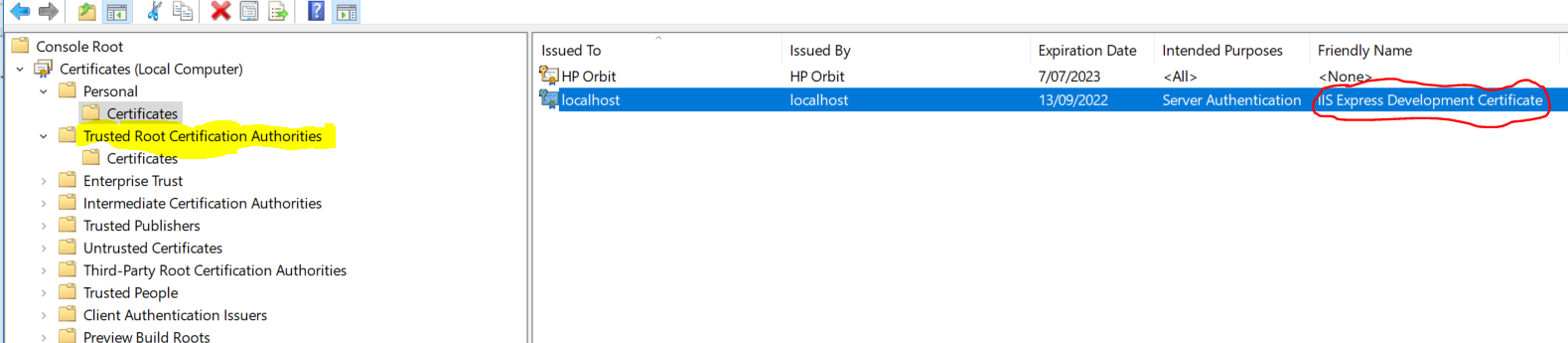
* Select **Local Computer** and click **Finish**.



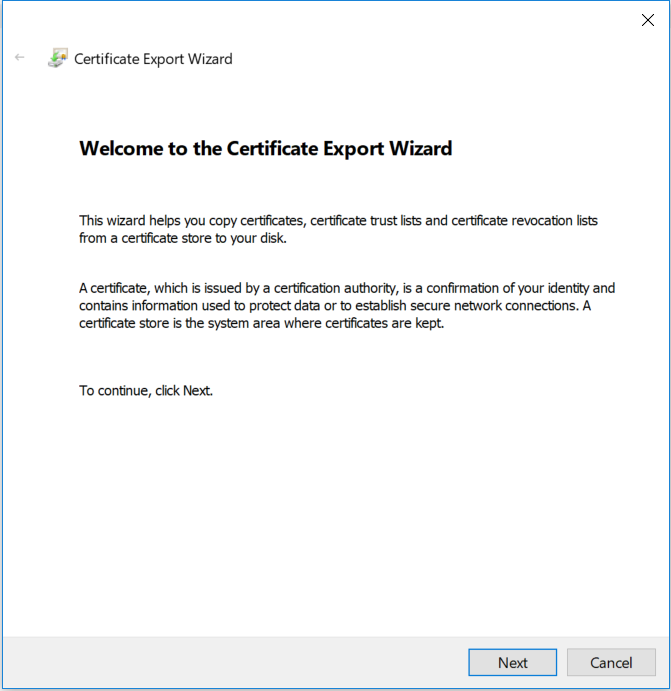
* Then click on **OK**.
* Now we have the Certificates of this local computer.
* Click on **Certificates > Personal > Certificates**.



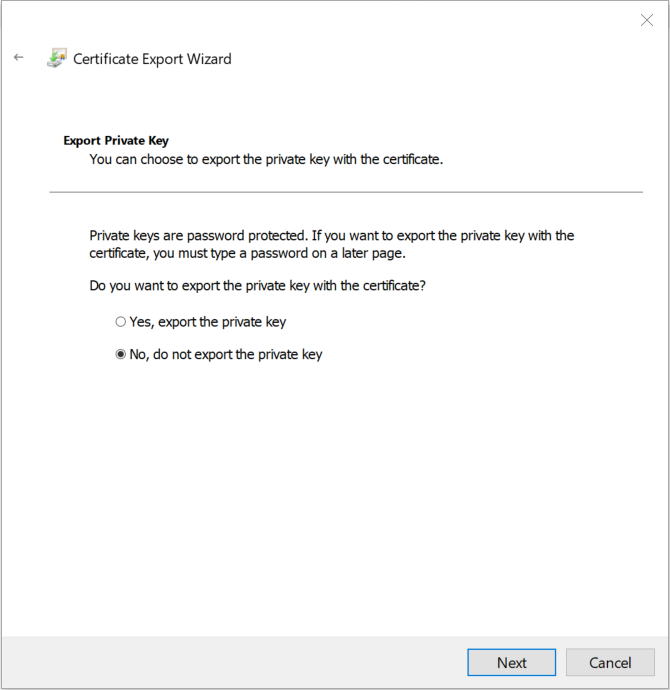
* The IIS Express Development Certificate is the SSL certificate that is distributed by Visual Studio. We will need to move this to the **Trusted Root Certification Authorities** folder to let the browser know that this certificate is ok.



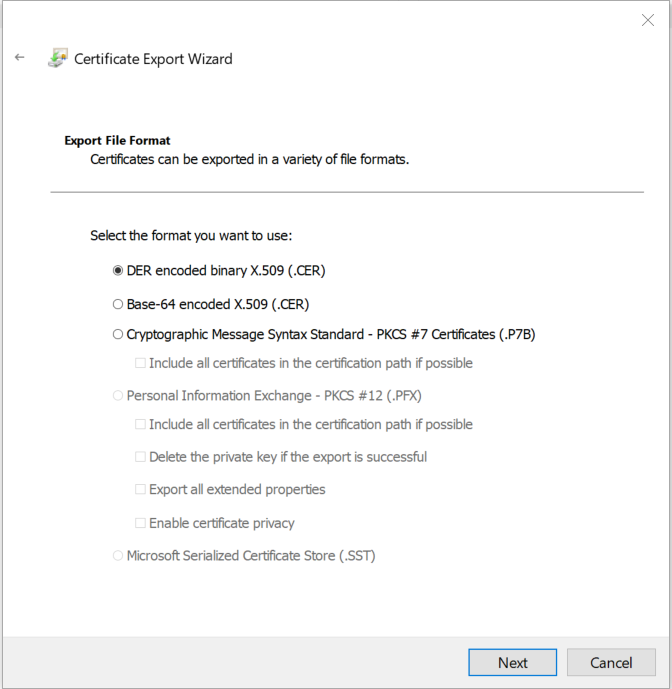
* To move the certificate into the trusted folder, we first must export the certificate into another folder and then import it from there into the trusted folder.
* Right-click on the IIS Express Development Certificate and select **All Task > Export** and it should launch the Certificate Export Wizard.



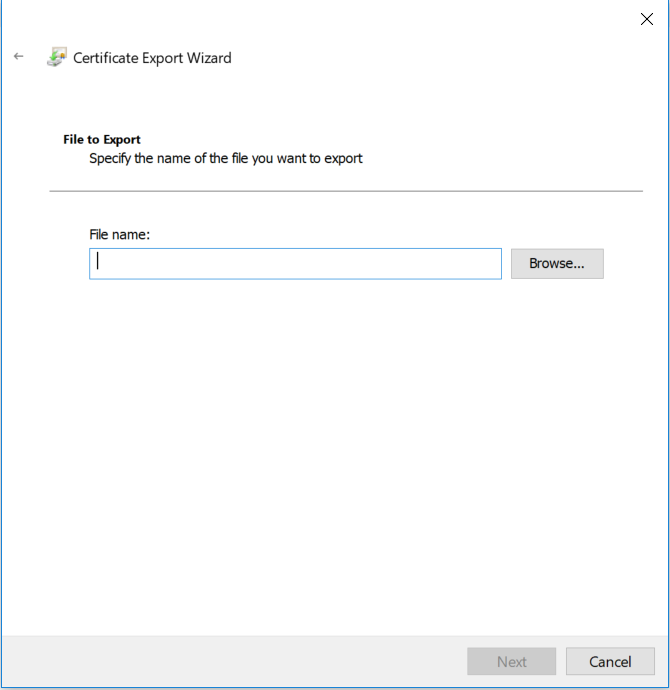
* Click **Next** then on the next screen select **No, do not export the private key** then click on **Next**.



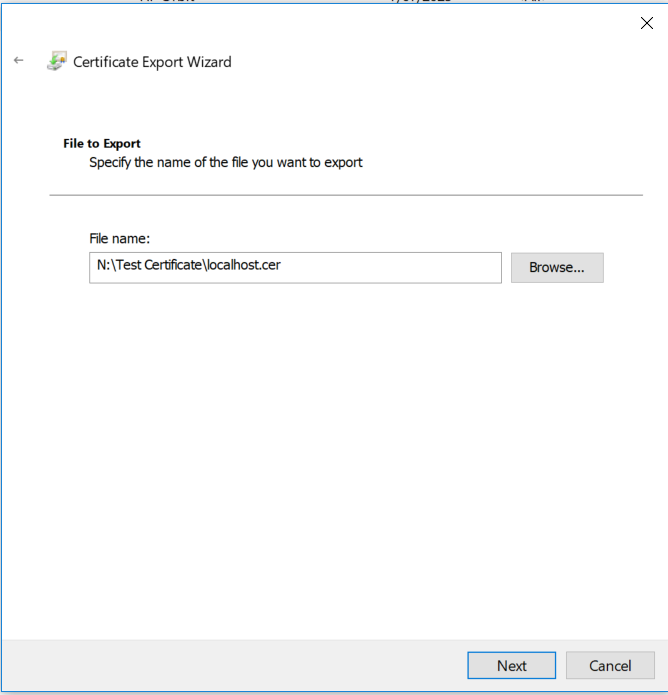
* Select **DER encoded binary X.509 (.CER)** and then click **Next**.



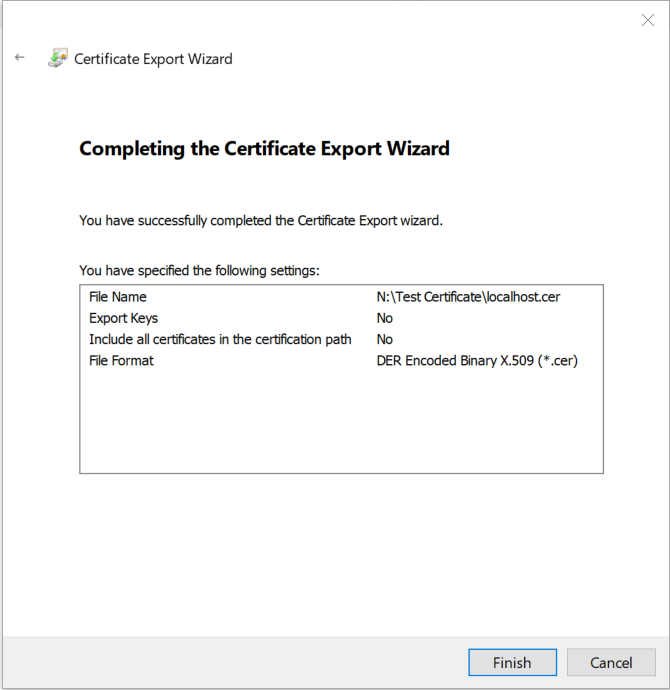
* Create a folder anywhere on you hard drive to save the certificate to.
* Click on **Browse** and then navigate to the folder you want to save the certificate to.



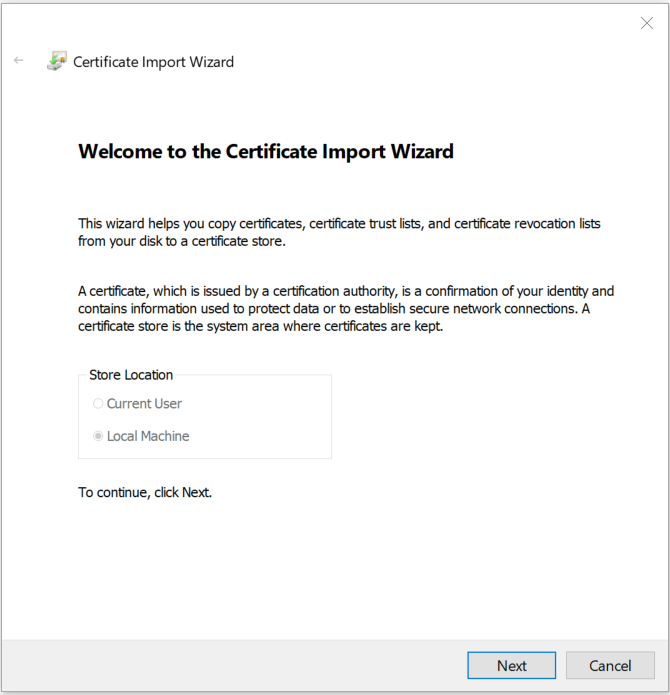
* Name the certificate **localhost** and click **Save**.
* Then click on **Next**.



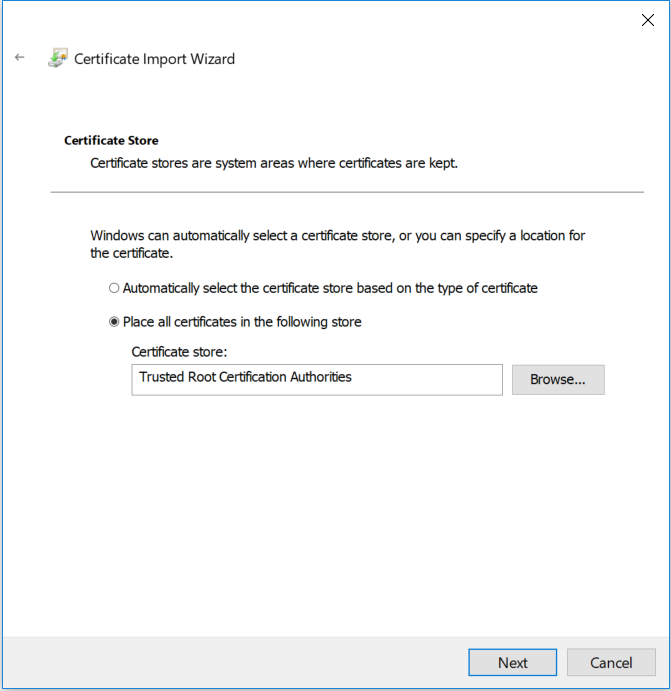
* Then click on **Finish**.



* Now we need to import the certificate.
* Go back to the Microsoft Management Console and right-click on **Trusted Root Certification Authorities** and select **All Tasks > Import**.
* Click **Next**.



* Browse for the certificate you exported earlier then click **Next**.
* We want to paste the certificate in the Trusted Root folder so click on **Next.**

S

* Then click on **Finish**.

**LECTURE 17 – ENABLE HTTPS**

* Now we will write code that automatically redirects the client to the https version if they try to navigate to the http version.
* On the EmployeeService project, right-click it and select **Add > Class.**
* Name the file **RequiredHttpsAttribute**.
* Go to RequiredHttpsAttribute.cs and let it inherit from **AuthorizationFilterAttribute**.
* Add the using for **System.Web.Http.Filters**.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Http.Filters;

namespace EmployeeService

{

public class RequiredHttpsAttribute : AuthorizationFilterAttribute

{

}

}

* Add the using statement for **System.Net.Http**.
* Amend the code to look like the following.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Http.Controllers;

using System.Web.Http.Filters;

using System.Net.Http;

namespace EmployeeService

{

public class RequiredHttpsAttribute : AuthorizationFilterAttribute

{

public override void OnAuthorization(HttpActionContext actionContext)

{

if (actionContext.Request.RequestUri.Scheme != Uri.UriSchemeHttps)

{

actionContext.Response = actionContext.Request.CreateResponse(System.Net.HttpStatusCode.Found);

actionContext.Response.Content = new StringContent("<p>Use HTTPS instead of HTTP");

UriBuilder uriBuilder = new UriBuilder(actionContext.Request.RequestUri);

uriBuilder.Scheme = Uri.UriSchemeHttps;

uriBuilder.Port = 44357;

actionContext.Response.Headers.Location = uriBuilder.Uri;

}

else

{

base.OnAuthorization(actionContext);

}

}

}

}

* We now need to register this Filter.
* Go to WebApiConfig.cs and amend the code to the following.

namespace EmployeeService

{

//public class CustomJsonFormatter : JsonMediaTypeFormatter

//{

// public CustomJsonFormatter()

// {

// this.SupportedMediaTypes.Add(new System.Net.Http.Headers.MediaTypeHeaderValue("text/html"));

// }

// public override void SetDefaultContentHeaders(Type type, HttpContentHeaders headers, MediaTypeHeaderValue mediaType)

// {

// base.SetDefaultContentHeaders(type, headers, mediaType);

// headers.ContentType = new MediaTypeHeaderValue("application/json");

// }

//}

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API configuration and services

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

EnableCorsAttribute cors = new EnableCorsAttribute("\*", "\*", "\*");

config.EnableCors();

config.Filters.Add(new RequiredHttpsAttribute());

}

}

}

* Launch the API and then navigate to <http://localhost:60607/api/Employees>.
* When you do this you will automatically be redirected to the <https://localhost:44357> version which is the SSL enabled version.
* By default, when the api is redirected from http to https it does so with a content-type of text/plain. We want to change this to something else.
* Go back to **EmployeeService/RequiredHttpsAttribute.cs** and add the using for **System.Text.**
* Then make the following changes to the code.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Http.Controllers;

using System.Web.Http.Filters;

using System.Net.Http;

using System.Text;

namespace EmployeeService

{

public class RequiredHttpsAttribute : AuthorizationFilterAttribute

{

public override void OnAuthorization(HttpActionContext actionContext)

{

if (actionContext.Request.RequestUri.Scheme != Uri.UriSchemeHttps)

{

actionContext.Response = actionContext.Request.CreateResponse(System.Net.HttpStatusCode.Found);

actionContext.Response.Content = new StringContent("<p>Use HTTPS instead of HTTP", Encoding.UTF8, "text/html");

UriBuilder uriBuilder = new UriBuilder(actionContext.Request.RequestUri);

uriBuilder.Scheme = Uri.UriSchemeHttps;

uriBuilder.Port = 44357;

actionContext.Response.Headers.Location = uriBuilder.Uri;

}

else

{

base.OnAuthorization(actionContext);

}

}

}

}

* Now this SSL certification is enabled throughout the entire application. If we want to allow SSL certification for only some methods or controllers then comment out the **config.Filters.Add(new RequiredHttpsAttribute());** from the WebApiConfig.cs file and just add the attributes above the specific controllers/methods in your application.
* The following is how you would impolement this over a Controller.

namespace EmployeeService.Controllers

{

[EnableCors("\*", "\*", "\*")]

[RequiredHttps]

public class EmployeesController : ApiController

{

public HttpResponseMessage Get(string gender = "all")

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

switch (gender.ToLower())

{

case "all":

return Request.CreateResponse(HttpStatusCode.OK, entities.Employees.ToList());

case "male":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "male").ToList());

case "female":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "female").ToList());

default:

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, "Value for gender must be All, Mall or Female. " + gender + " is invalid");

}

}

}

* The following is how you would implement this over an action method.

namespace EmployeeService.Controllers

{

[EnableCors("\*", "\*", "\*")]

public class EmployeesController : ApiController

{

[RequiredHttps]

public HttpResponseMessage Get(string gender = "all")

{

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

switch (gender.ToLower())

{

case "all":

return Request.CreateResponse(HttpStatusCode.OK, entities.Employees.ToList());

case "male":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "male").ToList());

case "female":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "female").ToList());

default:

return Request.CreateErrorResponse(HttpStatusCode.BadRequest, "Value for gender must be All, Mall or Female. " + gender + " is invalid");

}

}

}

**LECTURE 18 – IMPLEMENTING BASIC AUTHENTICATION IN ASPNET WEBAPI**

* Now we are going to create authentication that allows you to access the male employees if you login as a male, and female employees if you log in as a female.
* Go to the SQL Server Management Studio and run the following query to populate some test data.

Create Table Users

(

Id int identity primary key,

Username nvarchar(100),

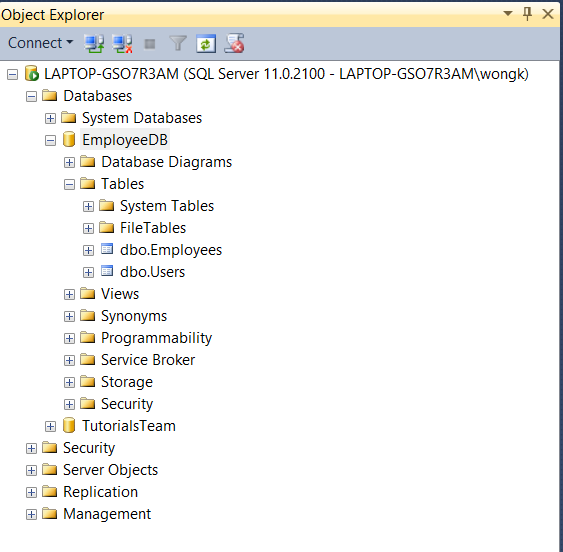
Password nvarchar(100)

)

Insert into Users values ('male', 'male')

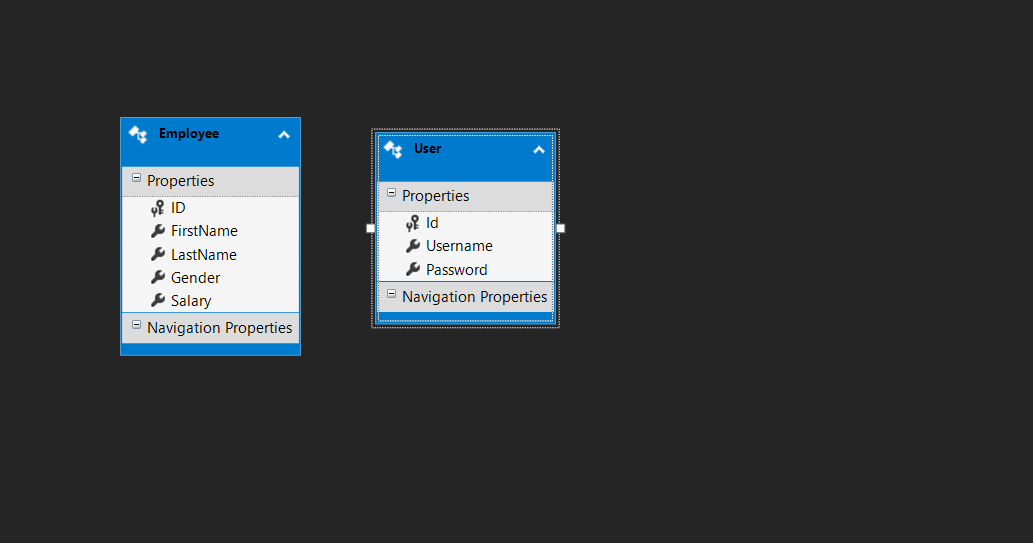
Insert into Users values ('female', 'female')

* You should now have a new Users Table in the EmployeeDB database.

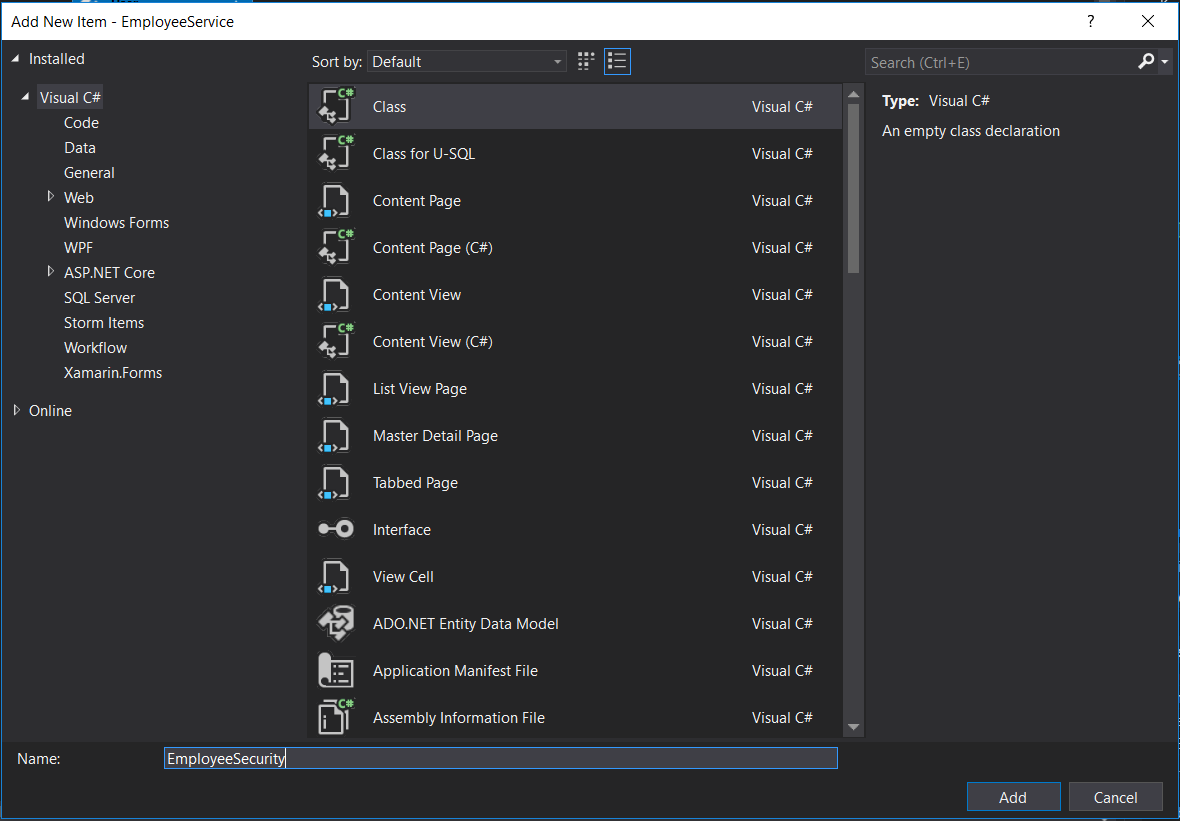


* Now we are going to update the Entity Data Model.
* Select on **EmployeeDataAccess/EmployeeDataModel.edmx** then go to the black part of the Diagram screen and right-click on it then select **Update Model from Database**.
* Select the **Users** table and then click on **Finish**.





* Now we want to create a custom class that will enable us to connect if we submit the correct username and password.
* Right click on **EmployeeService** and then select **Add > Class**.
* Name the class **EmployeeSecurity** then click on **Add**.



* Add the using statement for the **EmployeeDataAccess** to reference the Employee Data Model and the write the following boilerplate code for the Login method.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using EmployeeDataAccess;

namespace EmployeeService

{

public class EmployeeSecurity

{

public static bool Login(string username, string password)

{

}

}

}

* Write the following LINQ statement in the Login method.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using EmployeeDataAccess;

namespace EmployeeService

{

public class EmployeeSecurity

{

public static bool Login(string username, string password)

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Users.Any(user => user.Username.Equals(username, StringComparison.OrdinalIgnoreCase) && user.Password == password);

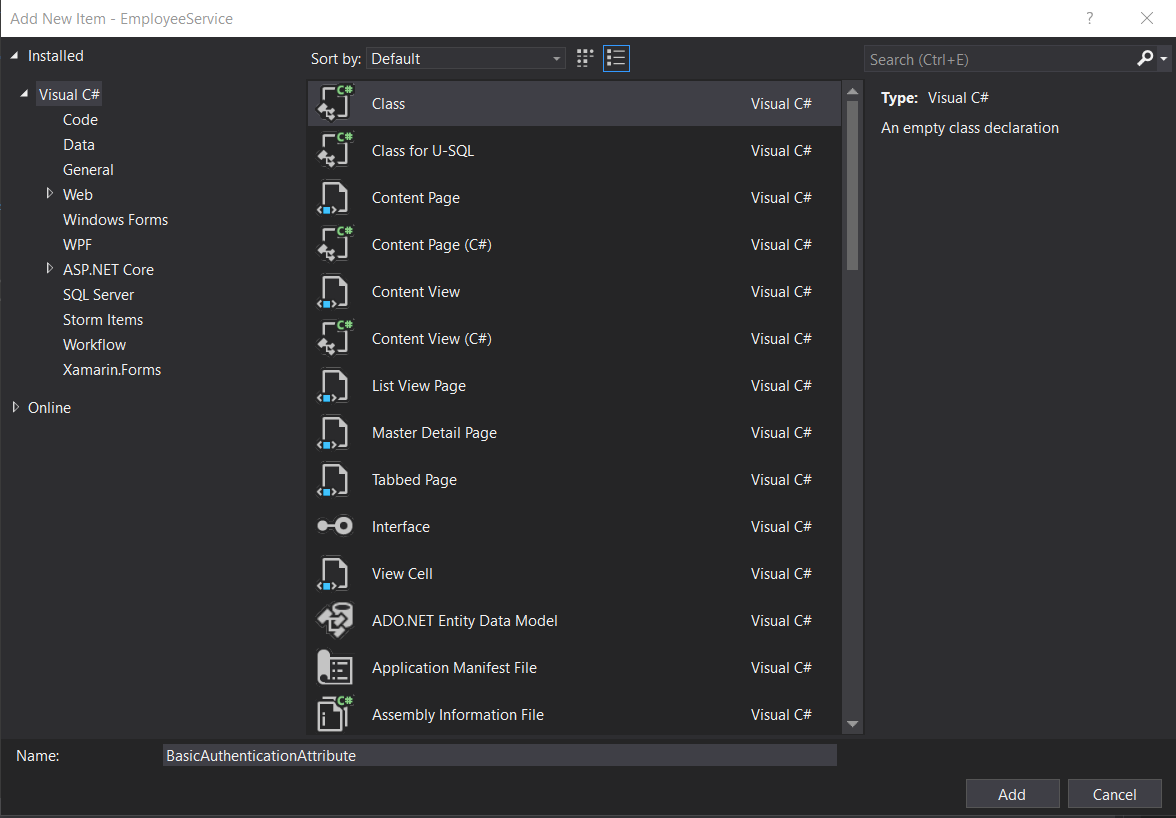
}

}

}

}

* Now we are going to create our own custom attribute.
* Right click on **EmployeeService** and select **Add > Class**.
* Name the class **BasicAuthenticationAttribute**.



* In the new class add the using statement for **System.Web.Http.Filters** and make the class inherit from **AuthorizationFilterAttribute**.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Http.Filters;

namespace EmployeeService

{

public class BasicAuthenticationAttribute : AuthorizationFilterAttribute

{

}

}

* We will need to override the base **onAuthorization** function on this Base Class so type in the following.

namespace EmployeeService

{

public class BasicAuthenticationAttribute : AuthorizationFilterAttribute

{

public override void OnAuthorization(HttpActionContext actionContext)

{

}

}

}

* Add the using statement for the following and make the following changes to the code.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Http.Controllers;

using System.Web.Http.Filters;

using System.Net.Http;

using System.Net;

using System.Text;

using System.Threading;

using System.Security.Principal;

namespace EmployeeService

{

public class BasicAuthenticationAttribute : AuthorizationFilterAttribute

{

public override void OnAuthorization(HttpActionContext actionContext)

{

if (actionContext.Request.Headers.Authorization == null)

{

actionContext.Response = actionContext.Request.CreateResponse(HttpStatusCode.Unauthorized);

}

else

{

string authenticationToken = actionContext.Request.Headers.Authorization.Parameter;

string decodedAuthenticationToken = Encoding.UTF8.GetString(Convert.FromBase64String(authenticationToken));

string[] usernamePasswordArray = decodedAuthenticationToken.Split(':');

string username = usernamePasswordArray[0];

string password = usernamePasswordArray[1];

if (EmployeeSecurity.Login(username, password))

{

Thread.CurrentPrincipal = new GenericPrincipal(new GenericIdentity(username), null);

}

else

{

actionContext.Response = actionContext.Request.CreateResponse(HttpStatusCode.Unauthorized);

}

}

}

}

}

* Go back to **EmployeesController** and add the following using statement make the following amendments to the code.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

using EmployeeDataAccess;

using System.Web.Http.Cors;

using System.Threading;

namespace EmployeeService.Controllers

{

[EnableCors("\*", "\*", "\*")]

public class EmployeesController : ApiController

{

public HttpResponseMessage Get(string gender = "all")

{

string username = Thread.CurrentPrincipal.Identity.Name;

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

switch (username.ToLower())

{

case "all":

return Request.CreateResponse(HttpStatusCode.OK, entities.Employees.ToList());

case "male":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "male").ToList());

case "female":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "female").ToList());

default:

return Request.CreateResponse(HttpStatusCode.BadRequest); }

}

}

* Much like the RequieHttpsAttirbute we can apply this custom BasicAuthenticationAttribute throughout our entire application by registering it as a filter in the WebApiConfig or we can use it on individual Controllers or Methods.
* We will enable the BasicAuthenticationAttribute over the Get method in the EmployeesController. This means that the Get Method will require basic authentication to be executed.

namespace EmployeeService.Controllers

{

[EnableCors("\*", "\*", "\*")]

public class EmployeesController : ApiController

{

[BasicAuthentication]

public HttpResponseMessage Get(string gender = "all")

{

string username = Thread.CurrentPrincipal.Identity.Name;

using(EmployeeDBEntities entities = new EmployeeDBEntities())

{

switch (username.ToLower())

{

case "all":

return Request.CreateResponse(HttpStatusCode.OK, entities.Employees.ToList());

case "male":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "male").ToList());

case "female":

return Request.CreateResponse(HttpStatusCode.OK,

entities.Employees.Where(e => e.Gender.ToLower() == "female").ToList());

default:

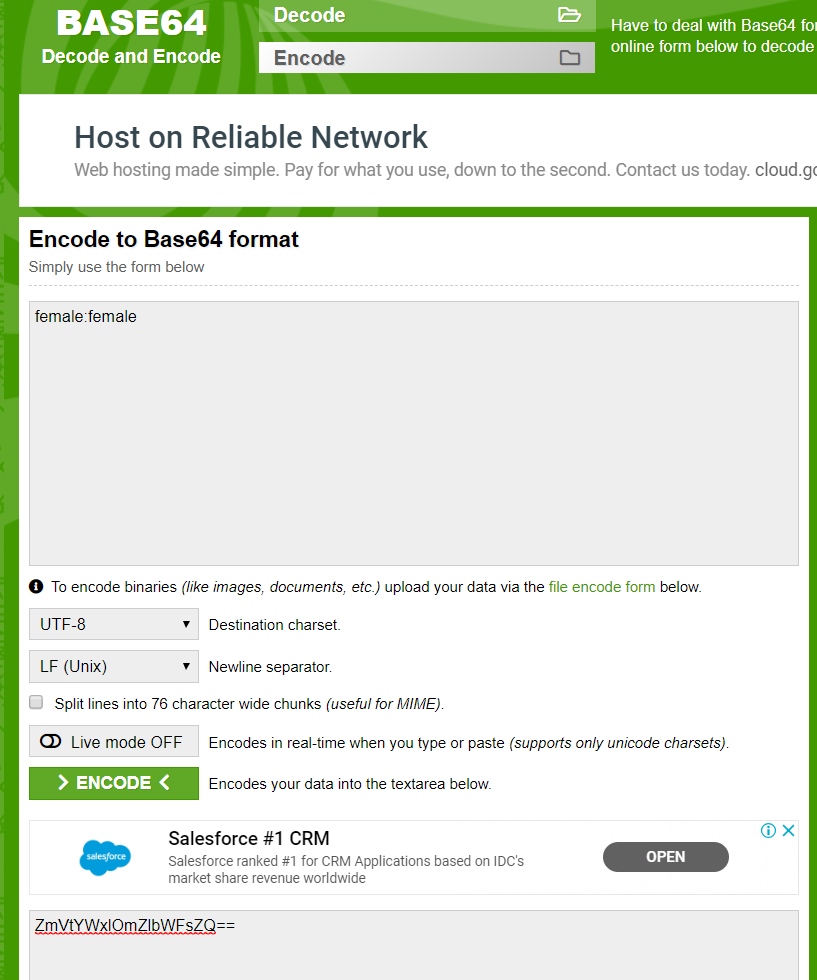
return Request.CreateResponse(HttpStatusCode.BadRequest);

}

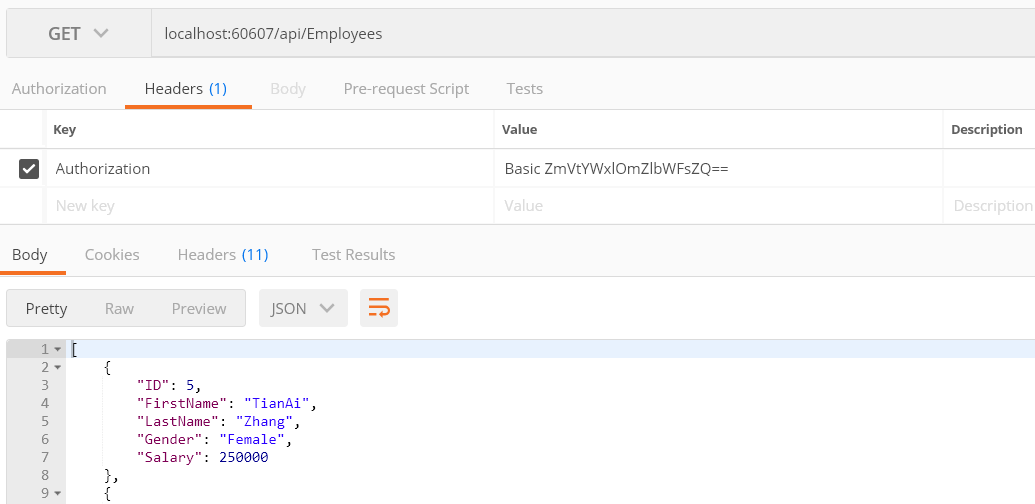
}

}

* If you run the API and try to go to localhost:60607/api/Employees, you will not be able to located anything. Either you get an error or you are returned nothing. This is because you need an Authorization Header along with the Request that also has the username and password in this format, female:female encoded in base64.



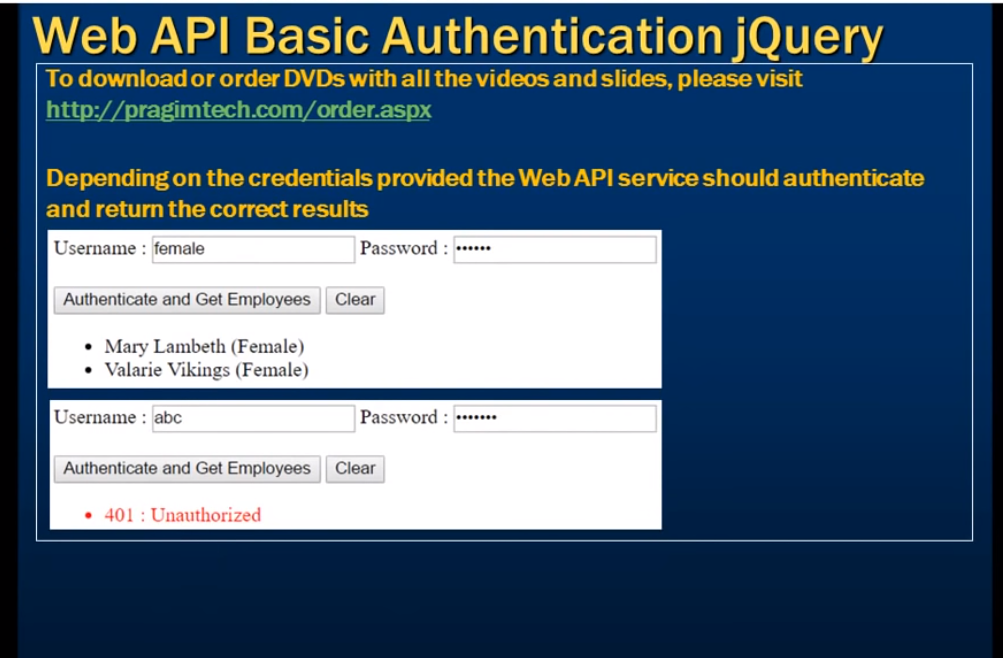
* In a web browser go to [**https://www.base64encode.org**](https://www.base64encode.org) and select **encode** then place your stirng in the textarea and hit encode. You should get a weird string in return. This is your base64 encoded string.
* Go to Postman and send the following GET Request with the following Authorization Header.



* You should now get a list of the female employees as that was what you logged in as.

**LECTURE 19 – CALL WEBAPI SERVICE WITH BASIC AUTHENTICATION USING JQUERY AJAX**

* We are going to try to pass basic authentication credentials to the API using an jquery ajax call.



* Go to **EmployeeService/Employees.html** and add the following code to add textboxes that get the username and password.

<body>

<div>

Username: <input type="text" value="" id="txtUsername"/>

Password: <input type="password" value="" id="txtPassword"/>

<br /><br />

<input type="button" id="btn" value="Authenticate and Get Employees" />

<input id="btnClear" type="button" value="Clear" />

<input type="button" value="Get All Employees" id="btn" />

<input type="button" value="Clear" id="btnClear" />

<ul id="ulEmployees"></ul>

</div>

</body>

</html>

* Make the following changes to the jQuery code in Employees.html.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script src="Scripts/jquery-3.3.1.js"></script>

<script type="text/javascript">

$(document).ready(function () {

var ulEmployees = $('#ulEmployees');

$('#btn').click(function () {

var username = $('#txtUsername').val();

var password = $('#txtPassword').val();

$.ajax({

type: 'GET',

url: 'api/Employees',

dataType: 'json',

headers: {

// btoa is a function that converts a string to Base64

'Authorization' : 'Basic ' + btoa(username + ':' + password)

},

success: function (data) {

ulEmployees.empty();

$.each(data, function (index, val) {

var fullName = val.FirstName + ' ' + val.LastName;

ulEmployees.append('<li>' + fullName + ' (' + val.Gender + ')</li>')

});

},

complete: function (jqXHR) {

// jqXHR is jQuery Xml Header Response

if (jqXHR.status == '401') {

ulEmployees.empty();

ulEmployees.append('<li style=”color: red”>' + jqXHR.status + ' : ' + jqXHR.statusText + '</li>')

}

}

});

});

$('#btnClear').click(function () {

ulEmployees.empty();

});

});

</script>

</head>

<body>

<div>

Username: <input type="text" value="" id="txtUsername"/>

Password: <input type="password" value="" id="txtPassword"/>

<br /><br />

<input type="button" id="btn" value="Authenticate and Get Employees" />

<input id="btnClear" type="button" value="Clear" />

<ul id="ulEmployees"></ul>

</div>

</body>

</html>

* We will now use the same technique but with the ClientApplication with CORS.
* Copy all the code from **EmployeeService/Employees.html** and copy it into **ClientApplication/HtmlPage1.html**.
* Make the following changes to the code in HtmlPage1.html.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script src="Scripts/jquery-3.3.1.js"></script>

<script type="text/javascript">

$(document).ready(function () {

var ulEmployees = $('#ulEmployees');

$('#btn').click(function () {

var username = $('#txtUsername').val();

var password = $('#txtPassword').val();

$.ajax({

type: 'GET',

url: 'http://localhost:60607/api/Employees',

dataType: 'json',

headers: {

// btoa is a function that converts a string to Base64

'Authorization': 'Basic ' + btoa(username + ':' + password)

},

success: function (data) {

ulEmployees.empty();

$.each(data, function (index, val) {

var fullName = val.FirstName + ' ' + val.LastName;

ulEmployees.append('<li>' + fullName + ' (' + val.Gender + ')</li>')

});

},

**LECTURE 20 – TOKEN AUTHENTICATION**

**\*\*NOTE:** When creating a project where the Authentication is set to Individual User accounts an AccountController is created that has a bunch of methods that can be used by the API for authentication purposes. The handling of this User identification/Authorization is handled by the **ASP.NET Identity Framework**.

* What we want to do here is get the data from the table on the left and display it on a web page only for authenticated users. We also want to create a registration page for which new users can register on the site and then see the table on the site.



* The first thing we’ll do is create the table of data that we will use with this authentication application.
* Go to SQL Server and delete the dbo.Employees table in the EmployeeDB database if it still exists. We are going to create a new Employees table that will overwrite the old one.
* run the following script.

Use EmployeeDB

Go

Create table Employees

(

ID int primary key identity,

FirstName nvarchar(50),

LastName nvarchar(50),

Gender nvarchar(50),

Salary int

)

Go

Insert into Employees values ('Mark', 'Hastings', 'Male', 60000)

Insert into Employees values ('Steve', 'Pound', 'Male', 45000)

Insert into Employees values ('Ben', 'Hoskins', 'Male', 70000)

Insert into Employees values ('Philip', 'Hastings', 'Male', 45000)

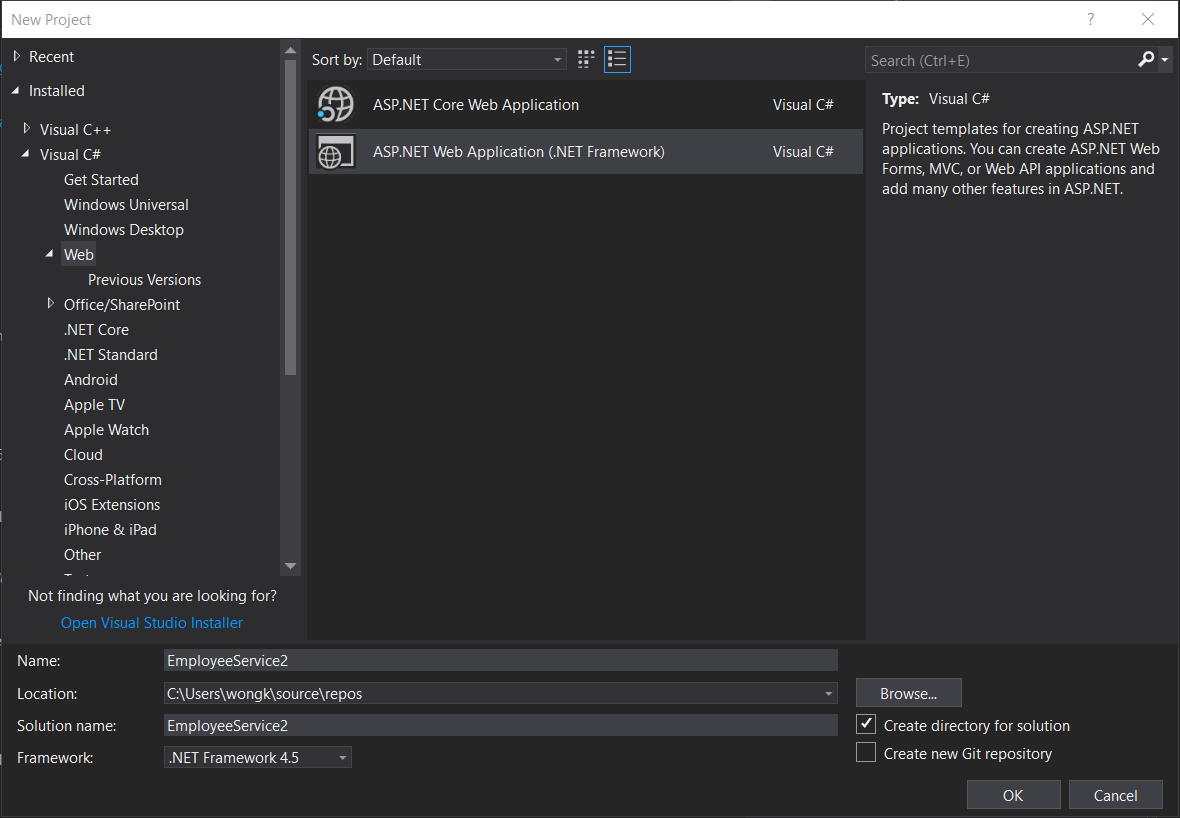
Insert into Employees values ('Mary', 'Lambeth', 'Female', 30000)

Insert into Employees values ('Valarie', 'Vikings', 'Female', 35000)

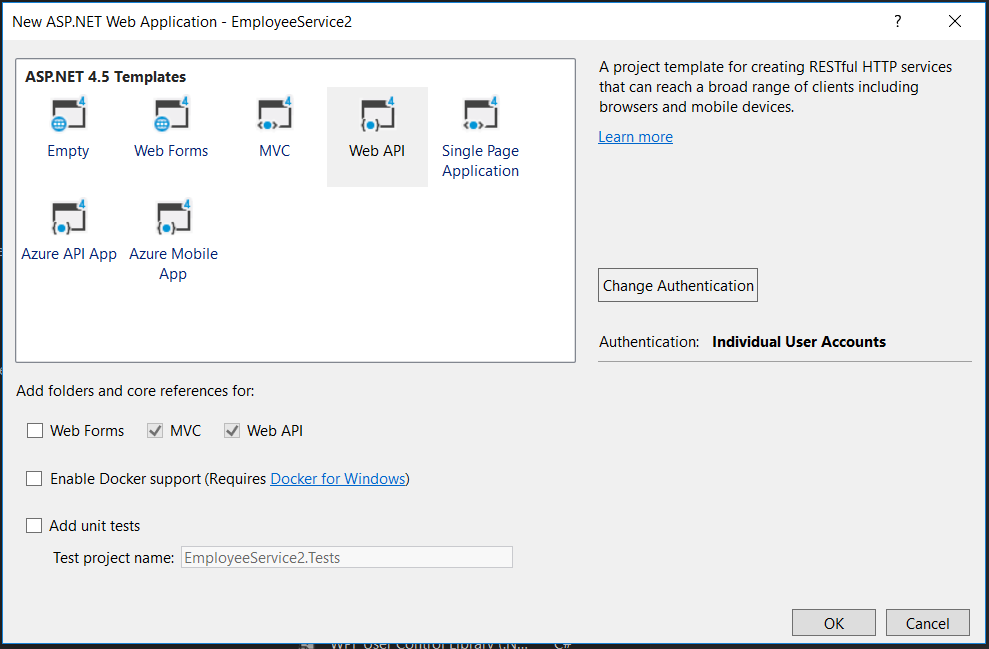
Insert into Employees values ('John', 'Stanmore', 'Male', 80000)

Go

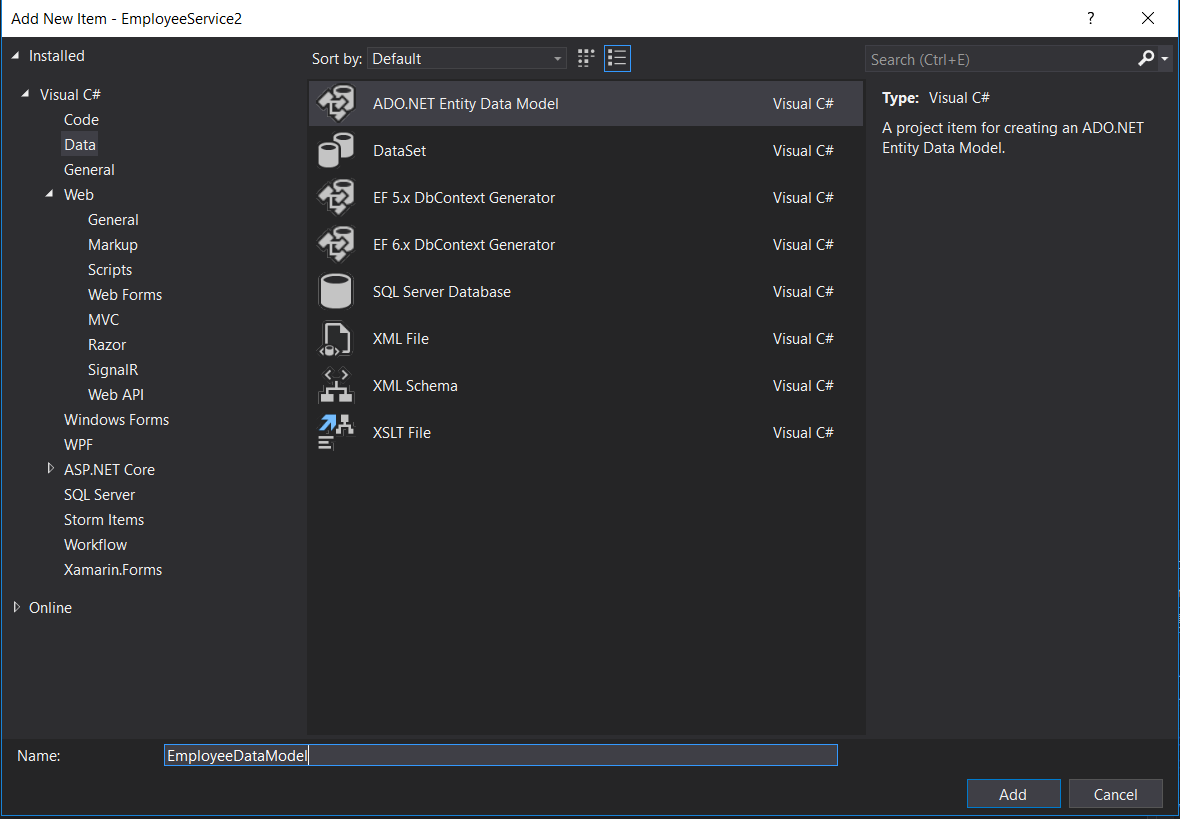
* we are going to start on an entirely new project for this.
* in Visual Studio go to **File > New > Project**.
* Go to **Visual C# > Web > ASP.NET Web Application (.NET Framework)**.
* Name the project **EmployeeService2** then click **OK**.



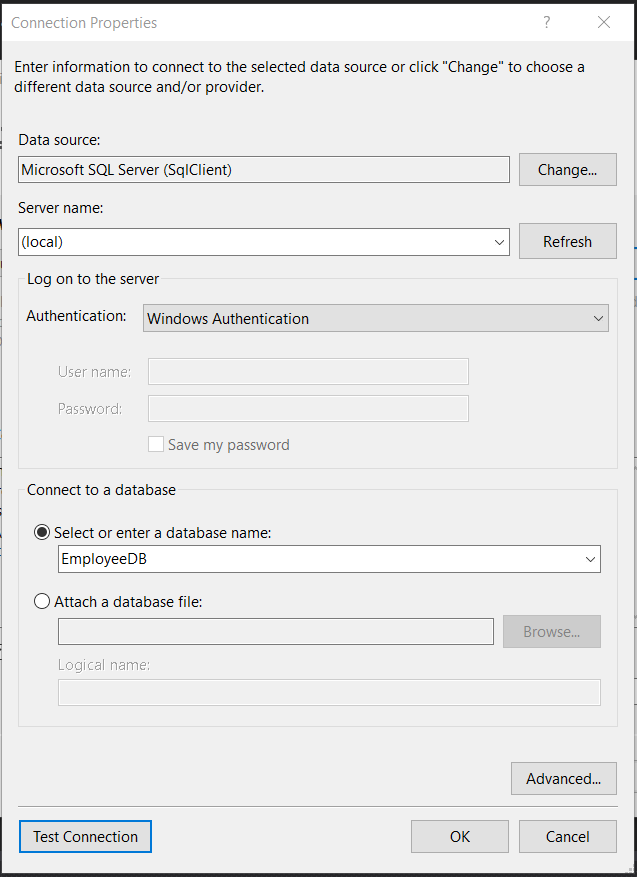
* Select Web API then ensure that Authentication is set to **Individual User Accounts**.
* If the Authentication is not set to Individual User Accounts then click on **Change Authentication** and select it, then click **OK**.



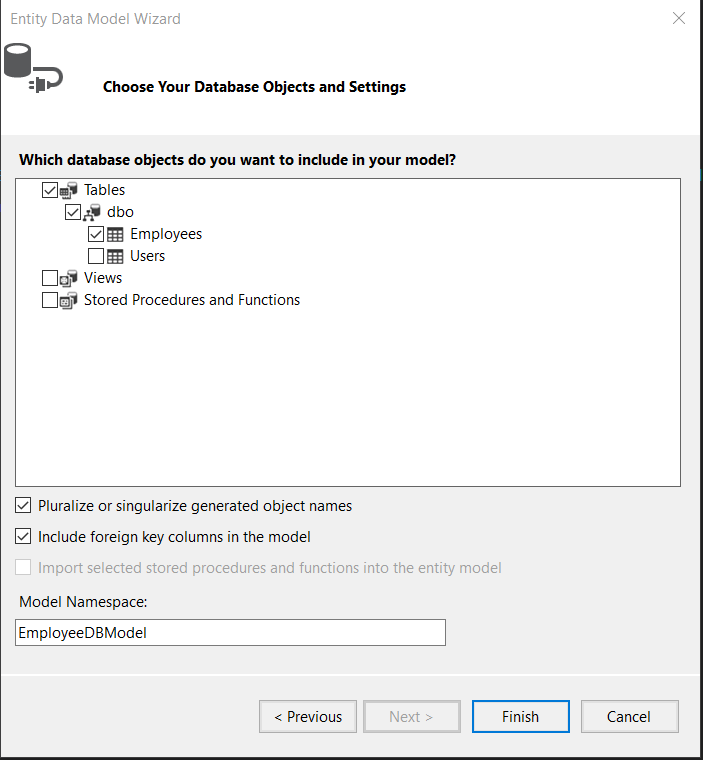
* Now we will add an ADO.NET Entity Data Model within the same project.
* Right click on the EmployeeService2 project and select **Add > New Item**.
* Select **Visual C# > Data > ADO.NET Entity Data Model** and name the data model **EmployeeDataModel**.



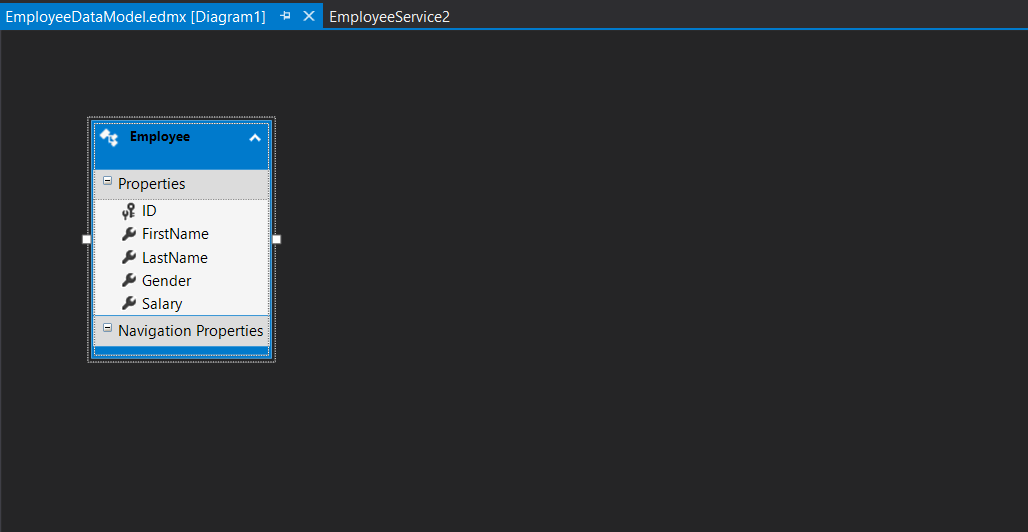
* Select **EF Designer from data** and then click **Next**.
* Click on **New Connection**.
* Ensure the Data source is set to **Microsoft SQL Server**.
* Ensure server name is set to **(local)**.
* Ensure the ‘Select or enter a database name’ is set to **EmployeeDB** then click on **OK**.



* Click on **Next**.
* Select the **Employees** table and leave the Model Namespace to **EmployeeDBModel** and click on **Finish**.



* The ADO.NET Entity Model should be created.



* Our next step is to add a Controller.
* Right click on the **Controllers** folder and select **Add > Controller**.
* Select **Web API 2 Controller – Empty** and then click on **Add**.
* Name the controller **EmployeesController**.
* Now write the following code in the EmployeesController.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

namespace EmployeeService2.Controllers

{

public class EmployeesController : ApiController

{

public IEnumerable<Employee> Get()

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.ToList();

}

}

}

}

* Currently the API does not require authorization to view the data. We will implement some code to force the API to require authorization before allowing clients to view the data in the EmployeesController.
* Add the following attribute to the EmployeesController.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

namespace EmployeeService2.Controllers

{

[Authorize]

public class EmployeesController : ApiController

{

public IEnumerable<Employee> Get()

{

using (EmployeeDBEntities entities = new EmployeeDBEntities())

{

return entities.Employees.ToList();

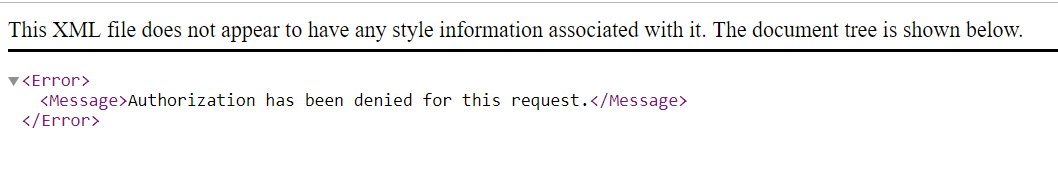
}

}

}

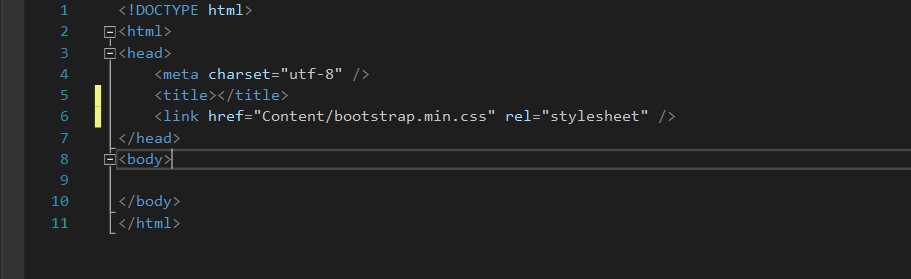
}

* If you build the API and try to access the api/Employees data you will get the following access denied message.



**LECTURE 21 – USER REGISTRATION**

* Now we are going to create a new user registration page.
* We are going to add a new HTML page.
* Right-click on EmployeeService2 and select **Add > HTML Page**.
* Name the page **Register.html**.
* In the **Content** folder drag and drop **bootstrap.min.css** into the head tag.



* Modify the code in Register.html to look as follows.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<link href="Content/bootstrap.min.css" rel="stylesheet" />

</head>

<body style="padding-top: 20px">

<div class="col-md-10 col-md-offset-1">

<div class="well">

<table class="table table-bordered">

<thead>

<tr class="success">

<td colspan="2">

New User Registration

</td>

</tr>

</thead>

<tbody>

<tr>

<td>Email</td>

<td>

<input type="text" id="txtEmail" placeholder="Email" />

</td>

</tr>

<tr>

<td>Password</td>

<td>

<input type="password" id="txtPassword" placeholder="Password" />

</td>

</tr>

<tr>

<td>Confirm Password</td>

<td>

<input type="password" id="txtConfirmPassword" placeholder="Confirm Password" />

</td>

</tr>

<tr class="success">

<td colspan="2">

<input type="button" id="btnRegister" class="btn btn-success" value="Register" />

</td>

</tr>

</tbody>

</table>

</div>

</div>

</body>

</html>

* Make the following changes.

</tbody>

</table>

<div class="modal fade" tabindex="-1" id="successModal" data-keyboard="false" data-backdrop="static">

<div class="modal-dialog modal-md">

<div class="modal-content">

<div class="modal-header">

<button type="button" class="close" data-dismiss="modal">

&times;

</button>

<h4>Success</h4>

</div>

<div class="modal-body">

<h2>Registration Successful</h2>

</div>

<div class="modal-footer">

<button type="button" data-dismiss="modal" class="btn btn-success">

Close

</button>

</div>

</div>

</div>

</div>

</div>

</div>

</body>

</html>

* In order to implement modal we need to write some jquery.
* First drag the jquery file and bootstrap javascript file right before the ending body tag. Both files are found in the scripts folder.

</div>

</div>

</div>

</div>

</div>

<script src="Scripts/jquery-3.3.1.min.js"></script>

<script src="Scripts/bootstrap.min.js"></script>

</body>

</html>

* Write the following jquery code right under the jquery and bootstrap javascript scripts.

</div>

</div>

</div>

</div>

</div>

</div>

<script src="Scripts/jquery-3.3.1.min.js"></script>

<script src="Scripts/bootstrap.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#btnRegister').click(function () {

$('#successModal').modal('show');

});

});

</script>

</body>

</html>

* Make the following changes for the error alert.

<button type="button" data-dismiss="modal" class="btn btn-success">

Close

</button>

</div>

</div>

</div>

</div>

<div id="divError" class="alert alert-danger collapse">

<a id="linkClose" href="#" class="close">&times;</a>

<div id="divErrorText">

</div>

</div>

</div>

</div>

<script src="Scripts/jquery-3.3.1.min.js"></script>

<script src="Scripts/bootstrap.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#linkClose').click(function () {

$('#divError').hide('fade');

});

$('#btnRegister').click(function () {

$('#successModal').modal('show');

});

});

</script>

</body>

</html>

* What we want to do is harvest the data from the form and use the data in the Register method of the AccountController.
* Make the following changes to the code in **Register.html**.

<div id="divError" class="alert alert-danger collapse">

<a id="linkClose" href="#" class="close">&times;</a>

<div id="divErrorText">

</div>

</div>

</div>

</div>

<script src="Scripts/jquery-3.3.1.min.js"></script>

<script src="Scripts/bootstrap.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#linkClose').click(function () {

$('#divError').hide('fade');

});

$('#btnRegister').click(function () {

$.ajax({

url: '/api/account/register',

method: 'POST',

data: {

email: $('#txtEmail').val(),

password: $('#txtPassword').val(),

confirmPassword: $('#txtConfirmPassword').val()

},

success: function () {

$('#successModal').modal('show');

},

error: function (jqXHR) {

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

});

});

});

</script>

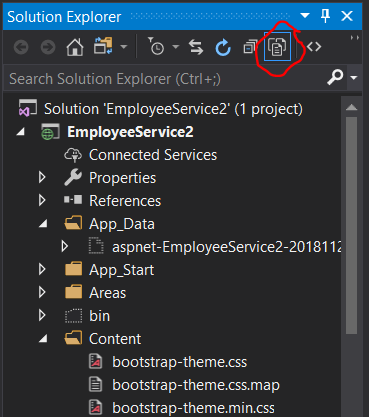
</body>

</html>

* Launch Register.html and register with a username and password.

**\*\*NOTE:** I’ve registered with [test1@gmail.com:Naruto#1](mailto:test1@gmail.com:Naruto#1). A second account was registered with [test2@gmail.com:Naruto@2](mailto:test2@gmail.com:Naruto@2). A third accounts was registered with test3@gmail.com:Naruto#3

* This registration (username and passwords) are all stored in a local mdf file that can be found in **EmployeeService/App\_Data**.
* To see this mdf file in the Solution Explorer click on the **Show All Files** button.



* Double click the mdf file to show the data file information in a local Server Explorer window.
* Then right-click on **Table/AspNetUsers** and select **Show Table Data** to show all Users that have just been registered on the application.

**LECTURE 22 – USING ASPNET IDENTITY WITH WEB API**

* In this lecture we’ll discuss customizing and using asp.net identity server with Web API.
* What we are going to do is change the name of the Identify Framework database file from **aspnet-EmployeeService2-\*\*timestamp\*\*.mdf** into something more meaningful. We change this name in the connectionString in the **Web.config**.
* Go to **EmployeeService2/Web.config**.
* Change the following into the following after.

<connectionStrings>

<add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet-EmployeeService2-20181125025635.mdf;Initial Catalog=aspnet-EmployeeService2-20181125025635;Integrated Security=True" providerName="System.Data.SqlClient" />

<add name="EmployeeDBEntities" connectionString="metadata=res://\*/EmployeeDataModel.csdl|res://\*/EmployeeDataModel.ssdl|res://\*/EmployeeDataModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=(local);initial catalog=EmployeeDB;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

Into

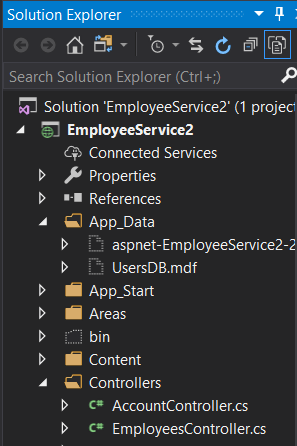
<connectionStrings>

<add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\UsersDB.mdf;Initial Catalog=UsersDB;Integrated Security=True" providerName="System.Data.SqlClient" />

<add name="EmployeeDBEntities" connectionString="metadata=res://\*/EmployeeDataModel.csdl|res://\*/EmployeeDataModel.ssdl|res://\*/EmployeeDataModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=(local);initial catalog=EmployeeDB;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

* Build the solution again and Register a new user.
* The name of the mdf file should have changed.



* Now what we are going to do is change the Data Source of the connectionString of the DefaultConnection so that the Users database is created in SQL Server instead of the App\_Data folder.
* Make the following changes to the connectionString in Web.config.

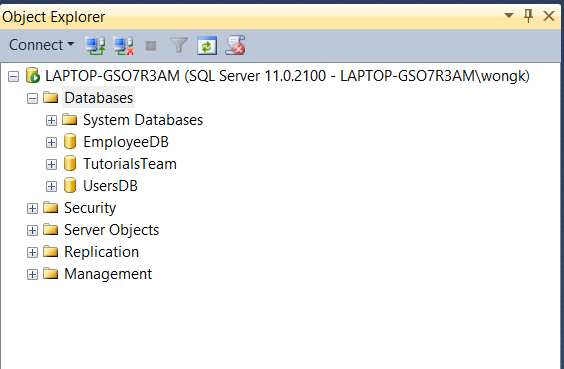
<connectionStrings>

<add name="DefaultConnection" connectionString="Data Source=(local);Initial Catalog=UsersDB;Integrated Security=True" providerName="System.Data.SqlClient" />

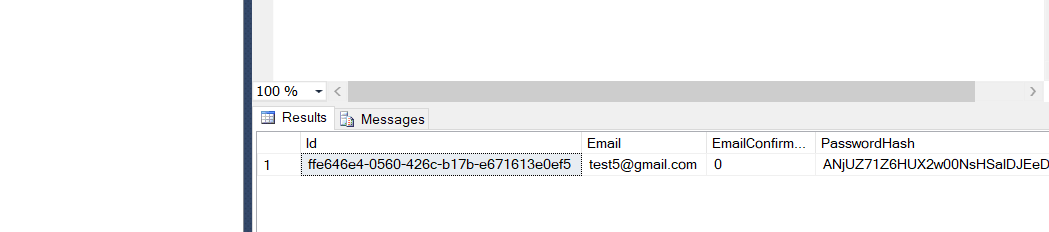
<add name="EmployeeDBEntities" connectionString="metadata=res://\*/EmployeeDataModel.csdl|res://\*/EmployeeDataModel.ssdl|res://\*/EmployeeDataModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=(local);initial catalog=EmployeeDB;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

* Rebuild the solution and register another new user.
* Now go back to SQL Server Management Studio and Refresh the Databases folder.
* A new database called **UsersDB** should appear. This is your new database with the Users info created.



* Notice how the new database in SQL Server only has the newly created User registered on here. None of the previously registered Users are on here. They are all still located in the Databases File in the App\_Data folder of the application.



* What we are going to do now is point all the Data Files into an existing Database so that we can store the information in there instead of creating a new **UsersDB** database.
* The way you do this is by changing the name of the **Initial Catalog** database in the DefaultConnection connectionString of the Web.config file to an existing database in SQL Server.
* Make the following changes.

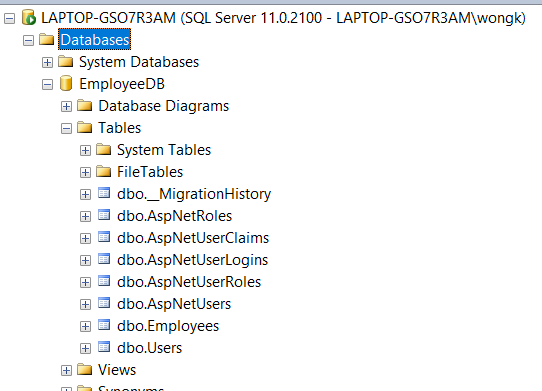
<connectionStrings>

<add name="DefaultConnection" connectionString="Data Source=(local);Initial Catalog=EmployeeDB;Integrated Security=True" providerName="System.Data.SqlClient" />

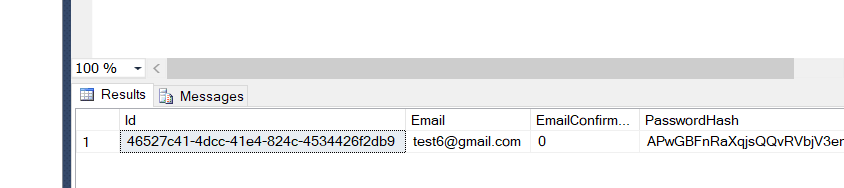
<add name="EmployeeDBEntities" connectionString="metadata=res://\*/EmployeeDataModel.csdl|res://\*/EmployeeDataModel.ssdl|res://\*/EmployeeDataModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=(local);initial catalog=EmployeeDB;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

* Build the API and register a new user.
* Go to SQL Server and Refresh the Databases folder.
* If you look down into the Tables folder you will notice the newly created Tables related to Users.

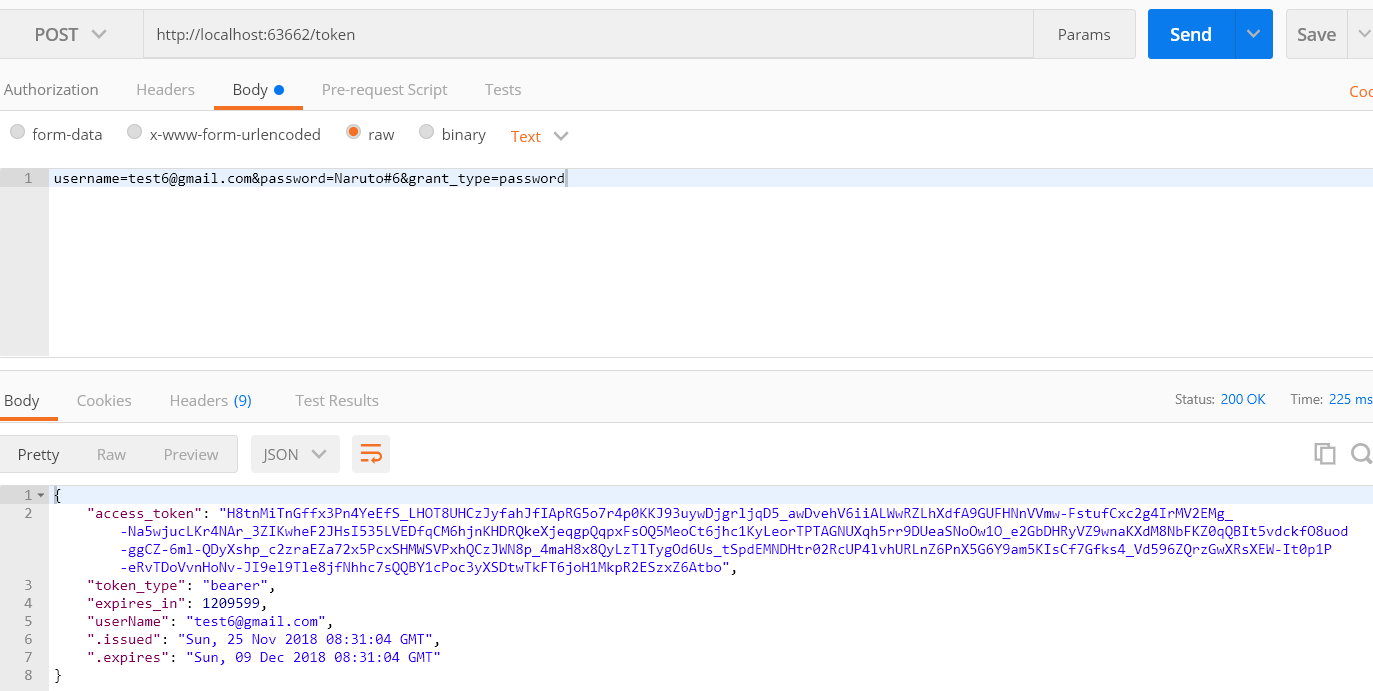


* Also the dbo.Users table will only have the information of the recently registered User and none of the older registered Users.

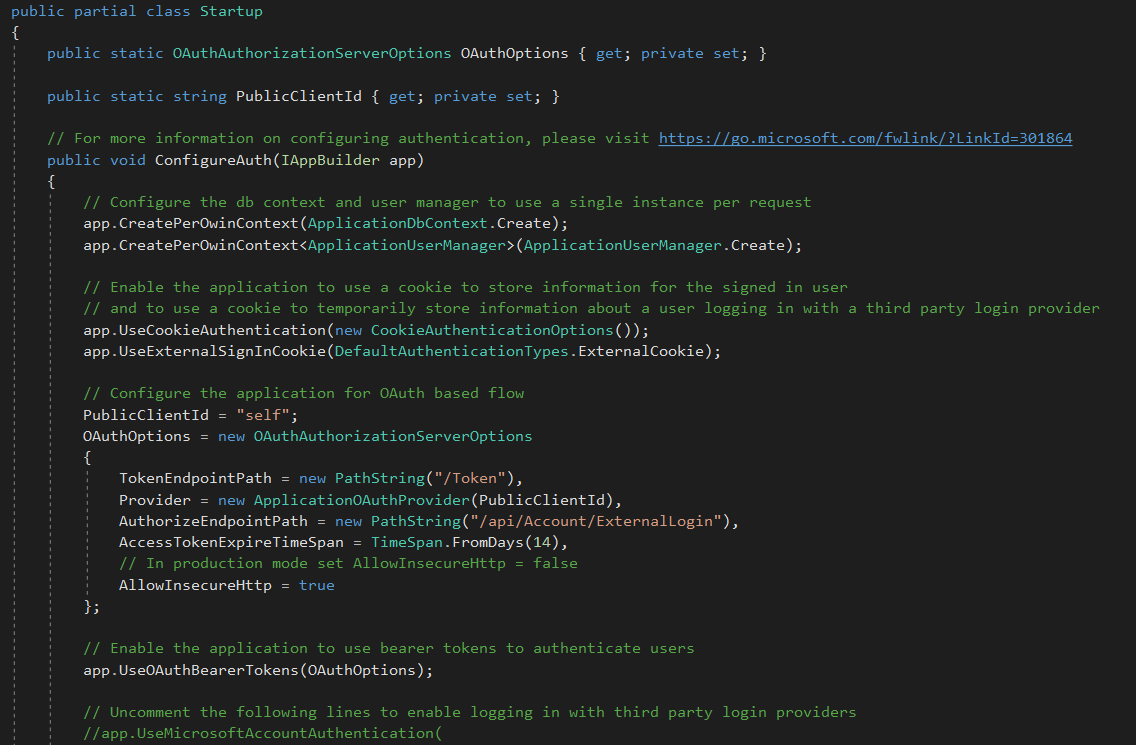


**LECTURE 23 – USING POSTMAN TO TEST ASPNET WEB API TOKEN BASED AUTHENTICATION**

* What we are going to do here is get the access token.
* Build the API and then run the following Request on Postman.



* You should receive the above access\_token in the Body Response.
* If you want to know what code creates this token go to **EmployeeService2/App\_Start/Startup.Auth.cs.**



* Now if you want to change the time span of how long the token will last (we’ll change it so it lasts 1 hour), make the following changes.

// Configure the application for OAuth based flow

PublicClientId = "self";

OAuthOptions = new OAuthAuthorizationServerOptions

{

TokenEndpointPath = new PathString("/Token"),

Provider = new ApplicationOAuthProvider(PublicClientId),

AuthorizeEndpointPath = new PathString("/api/Account/ExternalLogin"),

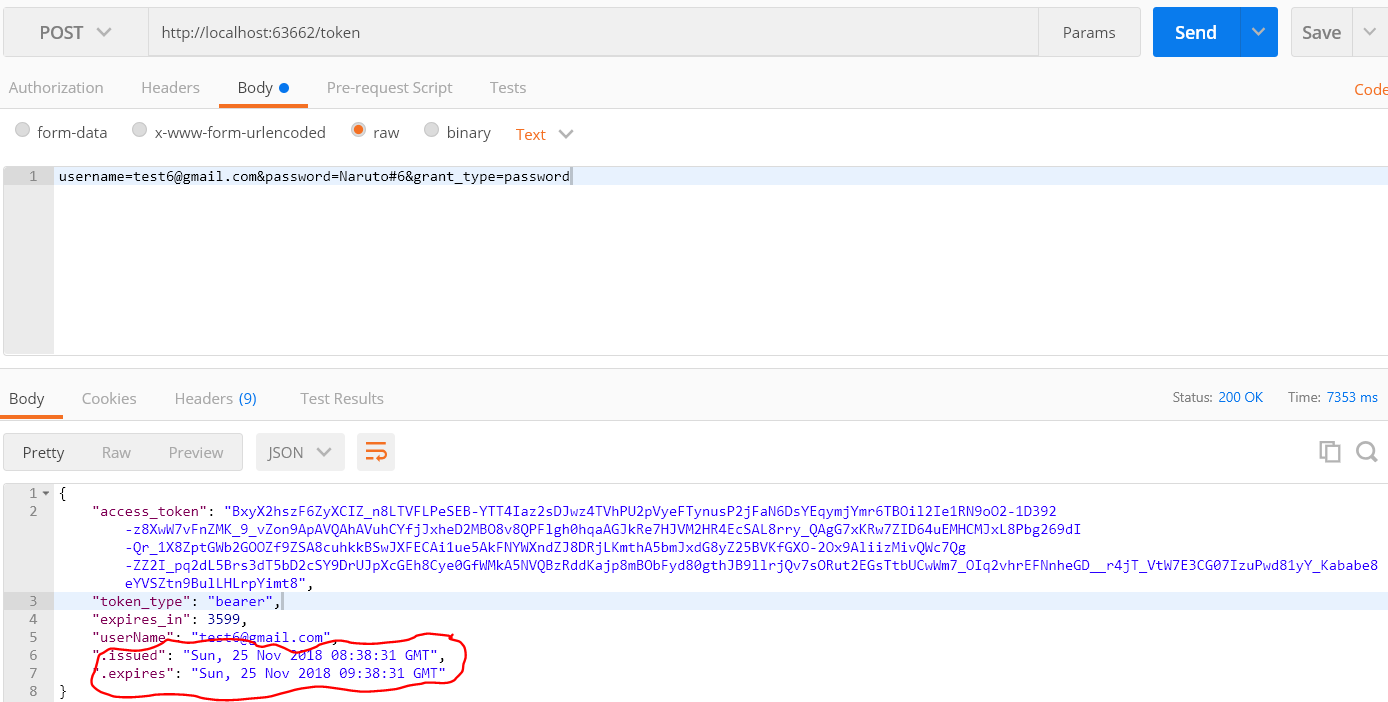
AccessTokenExpireTimeSpan = TimeSpan.FromHours(1),

// In production mode set AllowInsecureHttp = false

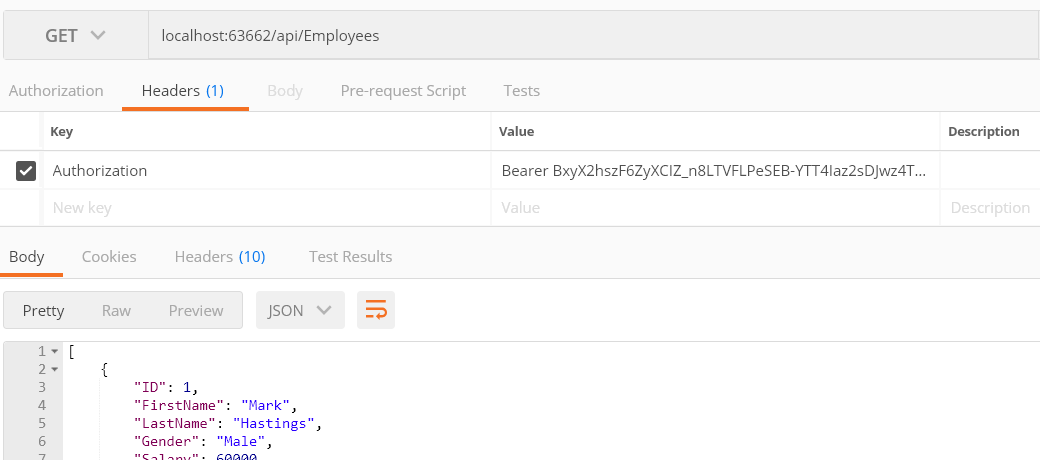
AllowInsecureHttp = true

};

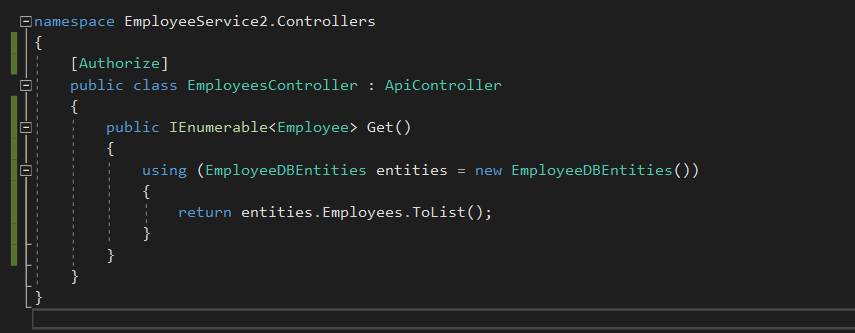
* Build the API again then reissue the POST Request as follows.
* You will notice that the token will now expire after 1 hour.



* Now send the following GET Request with the Header Authorization and the follow Bearer + Access Token included.



* The reason we needed to add the Authorization in the Header is because we have an **Authorize** attribute located on the EmployeesController.



**LECTURE 24 – LOGIN PAGE**

* Right-click on the EmployeeService2 project and select **Add > HTML Page** and name the page **Login.html**.
* Copy the code from Register.html and paste it in to Login.html.
* Make changes to the Login.html page until it looks like the following.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<link href="Content/bootstrap.min.css" rel="stylesheet" />

</head>

<body style="padding-top: 20px">

<div class="col-md-10 col-md-offset-1">

<div class="well">

<table class="table table-bordered">

<thead>

<tr class="success">

<td colspan="2">

Existing User Login

<a href="Register.html" class="btn btn-success pull-right">Register</a>

</td>

</tr>

</thead>

<tbody>

<tr>

<td>Username</td>

<td>

<input type="text" id="txtUsername" placeholder="Username" />

</td>

</tr>

<tr>

<td>Password</td>

<td>

<input type="password" id="txtPassword" placeholder="Password" />

</td>

</tr>

<tr class="success">

<td colspan="2">

<input type="button" id="btnLogin" class="btn btn-success" value="Login" />

</td>

</tr>

</tbody>

</table>

<div id="divError" class="alert alert-danger collapse">

<a id="linkClose" href="#" class="close">&times;</a>

<div id="divErrorText">

</div>

</div>

</div>

</div>

<script src="Scripts/jquery-3.3.1.min.js"></script>

<script src="Scripts/bootstrap.min.js"></script>

<script type="text/javascript">

//$(document).ready(function () {

// $('#linkClose').click(function () {

// $('#divError').hide('fade');

// });

// $('#btnRegister').click(function () {

// $.ajax({

// url: '/api/account/register',

// method: 'POST',

// data: {

// email: $('#txtEmail').val(),

// password: $('#txtPassword').val(),

// confirmPassword: $('#txtConfirmPassword').val()

// },

// success: function () {

// $('#successModal').modal('show');

// },

// error: function (jqXHR) {

// $('#divErrorText').text(jqXHR.responseText);

// $('#divError').show('fade');

// }

// });

// });

//});

</script>

</body>

</html>

* Make the following amendments to the jQuery code.

<script type="text/javascript">

$(document).ready(function () {

$('#linkClose').click(function () {

$('#divError').hide('fade');

});

$('#btnLogin').click(function () {

$.ajax({

url: '/token',

method: 'POST',

contentType: 'application/json',

data: {

username: $('#txtUsername').val(),

password: $('#txtPassword').val(),

grant\_type: 'password'

},

success: function (response) {

$('#divErrorText').text(JSON.stringify(response));

$('#divError').show('fade');

},

error: function (jqXHR) {

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

});

});

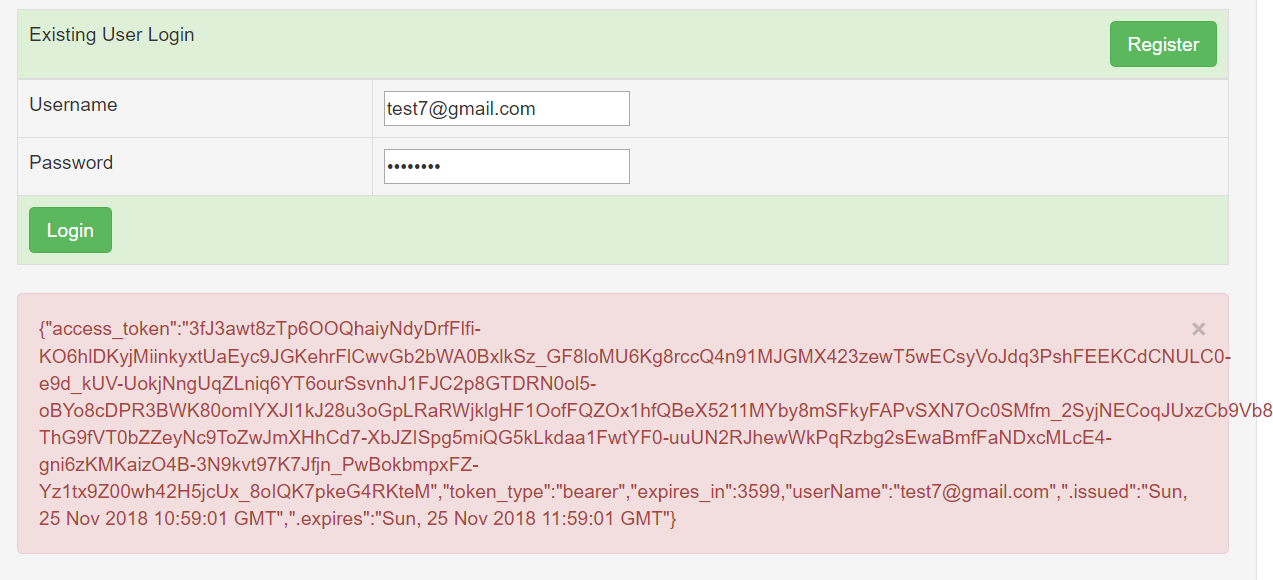
});

</script>

</body>

</html>

* Now if you build the Login.html page and you try to log in with an existing username and password, you should get the access token response.



* Now make the following amendment so that we store the access token in the sesstionStorage.

<script type="text/javascript">

$(document).ready(function () {

$('#linkClose').click(function () {

$('#divError').hide('fade');

});

$('#btnLogin').click(function () {

$.ajax({

url: '/token',

method: 'POST',

contentType: 'application/json',

data: {

username: $('#txtUsername').val(),

password: $('#txtPassword').val(),

grant\_type: 'password'

},

success: function (response) {

sessionStorage.setItem('accessToken', response.access\_token);

},

error: function (jqXHR) {

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

});

});

});

</script>

* Now let’s create a Data.html page in the project so that when the User logs in they will be redirected to this page where the data will be stored.
* Right-click on **EmployeeService2** and select **Add > HTML Page**.
* Name the page **Data.html**.
* Now go back to **Login.html**. We will write some code to redirect the User to the Data.html page once they log in.

<script type="text/javascript">

$(document).ready(function () {

$('#linkClose').click(function () {

$('#divError').hide('fade');

});

$('#btnLogin').click(function () {

$.ajax({

url: '/token',

method: 'POST',

contentType: 'application/json',

data: {

username: $('#txtUsername').val(),

password: $('#txtPassword').val(),

grant\_type: 'password'

},

success: function (response) {

sessionStorage.setItem('accessToken', response.access\_token);

window.location.href = "Data.html";

},

error: function (jqXHR) {

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

});

});

});

</script>

* What we are going to do now is add a **Login** button in the Register.html page so that if a User has already registered and wants to go to Login then they can be redirected there quickly.
* Go to Register.html and make the following amendments.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<link href="Content/bootstrap.min.css" rel="stylesheet" />

</head>

<body style="padding-top: 20px">

<div class="col-md-10 col-md-offset-1">

<div class="well">

<table class="table table-bordered">

<thead>

<tr class="success">

<td colspan="2">

New User Registration

<a href="Login.html" class="btn btn-success pull-right">Login</a>

</td>

</tr>

</thead>

**LECTURE 25 – BEARER TOKEN EXAMPLE**

* Now we are going to start creating our Data.html page.
* Copy all the code from Register.html and paste it in to the Data.html page.
* Amend the code in Data.html so that it looks like the following.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<link href="Content/bootstrap.min.css" rel="stylesheet" />

</head>

<body style="padding-top: 20px">

<div class="col-md-10 col-md-offset-1">

<div class="well">

<input type="button" id="btnLoadEmployees" class="btn btn-success" name="name" value="Load Employees" />

</div>

<div class="well hidden" id="divData">

<table id="tblData" class="table table-bordered">

<thead>

<tr>

<th>Id</th>

<th>First Name</th>

<th>Last Name</th>

<th>Gender</th>

<th>Salary</th>

</tr>

</thead>

<tbody id="tblBody"></tbody>

</table>

</div>

<div class="modal fade" tabindex="-1" id="errorModal" data-keyboard="false" data-backdrop="static">

<div class="modal-dialog modal-md">

<div class="modal-content">

<div class="modal-header">

<button type="button" class="close" data-dismiss="modal">

&times;

</button>

<h4 class="modal-title">Session Expired</h4>

</div>

<div class="modal-body">

<h2 class="modal-title">Please close this message to login again</h2>

</div>

<div class="modal-footer">

<button type="button" data-dismiss="modal" class="btn btn-success">

Close

</button>

</div>

</div>

</div>

</div>

<div id="divError" class="alert alert-danger collapse">

<a id="linkClose" href="#" class="close">&times;</a>

<div id="divErrorText">

</div>

</div>

</div>

<script src="Scripts/jquery-3.3.1.min.js"></script>

<script src="Scripts/bootstrap.min.js"></script>

<script type="text/javascript">

//$(document).ready(function () {

// $('#linkClose').click(function () {

// $('#divError').hide('fade');

// });

// $('#btnRegister').click(function () {

// $.ajax({

// url: '/api/account/register',

// method: 'POST',

// data: {

// email: $('#txtEmail').val(),

// password: $('#txtPassword').val(),

// confirmPassword: $('#txtConfirmPassword').val()

// },

// success: function () {

// $('#successModal').modal('show');

// },

// error: function (jqXHR) {

// $('#divErrorText').text(jqXHR.responseText);

// $('#divError').show('fade');

// }

// });

// });

//});

</script>

</body>

</html>

* Uncomment the jQuery.
* Make amendments to the jQuery so that it looks like the following.

<div id="divError" class="alert alert-danger collapse">

<a id="linkClose" href="#" class="close">&times;</a>

<div id="divErrorText">

</div>

</div>

</div>

</div>

<script src="Scripts/jquery-3.3.1.min.js"></script>

<script src="Scripts/bootstrap.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

$('#linkclose').click(function () {

$('#diverror').hide('fade');

});

$('#btnLoadEmployees').click(function () {

$.ajax({

url: '/api/Employees',

method: 'GET',

headers: {

'Authorization' : 'Bearer ' + sessionStorage.getItem('accessToken')

},

success: function (data) {

$('#divData').removeClass('hidden');

$.each(data, function (index, value) {

var row = $('<tr><td>' + value.ID + '</td><td>'

+ value.FirstName + '</td><td>'

+ value.LastName + '</td><td>'

+ value.Gender + '</td><td>'

+ value.Salary + '</td></tr>');

});

},

error: function (jqXHR) {

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

});

});

});

</script>

</body>

</html>

* Add the following code to redirect the User to the Login Page if they try to Load the Employees even when they’re not here.

<script type="text/javascript">

$(document).ready(function () {

if (sessionStorage.getItem('accessToken') == null) {

window.location.href = 'Login.html';

}

$('#linkclose').click(function () {

$('#diverror').hide('fade');

});

$('#btnLoadEmployees').click(function () {

$.ajax({

url: '/api/Employees',

method: 'GET',

headers: {

'Authorization' : 'Bearer ' + sessionStorage.getItem('accessToken')

},

* Write the following code to enable the display of the table data.

<script type="text/javascript">

$(document).ready(function () {

if (sessionStorage.getItem('accessToken') == null) {

window.location.href = 'Login.html';

}

$('#linkclose').click(function () {

$('#diverror').hide('fade');

});

$('#btnLoadEmployees').click(function () {

$.ajax({

url: '/api/Employees',

method: 'GET',

headers: {

'Authorization' : 'Bearer ' + sessionStorage.getItem('accessToken')

},

success: function (data) {

$('#divData').removeClass('hidden');

$('#tblBody').empty();

$.each(data, function (index, value) {

var row = $('<tr><td>' + value.ID + '</td><td>'

+ value.FirstName + '</td><td>'

+ value.LastName + '</td><td>'

+ value.Gender + '</td><td>'

+ value.Salary + '</td></tr>');

$('#tblData').append(row);

});

},

error: function (jqXHR) {

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

});

});

});

</script>

</body>

</html>

* We want to create a Modal that will inform the User to log in again because their token has expired. It will then redirect to the Login Page.
* Make the following changes to the code.

$('#btnLoadEmployees').click(function () {

$.ajax({

url: '/api/Employees',

method: 'GET',

headers: {

'Authorization' : 'Bearer ' + sessionStorage.getItem('accessToken')

},

success: function (data) {

$('#divData').removeClass('hidden');

$('#tblBody').empty();

$.each(data, function (index, value) {

var row = $('<tr><td>' + value.ID + '</td><td>'

+ value.FirstName + '</td><td>'

+ value.LastName + '</td><td>'

+ value.Gender + '</td><td>'

+ value.Salary + '</td></tr>');

$('#tblData').append(row);

});

},

error: function (jqXHR) {

if (jqXHR.status == '401') {

$('#errorModal').modal('show');

}

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

});

});

});

</script>

* Now we are going to redirect the User to Login page once the modal is closed.

**LECTURE 26 – ASPNET WEBAPI LOGOUT**

* We will look into implementing a logout functionality in the ASP.NET Web API.
* First increase the access token expiry time that we set to 10 seconds before.
* Go to **EmployeeService2/App\_Start/Starup.Auth.cs** and change the expiry time to **FromHours(1).**

// Configure the application for OAuth based flow

PublicClientId = "self";

OAuthOptions = new OAuthAuthorizationServerOptions

{

TokenEndpointPath = new PathString("/Token"),

Provider = new ApplicationOAuthProvider(PublicClientId),

AuthorizeEndpointPath = new PathString("/api/Account/ExternalLogin"),

AccessTokenExpireTimeSpan = TimeSpan.FromHours(1),

// In production mode set AllowInsecureHttp = false

AllowInsecureHttp = true

};

* We will now include a Logoff button to enable logging off.
* Go to **Data.html** and make the following amendments.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<link href="Content/bootstrap.min.css" rel="stylesheet" />

</head>

<body style="padding-top: 20px">

<div class="col-md-10 col-md-offset-1">

<div class="well">

<input type="button" id="btnLoadEmployees" class="btn btn-success" name="name" value="Load Employees" />

<input type="button" id="btnLogoff" class="btn btn-success pull-right" name="name" value="Log Off" />

</div>

* Make the following changes to the jQuery.

<script type="text/javascript">

$(document).ready(function () {

if (sessionStorage.getItem('accessToken') == null) {

window.location.href = 'Login.html';

}

$('#errorModal').on('hidden.bs.modal', function () {

window.location.href = 'Login.html';

});

$('#btnLogoff').click(function () {

sessionStorage.removeItem('accessToken');

window.location.href = 'Login.html';

});

$('#linkclose').click(function () {

$('#diverror').hide('fade');

});

$('#btnLoadEmployees').click(function () {

$.ajax({

url: '/api/Employees',

method: 'GET',

headers: {

'Authorization' : 'Bearer ' + sessionStorage.getItem('accessToken')

},

success: function (data) {

$('#divData').removeClass('hidden');

$('#tblBody').empty();

$.each(data, function (index, value) {

var row = $('<tr><td>' + value.ID + '</td><td>'

+ value.FirstName + '</td><td>'

+ value.LastName + '</td><td>'

+ value.Gender + '</td><td>'

+ value.Salary + '</td></tr>');

$('#tblData').append(row);

});

},

error: function (jqXHR) {

if (jqXHR.status == '401') {

$('#errorModal').modal('show');

}

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

});

});

});

</script>

* Now if you don’t want to lose the access token when the browser is closed you need to use **browser** **local storage** instead of browser session storage.
* The difference between browser local storage and browser session storage is that the access token is not lost when the browser is closed when using browser local storage while it is lost when using browser session storage.
* In Data.html change all instances of sessionStorage to localStorage.

<script type="text/javascript">

$(document).ready(function () {

if (localStorage.getItem('accessToken') == null) {

window.location.href = 'Login.html';

}

$('#errorModal').on('hidden.bs.modal', function () {

window.location.href = 'Login.html';

});

$('#btnLogoff').click(function () {

localStorage.removeItem('accessToken');

window.location.href = 'Login.html';

});

$('#linkclose').click(function () {

$('#diverror').hide('fade');

});

$('#btnLoadEmployees').click(function () {

$.ajax({

url: '/api/Employees',

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken')

},

success: function (data) {

$('#divData').removeClass('hidden');

$('#tblBody').empty();

$.each(data, function (index, value) {

var row = $('<tr><td>' + value.ID + '</td><td>'

+ value.FirstName + '</td><td>'

+ value.LastName + '</td><td>'

+ value.Gender + '</td><td>'

+ value.Salary + '</td></tr>');

$('#tblData').append(row);

});

},

error: function (jqXHR) {

if (jqXHR.status == '401') {

$('#errorModal').modal('show');

}

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

});

});

});

</script>

* Also change all the instances of sessionStorage to localStorage on **Login.html**.

<script type="text/javascript">

$(document).ready(function () {

$('#linkClose').click(function () {

$('#divError').hide('fade');

});

$('#btnLogin').click(function () {

$.ajax({

url: '/token',

method: 'POST',

contentType: 'application/json',

data: {

username: $('#txtUsername').val(),

password: $('#txtPassword').val(),

grant\_type: 'password'

},

success: function (response) {

localStorage.setItem('accessToken', response.access\_token);

window.location.href = "Data.html";

},

error: function (jqXHR) {

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

});

});

});

</script>

* From now on when you login to the application and then close the browser without explicitly logging out, your session will still exist. Just navigate to the Data.html page in the browser and you’ll be directed there without being prompted to log in first.

**LECTURE 27 – HOW TO GET AUTHENTICATED USER IDENTITY NAME IN ASPNET WEBAPI**

* What we want to do is get the logged in username and display it on the page.
* To see all the properties we get back when getting the access token make amendments to the code in **Login.html**.

<script type="text/javascript">

$(document).ready(function () {

$('#linkClose').click(function () {

$('#divError').hide('fade');

});

$('#btnLogin').click(function () {

$.ajax({

url: '/token',

method: 'POST',

contentType: 'application/json',

data: {

username: $('#txtUsername').val(),

password: $('#txtPassword').val(),

grant\_type: 'password'

},

success: function (response) {

//localStorage.setItem('accessToken', response.access\_token);

//window.location.href = "Data.html";

$('#divErrorText').text(JSON.stringify(response));

$('#divError').show('fade');

},

error: function (jqXHR) {

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

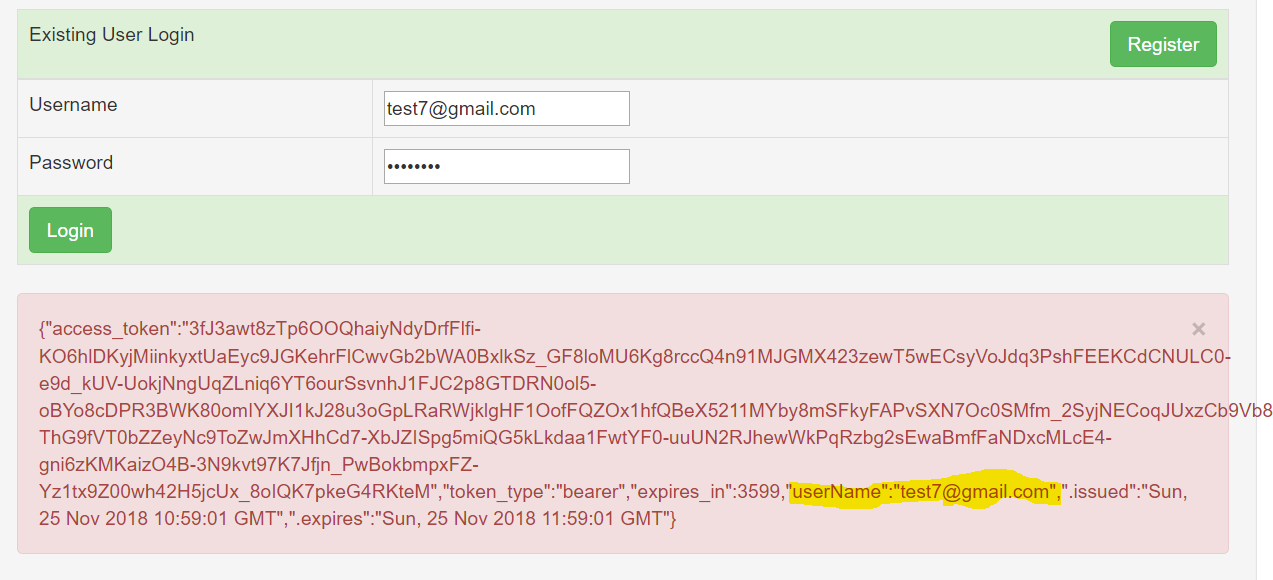
});

});

});

</script>

* When you build the API and try to log in you get the following response.
* You will notice that the access token also comes with the username.
* We can store this username in the localStorage for use later.



* Make the following changes to the jQuery in **Login.html**.

<script type="text/javascript">

$(document).ready(function () {

$('#linkClose').click(function () {

$('#divError').hide('fade');

});

$('#btnLogin').click(function () {

$.ajax({

url: '/token',

method: 'POST',

contentType: 'application/json',

data: {

username: $('#txtUsername').val(),

password: $('#txtPassword').val(),

grant\_type: 'password'

},

success: function (response) {

localStorage.setItem('accessToken', response.access\_token);

localStorage.setItem(userName, response. userName);

window.location.href = "Data.html";

},

error: function (jqXHR) {

$('#divErrorText').text(jqXHR.responseText);

$('#divError').show('fade');

}

});

});

});

</script>

* Now we want to display this username in the Data.html page.
* Go to Data.html and make the following changes in the markup.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<link href="Content/bootstrap.min.css" rel="stylesheet" />

</head>

<body style="padding-top: 20px">

<div class="col-md-10 col-md-offset-1">

<div class="well">

<input type="button" id="btnLoadEmployees" class="btn btn-success" name="name" value="Load Employees" />

<span id="spanUsername" class="text-muted"></span>

<input type="button" id="btnLogoff" class="btn btn-success pull-right" name="name" value="Log Off" />

</div>

* In Data.html make the following changes to the jQuery.

<script type="text/javascript">

$(document).ready(function () {

$('#spanUsername').text('Hello ' + localStorage.getItem(‘userName’));

if (localStorage.getItem('accessToken') == null) {

window.location.href = 'Login.html';

}

$('#errorModal').on('hidden.bs.modal', function () {

window.location.href = 'Login.html';

});

$('#btnLogoff').click(function () {

localStorage.removeItem('accessToken');

window.location.href = 'Login.html';

});