Contents

[Restful API 2](#_Toc55399337)

[Create ASP.Net Application 2](#_Toc55399338)

[Add NuGet Packages 2](#_Toc55399339)

[Add Startup class 2](#_Toc55399340)

[API with Static Data 3](#_Toc55399341)

[Add Controller 3](#_Toc55399342)

[Add Model 4](#_Toc55399343)

[Add Get/Post methods in Controller 4](#_Toc55399344)

[API with AX Data 7](#_Toc55399345)

[Add Controller 7](#_Toc55399346)

[Add Model 7](#_Toc55399347)

[Connect with AX using Business connector 8](#_Toc55399348)

[Add Get/Post methods in Controller 8](#_Toc55399349)

[Ref Link: 12](#_Toc55399350)

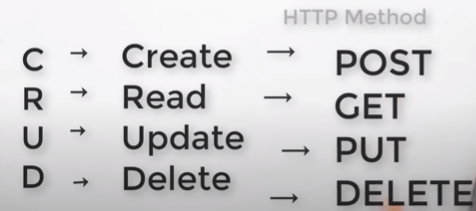
[Webhook: 12](#_Toc55399351)

# Restful API

Purpose: API used to integrate between two applications

Format of URL: {Localhost:IP/ Domain site}/api/{Controller}

Methods:



# Create ASP.Net Application

Create ASP.Net application in **Visual Studio 2015** and follow the steps as provided in the link above.

1. Go to **File > New > Project…**
2. Select the **Visual C# project category** and then select ASP.NET Web Application (.NET Framework)
3. Name your project **RestAPI** and click **OK**
4. Select the **Empty** project template and click OK (don’t check any boxes to add core references)

# Add NuGet Packages

Now you need to get a few **NuGet packages**. Use these commands in the Package Manager console to install them:

* Install-Package Microsoft.AspNet.WebApi
* Install-Package Microsoft.Owin.Host.SystemWeb
* Install-Package Microsoft.AspNet.WebApi.OwinSelfHost

# Add Startup class

Now right click on your project select **Add > Class**… and name it **Startup.cs**. Copy and paste this into the new file:

using System.Web.Http;

using Newtonsoft.Json.Serialization;

using Owin;

namespace RestAPI

{

public class Startup

{

public void Configuration(IAppBuilder app)

{

var config = new HttpConfiguration();

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.ContractResolver = new CamelCasePropertyNamesContractResolver();

config.Formatters.JsonFormatter.SerializerSettings.DateTimeZoneHandling = Newtonsoft.Json.DateTimeZoneHandling.Utc;

app.UseWebApi(config);

}

}

}

In the code above highlighted code in yellow removes the XmlFormatter (which was the default output formatter), and configures the **JsonFormatter** to camel-case property names and to use **UTC time for dates**.

# API with Static Data

## Add Controller

Next add a **Controllers folder** to your project. Then right click on the Controllers folder and select **Add > New Item….** On the left select **Visual C# > Web > Web API**. Then click on **Web API Controller Class (v2.1)**, name it **<ListItems>Controller.cs**, and click **Add**.

Now you should have a controller with methods to **Get**, **Post**, **Put**, and **Delete** list items. Let’s test it.

Press **F5** to launch your API. After the browser opens, add **/api/listitems** to the end of the URL and hit **Enter**

## Add Model

[Create a **Resource** and the ASP.NET **Web API Actions**](https://developer.okta.com/blog/2019/03/13/build-rest-api-with-aspnet-web-api#create-a-resource-and-the-aspnet-web-api-actions)

Now let’s make this API actually do something useful. In this section, you’ll create a list item resource and wire up all of the controller actions so you can create, read, update, and delete items.

Go ahead and create a **Models** folder in your project and add a **CustomListItem.cs** class. It should look like this:

The structure of the **CustomListItem.cs** is to be defined as per the data being posted.

Below is the CustomListItem.cs for your reference:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace RestAPI.Models

{

public class CustomListItem

{

public int Id { get; set; }

public string Text { get; set; }

}

}

## Add Get/Post methods in Controller

Back in your **ListItemsController** class, add a **private static property** to store your list items in memory. Add the private property inside the class declaration.

private static List<CustomListItem> \_listItems { get; set; } = new List<CustomListItem>();

You will also need to add a **using statement** to the top of the controller.

using RestAPI.Models;

To read items, update Get method as below:

public IEnumerable<CustomListItem> Get()

{

return \_listItems;

}

To create items, update Post method as specified below:

public HttpResponseMessage Post([FromBody]CustomListItem model)

{

if (string.IsNullOrEmpty(model?.Text))

{

return Request.CreateResponse(HttpStatusCode.BadRequest);

}

var maxId = 0;

if (\_listItems.Count > 0)

{

maxId = \_listItems.Max(x => x.Id);

}

model.Id = maxId + 1;

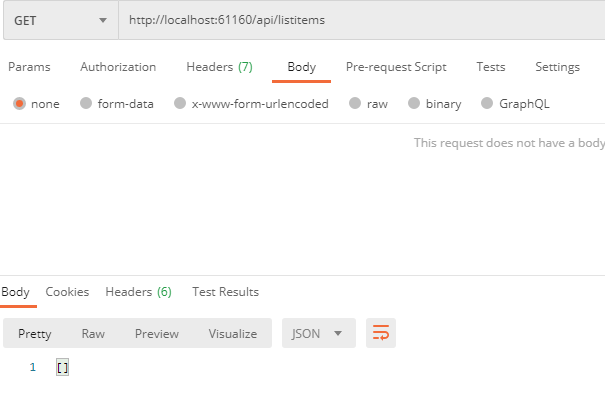
\_listItems.Add(model);

return Request.CreateResponse(HttpStatusCode.Created, model);

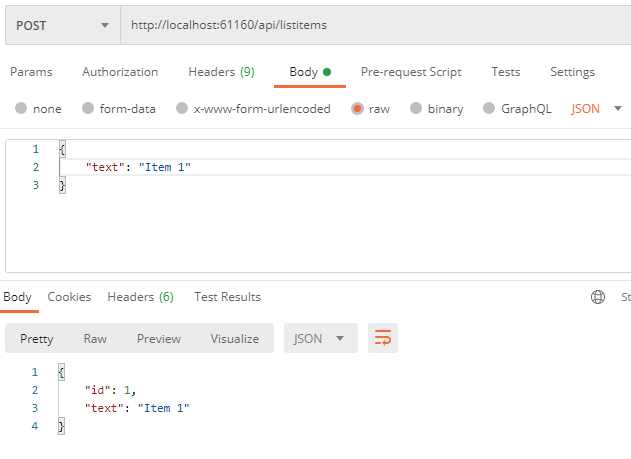
}

Build and run the application and test the post method in Postman as below:

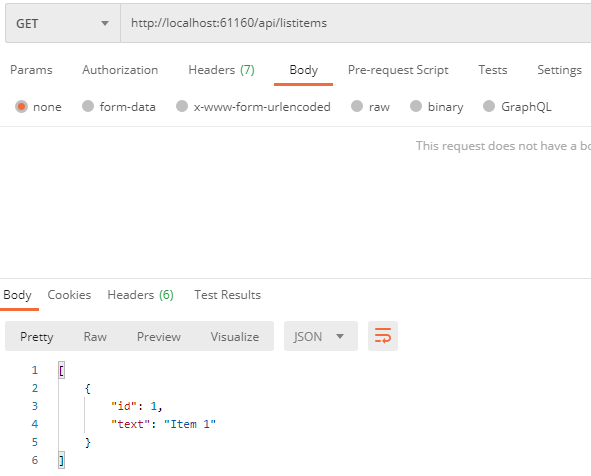
**Get Method**



**Post Method**



**Get Method after Post**



# API with AX Data

## Add Controller

Next add a **Controllers folder** to your project. Then right click on the Controllers folder and select **Add > New Item….** On the left select **Visual C# > Web > Web API**. Then click on **Web API Controller Class (v2.1)**, name it **<AxTrainings>Controller.cs**, and click **Add**.

Now you should have a controller with methods to **Get**, **Post**, **Put**, and **Delete** list items. Let’s test it.

Press **F5** to launch your API. After the browser opens, add **/api/AxTrainings** to the end of the URL and hit **Enter**

## Add Model

[Create a **Resource** and the ASP.NET **Web API Actions**](https://developer.okta.com/blog/2019/03/13/build-rest-api-with-aspnet-web-api#create-a-resource-and-the-aspnet-web-api-actions)

Add **AxTrainings** class in RestAPI.Models as shown below:

public class AXTrainings

{

public string Id { get; set; }

public string Name { get; set; }

public string TrainingDate { get; set; }

}

## Connect with AX using Business connector

To connect with the business connector, add reference of business connectors’ dll file from Bin folder of AX client as shown below:

C:\Program Files (x86)\<Microsoft Dynamics AX>\60\Client\Bin

Add using statement as below:

using Microsoft.Dynamics.BusinessConnectorNet;

## Add Get/Post methods in Controller

Back in your **AxTrainingsController** class, add a **private static property** to store your list items in memory. Add the private property inside the class declaration.

private static List<AXTrainings> \_AxTrainings { get; set; } = new List<AXTrainings>();

You will also need to add a **using statement** to the top of the controller.

using RestAPI.Models;

To read AxTrainings from AX, update Get method as below:

public IEnumerable<AXTrainings> Get()

{

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

AXTrainings objAXTrainings;

Axapta ax;

AxaptaRecord axRecord;

string tableName = "Dev\_Trainings";

string strId = "Id";

string strName = "Name";

string strTrainingDate = "TrainingDate";

object fieldId, fieldName, fieldTrainingDate;

try

{

ax = new Axapta();

ax.Logon("USMF", "en-us", "TAURUS", null);

\_AxTrainings.Clear();

using (axRecord = ax.CreateAxaptaRecord(tableName))

{

axRecord.ExecuteStmt("select \* from %1 ");

while (axRecord.Found)

{

fieldId = axRecord.get\_Field(strId);

fieldName = axRecord.get\_Field(strName);

fieldTrainingDate = axRecord.get\_Field(strTrainingDate);

objAXTrainings = new AXTrainings();

objAXTrainings.Id = fieldId.ToString();

objAXTrainings.Name = fieldName.ToString();

objAXTrainings.TrainingDate = fieldTrainingDate.ToString();

\_AxTrainings.Add(objAXTrainings);

axRecord.Next();

}

}

return \_AxTrainings;

}

catch (Exception e)

{

//return e.Message;

}

return \_AxTrainings;

}

To create AxTraining, update Post method as specified below:

public HttpResponseMessage Post([FromBody]AXTrainings objAXTrainings)

{

Axapta ax;

AxaptaRecord axRecord;

string tableName = "Dev\_Trainings";

if (string.IsNullOrEmpty(objAXTrainings?.Name))

{

return Request.CreateResponse(HttpStatusCode.BadRequest);

}

// insert record in ax

try

{

ax = new Axapta();

ax.Logon("USMF", "en-us", "TAURUS", null);

\_AxTrainings.Clear();

using (axRecord = ax.CreateAxaptaRecord(tableName))

{

axRecord.set\_Field("Id", objAXTrainings.Id);

axRecord.set\_Field("Name", objAXTrainings.Name);

axRecord.set\_Field("TrainingDate", objAXTrainings.TrainingDate);

// Commit the record to the database.

axRecord.Insert();

}

return Request.CreateResponse(HttpStatusCode.Created, objAXTrainings);

}

catch (Exception e)

{

objAXTrainings = new AXTrainings();

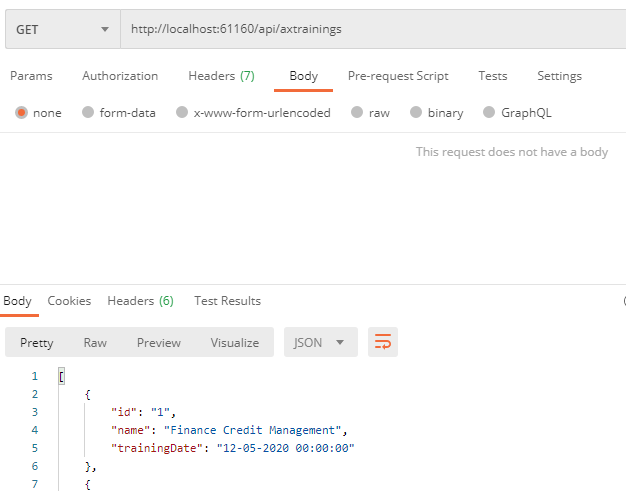
return Request.CreateResponse(HttpStatusCode.Created, objAXTrainings);

}

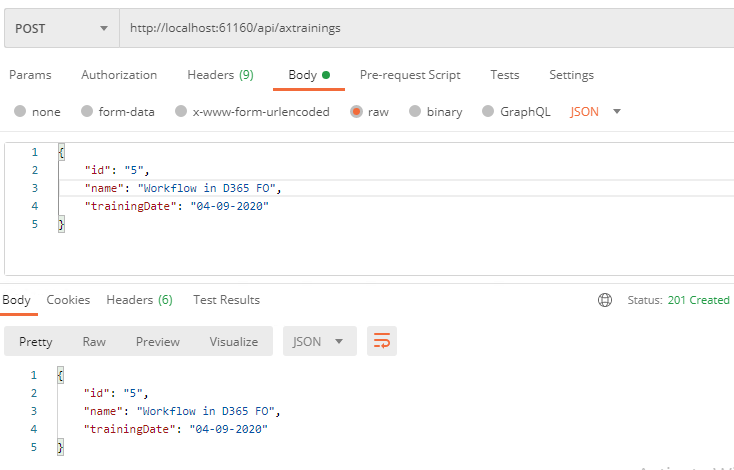
}

Build and run the application and test the post method in Postman as below:

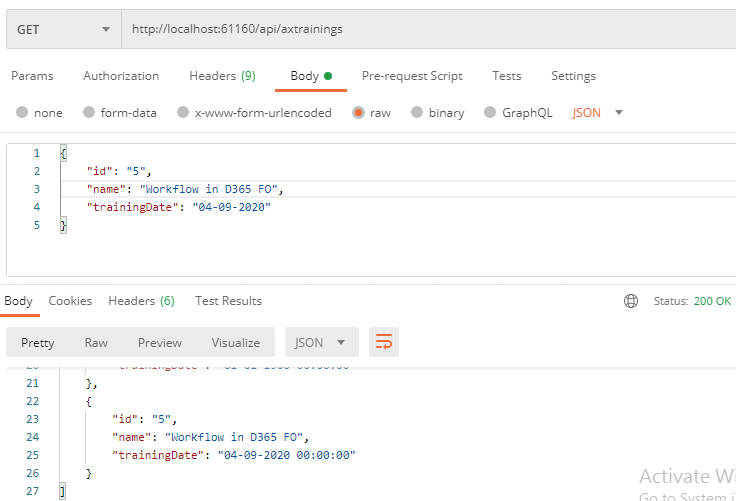
**Get Method**



**Post Method**



**Get Method after Post**



# Ref Link:

<https://developer.okta.com/blog/2019/03/13/build-rest-api-with-aspnet-web-api>

# Webhook:

Webhook delivers data from one application to other application immediately on occurring the event. Webhook used to get real time data between two applications.