# REST(JSON) based CRUD operations with ASP.NET Web API

Describe how to create restful CRUD (Create, Read, Update and Delete) operations on database using ASP.NET WEB API.

For this article we will use ASP.NET Web API empty template as default template comes with bunch of files which may not be useful for you. The ASP.NET Web API comes with Visual Studio 2013. If you do not have it you may download trial version here. If you have Visual Studio 2012 you can download ASP.NET Web API empty template here.

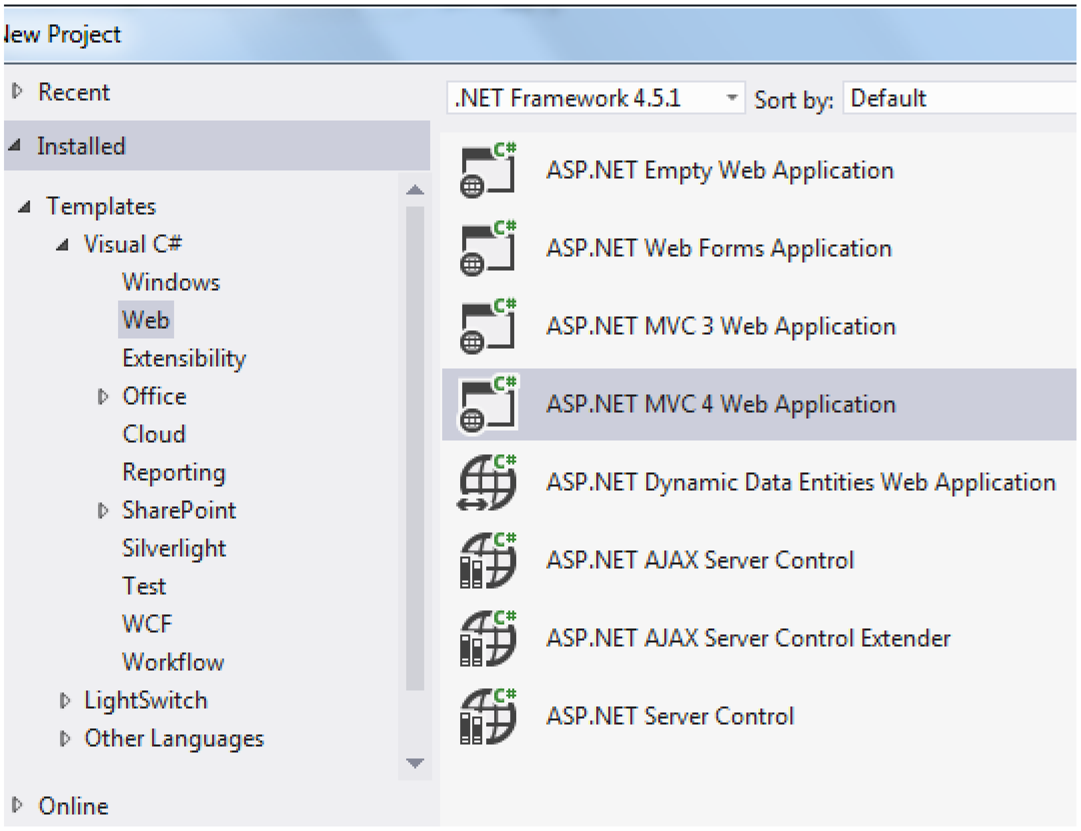
The operations in this article will perform CRUD on Northwind database. Clients can access those operations in RESTful manner that is resource based. You can see more details about Create RESTful services.

## Introduction to ASP.NET Web API

The ASP.NET Web API comes with ASP.NET MVC4 or later versions. It is a great framework to develop HTTP / HTTPS based services which can be used by broad range of clients like Smart Phones, Tablets, Browsers or even desktop applications. This framework is part of core ASP.NET platform and can be used by any applications which can understand HTTP communications. Develop your first ASP.NET Web API with MVC application

## Create ASP.NET Web API application

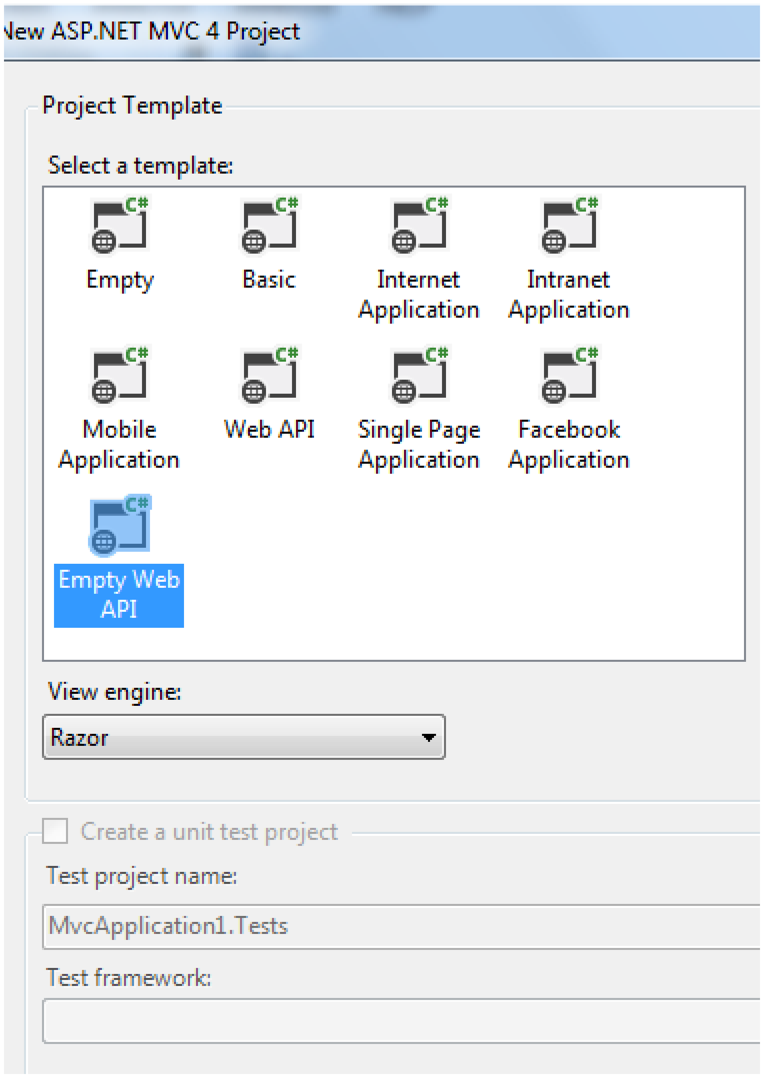
Open your Visual Studio - click on File -> New Project -> Select **Web**template -> choose **ASP.NET MVC 4 Web Application** -> name it as**CustomerApp** and click Ok.



.NET Framework 4.5.1 Sort by: Default

ASP.NET Empty Web Application
ASP.NET Web Forms Application
ASP.NET MVC 3 Web Application
ASP.NET MVC 4 Web Application
ASP.NET Dynamic Data Entities Web Application
ASP.NET AJAX Server Control
ASP.NET AJAX Server Control Extender
ASP.NET Server Control

From next template choose **Empty Web API** and ViewEngine as **Razor** -> click Ok.



## Models in Web API

Create new folder with name **Models** under root directory of**CustomerApp** application.

Add new class to Models folder by right click on Models -> Add -> Class -> give name **Customer** and click ok.

Add below properties for Customer model.

```
namespace CustomerApp.Models
{
    public class Customer
    {
        public string CustomerID { get; set; }
        public string CompanyName { get; set; }
        public string ContactName { get; set; }
        public string ContactTitle { get; set; }
        public string City { get; set; }
```

```
            public string Country { get; set; }
            public string Phone { get; set; }
        }
    }
```

## Repository Pattern in ASP.NET Web API

The Repository Pattern helps you to create abstract layer by isolating business objects from Data Access Layer. With Repository Pattern you can maximize the amount of code for automated unit testing and isolated data access from data store. This pattern allows you to centralize data access, code readability and maintainability and have flexible architecture.

In this step you will create repository class for Customer entity. You will create customer repository interface and customer repository class under**Models** folder or you can create separate folder to keep Repository interfaces and classes. When Controller or ApiController instantiate repositorry, then that controller can access any object which implements same repository interface.

Right click Models folder from Solution Explorer -> select **Add** -> select**Interface** -> name it as **ICustomerRepository** and click Ok.

Add below CRUD methods to ICustomerRepository which will be called in REST style

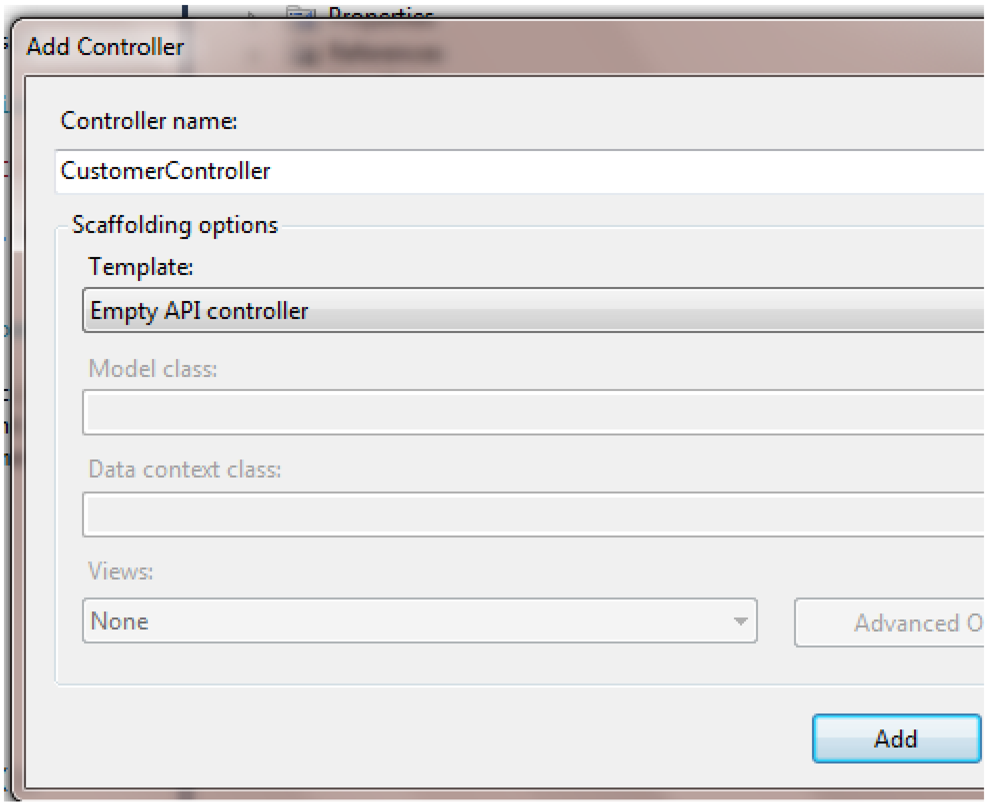
```
    namespace NorthwindCustomer.Models
    {
        public interface ICustomerRepository
        {
            IEnumerable<Customer> GetAll();
            Customer Get(string customerID);
            Customer Add(Customer item);
            void Remove(string customerID);
            bool Update(Customer item);
        }
    }
```

## Customer ApiController

Microsoft has included ApiController with release of ASP.NET 4 Beta version. The beauty of ApiController is, it can dynamically return JSON, XML or any other format depending on client **Accept** header of request message. Previous to this for returning different output required different implementation of methods. For more info on ASP.NET Web API conceptsGetting started with ASP.NET Web API.

Add Customer ApiController by right click on Controllers folder from Solution Explorer. Select Add -> Controller. Name it as CustomerController

From next window select Empty ApiController



Controller name: CustomerController

Scaffolding options

Template: Empty API controller

Views: None

Add below code to **CustomerController**.

```
using System.Net;
using System.Net.Http;
using System.Web.Http;
using CustomerApp.Models;

namespace CustomerApp.Controllers
{
    public class CustomerController : ApiController
    {
        static readonly ICustomerRepository repository = new CustomerRepository();

        public IEnumerable<Customer> GetAllCustomers()
        {
            return repository.GetAll();
        }

        public Customer GetCustomer(string customerID)
        {
            Customer customer = repository.Get(customerID);
            if (customer == null)
            {
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }
            return customer;
        }

        public IEnumerable<Customer> GetCustomersByCountry(string country)
        {
            return repository.GetAll().Where(
                c => string.Equals(c.Country, country,
                         StringComparison.OrdinalIgnoreCase));
        }

        public Customer PostProduct(Customer customer)
        {
            customer = repository.Add(customer);
            return customer;
        }

        public HttpResponseMessage PostCustomer(Customer customer)
        {
            customer = repository.Add(customer);
            var response = Request.CreateResponse
                    <Customer>(HttpStatusCode.Created, customer);
            string uri = Url.Link("DefaultApi",
                                new { customerID = customer.CustomerID });
            response.Headers.Location = new Uri(uri);
            return response;
        }

        public void PutProduct(string customerID, Customer customer)
        {
            customer.CustomerID = customerID;
            if (!repository.Update(customer))
            {
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }
        }

        public void DeleteProduct(string customerID)
        {
            Customer customer = repository.Get(customerID);
            if (customer == null)
            {
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }
            repository.Remove(customerID);
        }
    }
}
```

This API controller can be called by any client like ASP.NET Web Forms, ASP.Net MVC, Console Application, Windows application or any application that can communicate with Http.

## Configure routing table for ApiController

When http request receives by Web API framework, it uses routing table to determine which action to be invoke. Public methods of ApiController are called as action methods.

Web API framework first selects controller by**IHttpControllerSelector.SelectController** method and then selects action method by **IHttpActionSelector.SelectAction** depending on matching value in request uri and configured route table.

You can configure route table in **WebApiConfig.cs** file. So open this file from **App_Start** folder add below code.

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

            config.Routes.MapHttpRoute(
                name: "WithActionApi",
                routeTemplate: "api/{controller}/{action}/{customerID}"
            );

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

Make sure the below code is present in Global.asx.cs file.

```
    protected void Application_Start()
    {
        WebApiConfig.Register(GlobalConfiguration.Configuration);
    }
```

Now you can execute below RESTful Read operation using browser. The first url will return all Customers and second url will read single customer depending on customerID provided. You might have to change port number.

```
    http://localhost:38762/api/customer/getallcustomers


    http://localhost:38762/api/customer/getcustomer/alfki
```

## Return JSON response by default

If you execute GET request, ASP.NET Web API framework will return response in XML, JSON or any other format depending on accept header of request. Chorme and Firefox browser has XML as default value for accept header whereas IE has JSON as default value. So if you execute request through Chrome or Firefox you will get response in XML and if you use IE you will get JSON response.

If you want your Web API should return JSON response as default you will have to configure JSON Media Type Formatter.

Open **WebApiConfig.cs** file from App_start folder and add below code after route configuration.

```
  config.Formatters.JsonFormatter.
          SupportedMediaTypes.Add(new MediaTypeHeaderValue("text/html"));
```