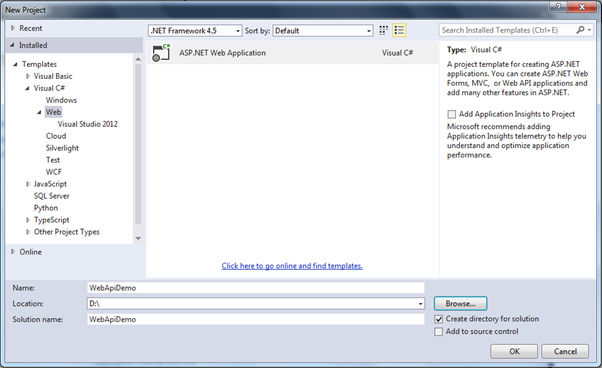
# Create Web API for CRUD operation

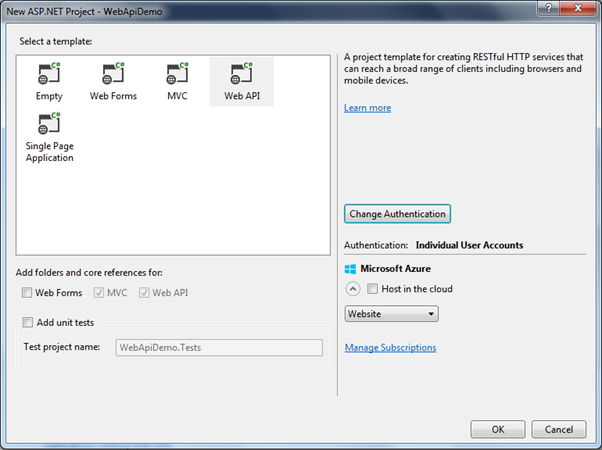
Here we will create a new Web API project and implement GET, POST, PUT and DELETE method for CRUD operation using Entity Framework.

First, create a new Web API project in Visual Studio 2013 for Web express edition.

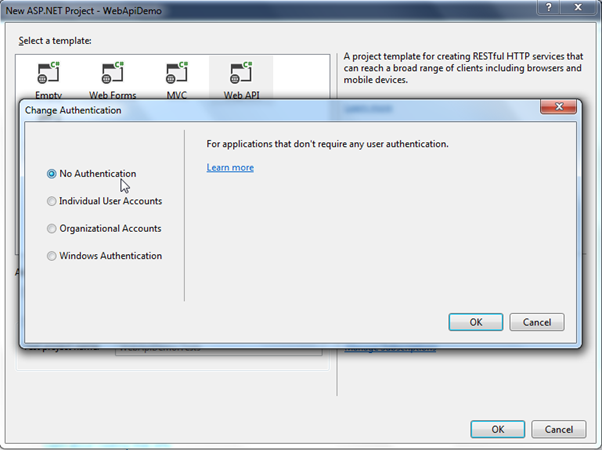
Open Visual Studio 2015 for Web and click on **File** menu -> **New Project..** This will open New Project popup as shown below.

[](https://www.tutorialsteacher.com/Content/images/webapi/create-webapi-crud1.png)Create Web API Project

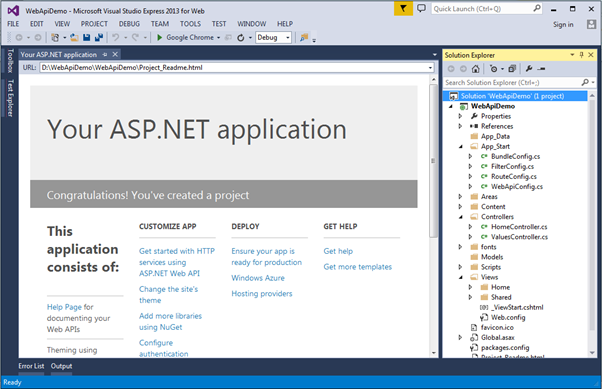
In the New Project popup, select **Web** template under **Visual C#**. Enter project name WebApiDemo and the location where you want to create the project. Click **OK** to continue. This will open another popup to select a project template. Select Web API project as shown below.

[](https://www.tutorialsteacher.com/Content/images/webapi/create-webapi-crud2.png)Select Web API Project Template

Here, we are not going to use any authentication in our demo project. So, click on **Change Authentication** button to open Authentication popup and select **No Authentication** radio button and then click **OK** as shown below.

[](https://www.tutorialsteacher.com/Content/images/webapi/create-webapi-crud3.png)Change Authentication

Now, click **OK** in New ASP.NET Project popup to create a project as shown below.

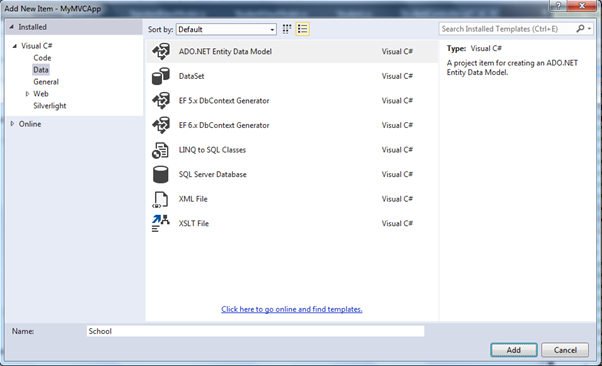
[](https://www.tutorialsteacher.com/Content/images/webapi/create-webapi-crud4.png)Web API Project

As you can see, a new WebApiDemo project is created with all necessary files. It has also added default ValuesController. Since, we will be adding our new Web API controller we can delete the default ValuesController.

Here, we are going to use Entity Framework DB-First approach to access an existing school database. So, let's add EF data model for the school database using DB First approach.

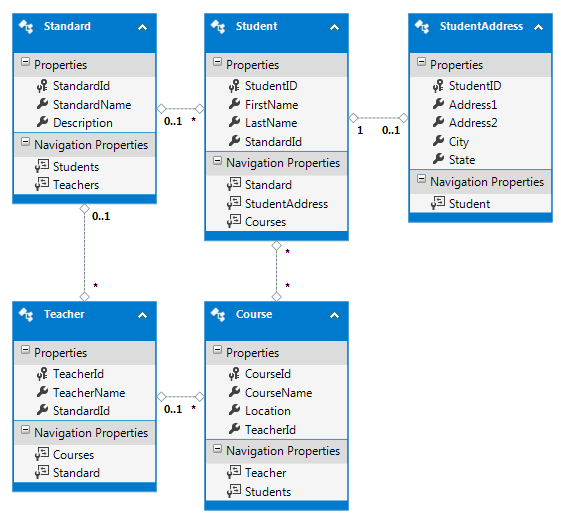
### **Add Entity Framework Data Model**

To add EF data model using DB-First approach, right click on your project -> click **New Item.** This will open Add New Item popup as shown below.

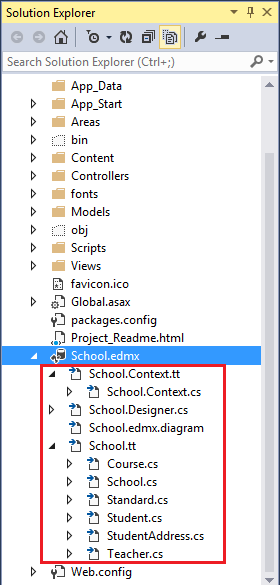
[](https://www.tutorialsteacher.com/Content/images/webapi/create-webapi-crud5.png)Create Entity Data Model

Select **Data** in the left pane and select **ADO.NET Entity Data Model** in the middle pane and enter the name of a data model and click **Add**. This will open Entity Data Model Wizard using which you can generate Entity Data Model for an existing School database. The scope of the topic is limited to Web API so we have not covered how to generate EDM. Learn about it [here](http://www.entityframeworktutorial.net/EntityFramework5/create-dbcontext-in-entity-framework5.aspx).

EntityFramework will generate following data model after completing all the steps of Entity Data Model Wizard.

[](https://www.tutorialsteacher.com/Content/images/webapi/ef-data-model.png)Generated Entities in the EDM Designer

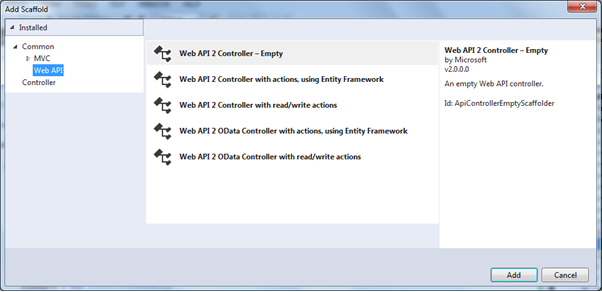
Entity Framework also generates entities and context classes as shown below.

[](https://www.tutorialsteacher.com/Content/images/webapi/webapi-project.png).edmx in the Project

Now, we are ready to implement CRUD operation using Entity Framework in our Web API project. Now, let's add a Web API controller in our project.

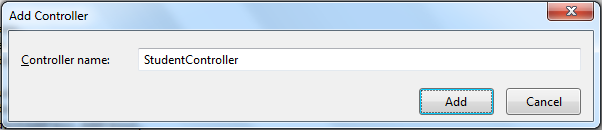
## **Add Web API Controller**

To add a Web API controller in your MVC project, right click on the **Controllers** folder or another folder where you want to add a Web API controller -> select **Add** -> select **Controller**. This will open **Add Scaffold** popup as shown below.

[](https://www.tutorialsteacher.com/Content/images/webapi/create-webapi-controller.png)Create Web API Controller

In the Add Scaffold popup, select **Web API** in the left pane and select **Web API 2 Controller - Empty** in the middle pane and click **Add**. (We select Empty template as we plan to add action methods and Entity Framework by ourselves.)

This will open **Add Controller** popup where you need to enter the name of your controller. Enter "StudentController" as a controller name and click **Add** as shown below.

[](https://www.tutorialsteacher.com/Content/images/webapi/webapi-controller-name.png)Create Web API Controller

This will add empty StudentController class derived from ApiController as shown below.

Web API Controller

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

namespace MyMVCApp.Controllers

{

public class StudentController : ApiController

{

}

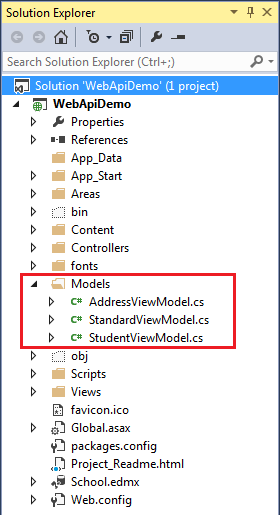
}

We will implement GET, POST, PUT and DELETE action methods in this controller in the subsequent sections.

### **Add Model**

We will be accessing underlying database using Entity Framework (EF). As you have seen above, EF creates its own entity classes. Ideally, we should not return EF entity objects from the Web API. It is recommended to return DTO (Data Transfer Object) from Web API. As we have created Web API project with MVC, we can also use MVC model classes which will be used in both MVC and Web API.

Here, we will return Student, Address and Standard from our Web API. So, create StudentViewModel, AddressViewModel and StandardViewModel in the Models folder as shown below.

[](https://www.tutorialsteacher.com/Content/images/webapi/create-webapi-crud6.png)Models

The followings are model classes.

Model Classes

public class StudentViewModel

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public AddressViewModel Address { get; set; }

public StandardViewModel Standard { get; set; }

}

public class StandardViewModel

{

public int StandardId { get; set; }

public string Name { get; set; }

public ICollection<StudentViewModel> Students { get; set; }

}

public class AddressViewModel

{

public int StudentId { get; set; }

public string Address1 { get; set; }

public string Address2 { get; set; }

public string City { get; set; }

public string State { get; set; }

}

Note: ViewModel classes or DTO classes are just for data transfer from Web API controller to clients. You may name it as per your choice.

# <https://www.tutorialsteacher.com/Content/images/webapi/implement-get3.png>

# Create Web API for CRUD operation Implement Get Method

This section is a continuation of the previous section where we created the necessary infrastructure for our Web API.

In this section we will implement Get action methods in our Web API controller class that will handle HTTP GET requests.

As per the Web API naming convention, action method that starts with a work "Get" will handle HTTP GET request. We can either name it only Get or with any suffix. Let's add our first Get action method and give it a name GetAllStudents because it will return all the students from the DB. Following an appropriate naming methodology increases readability and anybody can understand the purpose of a method easily.

The following GetAllStudents() action method in StudentController class (which we created in the previous section) returns all the students from the database using Entity Framework.

Example: Get Method in Web API Controller

public class StudentController : ApiController

{

public IHttpActionResult GetAllStudents ()

{

IList<StudentViewModel> students = null;

using (var ctx = new SchoolDBEntities())

{

students = ctx.Students.Include("StudentAddress")

.Select(s => new StudentViewModel()

{

Id = s.StudentID,

FirstName = s.FirstName,

LastName = s.LastName

}).ToList<StudentViewModel>();

}

if (students.Count == 0)

{

return NotFound();

}

return Ok(students);

}

}

As you can see in the above example, GetAllStudents() method returns all the students using EF. If no student exists in the DB then it will return 404 NotFound response otherwise it will return 200 OK response with students data. The NotFound() and Ok() methods defined in the ApiController returns 404 and 200 response respectively.

In the database, every student has zero or one address. Suppose, you want to implement another GET method to get all the Students with its address then you may create another Get method as shown below.

Example: Get Methods in Web API Controller

public class StudentController : ApiController

{

public IHttpActionResult GetAllStudents ()

{

IList<StudentViewModel> students = null;

using (var ctx = new SchoolDBEntities())

{

students = ctx.Students.Include("StudentAddress")

.Select(s => new StudentViewModel()

{

Id = s.StudentID,

FirstName = s.FirstName,

LastName = s.LastName

}).ToList<StudentViewModel>();

}

if (students.Count == 0)

{

return NotFound();

}

return Ok(students);

}

public IHttpActionResult GetAllStudentsWithAddress()

{

IList<StudentViewModel> students = null;

using (var ctx = new SchoolDBEntities())

{

students = ctx.Students.Include("StudentAddress").Select(s => new StudentViewModel()

{

Id = s.StudentID,

FirstName = s.FirstName,

LastName = s.LastName,

Address = s.StudentAddress == null ? null : new AddressViewModel()

{

StudentId = s.StudentAddress.StudentID,

Address1 = s.StudentAddress.Address1,

Address2 = s.StudentAddress.Address2,

City = s.StudentAddress.City,

State = s.StudentAddress.State

}

}).ToList<StudentViewModel>();

}

if (students.Count == 0)

{

return NotFound();

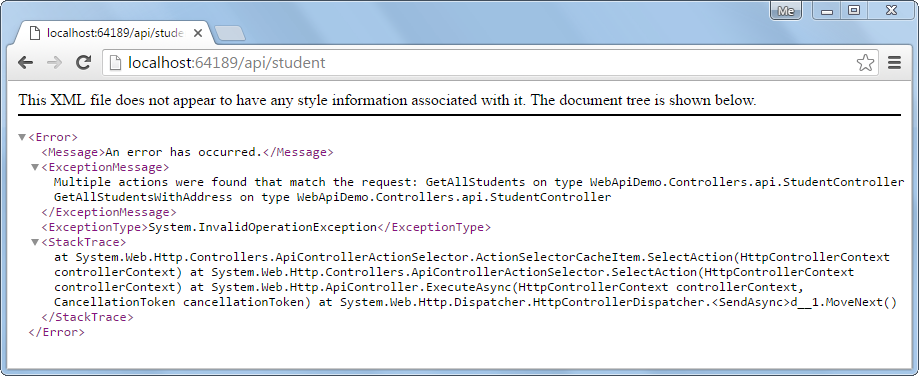
}

return Ok(students);

}

}

The above web API example will compile without an error but when you execute HTTP GET request then it will respond with the following multiple actions found error.

[](https://www.tutorialsteacher.com/Content/images/webapi/multiple-action-error.png)Web API Error

This is because you cannot have multiple action methods with same number of parameters with same type. Both action methods above do not include any parameters. So Web API does not understand which method to execute for the HTTP GET request http://localhost:64189/api/student.

The following example illustrates how to handle this kind of scenario.

Example: Get Method in Web API Controller

public class StudentController : ApiController

{

public StudentController()

{

}

public IHttpActionResult GetAllStudents(bool includeAddress = false)

{

IList<StudentViewModel> students = null;

using (var ctx = new SchoolDBEntities())

{

students = ctx.Students.Include("StudentAddress")

.Select(s => new StudentViewModel()

{

Id = s.StudentID,

FirstName = s.FirstName,

LastName = s.LastName,

Address = s.StudentAddress == null || includeAddress == false ? null : new AddressViewModel()

{

StudentId = s.StudentAddress.StudentID,

Address1 = s.StudentAddress.Address1,

Address2 = s.StudentAddress.Address2,

City = s.StudentAddress.City,

State = s.StudentAddress.State

}

}).ToList<StudentViewModel>();

}

if (students.Count == 0)

{

return NotFound();

}

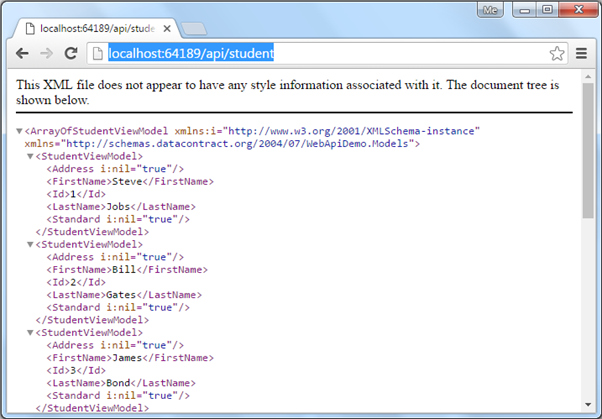
return Ok(students);

}

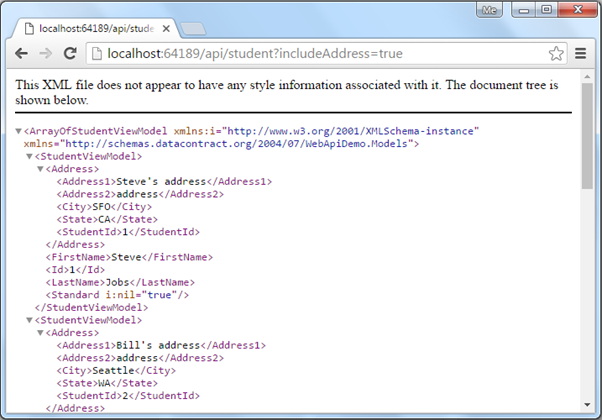
}

As you can see, GetAllStudents action method includes parameter includeAddress with default value false. If an HTTP request contains includeAddress parameter in the query string with value true then it will return all students with its address otherwise it will return students without address.

For example, http://localhost:64189/api/student (64189 is a port number which can be different in your local) will return all students without address as shown below.

[](https://www.tutorialsteacher.com/Content/images/webapi/implement-get1.png)Access Web API GET Method in the Browser

An HTTP request http://localhost:64189/api/student?includeAddress=true will return all students with address as shown below.

[](https://www.tutorialsteacher.com/Content/images/webapi/implement-get2.png)Access Web API GET Method in the Browser

## Implement Multiple GET methods

As mentioned, Web API controller can include multiple Get methods with different parameters and types.

Let's add following action methods in StudentController to demonstrate how Web API handles multiple HTTP GET requests.

| **Action method** | **Purpose** |
| --- | --- |
| GetStudentById(int id) | Returns student whose id matches with the specified id. |
| GetAllStudents(string name) | Returns list of students whose name matches with the specified name. |
| GetAllStudentsInSameStandard(int standardId) | Returns list of students who are in the specified standard. |

The following example implements the above action methods.

Example: Multiple Get Methods in Web API Controller

public class StudentController : ApiController

{

public StudentController()

{

}

public IHttpActionResult GetAllStudents(bool includeAddress = false)

{

IList<StudentViewModel> students = null;

using (var ctx = new SchoolDBEntities())

{

students = ctx.Students.Include("StudentAddress").Select(s => new StudentViewModel()

{

Id = s.StudentID,

FirstName = s.FirstName,

LastName = s.LastName,

Address = s.StudentAddress == null || includeAddress == false ? null : new AddressViewModel()

{

StudentId = s.StudentAddress.StudentID,

Address1 = s.StudentAddress.Address1,

Address2 = s.StudentAddress.Address2,

City = s.StudentAddress.City,

State = s.StudentAddress.State

}

}).ToList<StudentViewModel>();

}

if (students == null)

{

return NotFound();

}

return Ok(students);

}

public IHttpActionResult GetStudentById(int id)

{

StudentViewModel student = null;

using (var ctx = new SchoolDBEntities())

{

student = ctx.Students.Include("StudentAddress")

.Where(s => s.StudentID == id)

.Select(s => new StudentViewModel()

{

Id = s.StudentID,

FirstName = s.FirstName,

LastName = s.LastName

}).FirstOrDefault<StudentViewModel>();

}

if (student == null)

{

return NotFound();

}

return Ok(student);

}

public IHttpActionResult GetAllStudents(string name)

{

IList<StudentViewModel> students = null;

using (var ctx = new SchoolDBEntities())

{

students = ctx.Students.Include("StudentAddress")

.Where(s => s.FirstName.ToLower() == name.ToLower())

.Select(s => new StudentViewModel()

{

Id = s.StudentID,

FirstName = s.FirstName,

LastName = s.LastName,

Address = s.StudentAddress == null ? null : new AddressViewModel()

{

StudentId = s.StudentAddress.StudentID,

Address1 = s.StudentAddress.Address1,

Address2 = s.StudentAddress.Address2,

City = s.StudentAddress.City,

State = s.StudentAddress.State

}

}).ToList<StudentViewModel>();

}

if (students.Count == 0)

{

return NotFound();

}

return Ok(students);

}

public IHttpActionResult GetAllStudentsInSameStandard(int standardId)

{

IList<StudentViewModel> students = null;

using (var ctx = new SchoolDBEntities())

{

students = ctx.Students.Include("StudentAddress").Include("Standard").Where(s => s.StandardId == standardId)

.Select(s => new StudentViewModel()

{

Id = s.StudentID,

FirstName = s.FirstName,

LastName = s.LastName,

Address = s.StudentAddress == null ? null : new AddressViewModel()

{

StudentId = s.StudentAddress.StudentID,

Address1 = s.StudentAddress.Address1,

Address2 = s.StudentAddress.Address2,

City = s.StudentAddress.City,

State = s.StudentAddress.State

},

Standard = new StandardViewModel()

{

StandardId = s.Standard.StandardId,

Name = s.Standard.StandardName

}

}).ToList<StudentViewModel>();

}

if (students.Count == 0)

{

return NotFound();

}

return Ok(students);

}

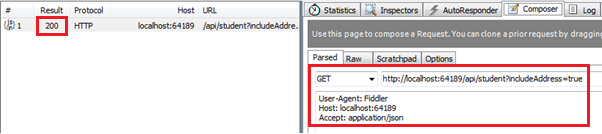
}

Now, the above Web API will handle following HTTP GET requests.

| **HTTP GET Request URL** | **Description** |
| --- | --- |
| http://localhost:64189/api/student | Returns all students without associated address. |
| http://localhost:64189/api/student?includeAddress=false | Returns all students without associated address. |
| http://localhost:64189/api/student?includeAddress=true | Returns all students with address. |
| http://localhost:64189/api/student?id=123 | Returns student with the specified id. |
| http://localhost:64189/api/student?name=steve | Returns all students whose name is steve. |
| http://localhost:64189/api/student?standardId=5 | Returns all students who are in 5th standard. |

Similarly you can implement Get methods to handle different HTTP GET requests in the Web API.

The following figure shows HTTP GET request in Fiddler.

[](https://www.tutorialsteacher.com/Content/images/webapi/implement-get3.png)Http GET request in Fiddler

The following figure shows HTTP GET response of above request in Fiddler.

# [https://www.tutorialsteacher.com/../Content/images/webapi/fiddler-response.png](https://www.tutorialsteacher.com/Content/images/webapi/fiddler-response.png)Http GET response in Fiddler

Create Web API for CRUD operation - Implement Post Method

This section is a continuation of the previous two sections where we created necessary infrastructure for the Web API and also implemented GET methods. Here, we will implement POST method in the Web API.

The HTTP POST request is used to create a new record in the data source in the RESTful architecture. So let's create an action method in our StudentController to insert new student record in the database using Entity Framework.

The action method that will handle HTTP POST request must start with a word Post. It can be named either Post or with any suffix e.g. POST(), Post(), PostNewStudent(), PostStudents() are valid names for an action method that handles HTTP POST request.

The following example demonstrates Post action method to handle HTTP POST request.

Example: Post Method in Web API Controller

public class StudentController : ApiController

{

public StudentController()

{

}

//Get action methods of the previous section

public IHttpActionResult PostNewStudent(StudentViewModel student)

{

if (!ModelState.IsValid)

return BadRequest("Invalid data.");

using (var ctx = new SchoolDBEntities())

{

ctx.Students.Add(new Student()

{

StudentID = student.Id,

FirstName = student.FirstName,

LastName = student.LastName

});

ctx.SaveChanges();

}

return Ok();

}

}

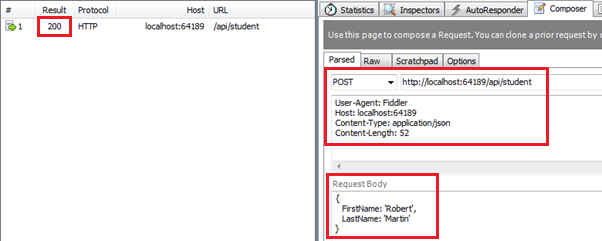
As you can see above, we named action method as PostNewStudent. You can give any name as per your requirement but it must start with the word "Post". The PostNewStudent() action method includes parameter of StudentViewModel type which includes all the information about new student.

In the Post method we first need to validate the model using ModelState.IsValid. This will make sure that student object includes all the necessary information. If it is not valid then you can return BadRequest response. If it is valid then add student using Entity Framework context and return 200 OK status response.

Note:

This is just a demo project. However, you can return newly created student object with Id in the response.

Now, you can send HTTP POST request using Fiddler as shown below and see the response.

[](https://www.tutorialsteacher.com/Content/images/webapi/fiddler-post-request.png)Execute HTTP POST request in Fiddler

As you can see in the above figure, HTTP POST request includes *StudentViewModel* object into JSON format in the request body. After successful execution the response status is 200 OK.

Create Web API for CRUD operation - Implement Put Method

This section is a continuation of the previous three sections where we created necessary infrastructure for the Web API and also implemented GET & POST methods. Here, we will implement PUT method in the Web API.

The HTTP PUT method is used to update an existing record in the data source in the RESTful architecture.

So let's create an action method in our StudentController to update an existing student record in the database using Entity Framework. The action method that will handle HTTP PUT request must start with a word Put. It can be named either Put or with any suffix e.g. PUT(), Put(), PutStudent(), PutStudents() are valid names for an action method that handles HTTP PUT request.

The following example demonstrates Put action method to handle HTTP PUT request.

Example: Put Method in Web API Controller

public class StudentController : ApiController

{

public StudentController()

{

}

public IHttpActionResult Put(StudentViewModel student)

{

if (!ModelState.IsValid)

return BadRequest("Not a valid model");

using (var ctx = new SchoolDBEntities())

{

var existingStudent = ctx.Students.Where(s => s.StudentID == student.Id)

.FirstOrDefault<Student>();

if (existingStudent != null)

{

existingStudent.FirstName = student.FirstName;

existingStudent.LastName = student.LastName;

ctx.SaveChanges();

}

else

{

return NotFound();

}

}

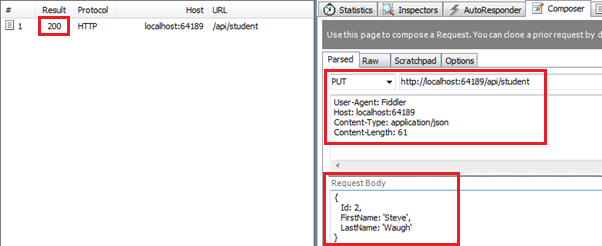
return Ok();

}

}

As you can see above, Put action method includes a parameter of StudentViewModel. It then creates new student entity using passed StudentViewModel object and then changes the state to be modified.

Now, you can send HTTP PUT request using Fiddler as shown below and see the response.

[](https://www.tutorialsteacher.com/Content/images/webapi/fiddler-put-request.png)Execute PUT request in Fiddler

As you can see in the above figure, HTTP PUT request includes *StudentViewModel* object into JSON format in the request body. After successfull execution the response status is 200 OK.

Create Web API for CRUD operation - Implement Delete Method

This section is a continuation of the previous four sections where we created necessary infrastructure for the Web API and also implemented GET, POST & PUT methods. Here, we will implement Delete action method in the Web API.

The HTTP DELETE request is used to delete an existing record in the data source in the RESTful architecture.

So let's create an action method in our StudentController to delete an existing student record in the database using Entity Framework. The action method that will handle HTTP DELETE request must start with the word "Delete". It can be named either Delete or with any suffix e.g. DELETE(), Delete(), DeleteStudent(), DeleteAllStudents() are valid names for an action method that handles HTTP DELETE request.

The following example demonstrates Delete action method to handle HTTP DELETE request.

Example: Delete Method in Web API Controller

public class StudentController : ApiController

{

public StudentController()

{

}

public IHttpActionResult Delete(int id)

{

if (id <= 0)

return BadRequest("Not a valid student id");

using (var ctx = new SchoolDBEntities())

{

var student = ctx.Students

.Where(s => s.StudentID == id)

.FirstOrDefault();

ctx.Entry(student).State = System.Data.Entity.EntityState.Deleted;

ctx.SaveChanges();

}

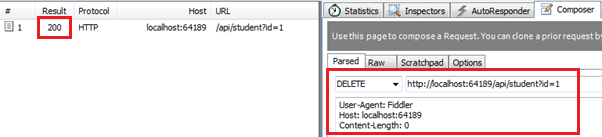
return Ok();

}

}

As you can see above, Delete action method includes an id parameter of int type because it just needs an id to delete a record. It fetches an existing student from the database that matches with the specified id and then marks its status as deleted. This will delete a student from the database.

Now, you can send HTTP DELETE request using Fiddler as shown below and view the response.

[](https://www.tutorialsteacher.com/Content/images/webapi/fiddler-delete-request.png)Execute HTTP DELETE request in Fiddler

As you can see in the above figure, HTTP DELETE request url http://localhost:64189/api/student?id=1 includes query string id. This id query string will be passed as an id parameter in Delete() method. After successful execution the response status is 200 OK.

Thus you can create Get, Post, Put and Delete methods to implement HTTP GET, POST, PUT and DELETE requests respectively.