**Веб-служба**, *веб-сервис* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) *web service*) — идентифицируемая [веб-адресом](https://ru.wikipedia.org/wiki/URI) программная система со стандартизированными [интерфейсами](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81).

**Се́рвис-ориенти́рованная архитекту́ра** (*SOA*, [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *service-oriented architecture*) — [модульный](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) подход к разработке [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), основанный на использовании [распределённых](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), [слабо связанных](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%BB%D0%B0%D0%B1%D0%BE%D0%B5_%D1%81%D0%B2%D1%8F%D0%B7%D1%8B%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5&action=edit&redlink=1) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) [*loose coupling*](https://en.wikipedia.org/wiki/loose_coupling)) заменяемых компонентов, оснащённых стандартизированными [интерфейсами](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) для взаимодействия по стандартизированным [протоколам](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB).

Программные комплексы, разработанные в соответствии с сервис-ориентированной архитектурой, обычно реализуются как набор [веб-служб](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0), взаимодействующих по протоколу [SOAP](https://ru.wikipedia.org/wiki/SOAP), но существуют и другие реализации (например, на базе [jini](https://ru.wikipedia.org/wiki/Jini), [CORBA](https://ru.wikipedia.org/wiki/CORBA), на основе [REST](https://ru.wikipedia.org/wiki/REST)).

Главное, что отличает SOA - это использование независимых сервисов с чётко определёнными интерфейсами, которые для выполнения своих задач могут быть вызваны неким стандартным способом, при условии, что сервисы заранее ничего не знают о приложении, которое их вызовет, а приложение не знает, каким образом сервисы выполняют свою задачу.

**ASP.NET Web API решает простую задачу: он поставляет услуги, которые поставляют данные из приложений ASP.NET для клиентов по HTTP или HTTPS протоколу**

#### Использование REST для построения Web-сервисов.

Web-сервис – это приложение работающее в World Wide Web и доступ к которому предоставляется по HTTP-протоколу, а обмен информации идет с помощью формата XML или JSON.

***Web API – это по сути прокси для работы с данными (базой данных).***

***Web API – это новый способ написания простых HTTP веб-служб.***

***ASP.NET Web API –Framework для создания HTTP сервисов. Позволяет создавать RESTсервисы***

Для каждой единицы информации (info) определяется 5 действий. А именно:

**GET /info/** *(Index)* – получает список всех объектов. Как правило, это упрощенный список, т.е. содержащий

только поля идентификатора и названия объекта, без остальных данных.

**GET /info/{id}** *(View)* – получает полную информацию о объекте.

**PUT /info/** или **POST /info/** *(Create)* – создает новый объект. Данные передаются в теле запроса без применения кодирования, даже urlencode.

**POST /info/{id}** или **PUT /info/{id}** *(Edit)* – изменяет данные с идентификатором {id}, возможно заменяет их. Данные так же передаются в теле запроса, но в отличие от PUT здесь есть некоторый нюанс. Дело в том, что POST-запрос подразумевает наличие urldecoded-post-data. Т.е. если не применять кодирования – это нарушение стандарта. Тут кто как хочет – некоторые не обращают внимания на стандарт, некоторые используют какую-нибудь post-переменную.

**DELETE /info/{id}** *(Delete)* – удаляет данные с идентификатором {id}.

REST очень проста в плане использования. По виду пришедшего запроса сразу можно определить, что он делает, не разбираясь в форматах (в отличие от SOAP, XML-RPC). Данные передаются без применения дополнительных слоев, поэтому REST считается менее ресурсоемким, поскольку не надо парсить запрос чтоб понять что он должен сделать и не надо переводить данные из одного формата в другой.

ASP.NET Web API использует понятие обычного MVC контроллера и базируется на нем для того, чтобы создать для разработчика простое и продуктивное событие. ***Web API оставляет SOAP в истории как средство, которое используют приложения для взаимодействия.***

На сегодняшний момент, из-за повсеместного использования HTTP, большинство рабочих сред и систем программирования поддерживают основные принципы HTTP веб-коммуникации. В связи с тем, что вопрос совместимости решается другими способами, SOAP может быть отодвинут в сторону возрастающими технологиями наследования, а разработчики могут быстро создавать простые HTTP веб-службы (web API) с помощью ASP.NET Web API фреймворка.

## Маршрутизация

Если мы создаем приложение asp.net mvc + asp.net wep api – то маршрутизация задается в файле WebApiConfig.cs, статический метод Register(HttpConfiguration config), которого вызывается в Global.asax в обработчике события Application\_Start.

## Наименования методов контроллера.

**Метод API контроллера будет выполнен в следующих случаях -**

* **Имя запускаемого метода совпадает с именемHTTPзапроса.**
* **Начало имени метода начинается с имени HTTP запроса.**
* **Метод декорирован атрибутом [AcceptVerbs] (HttpGet, HttpPost, HttpPut, HttpDelete)**

Web API предоставляет встроенную реализацию интерфейса - класс ApiControllerActionSelector, который по умолчанию собственно и управляет выбором метода контроллера. Выбор метода для обработки запроса представляет многоэтапный процесс:

1. Сначала идет обращение к механизму рефлексии для получения у контроллера всех возможных методов.
2. Затем у всех методов-кандидатов просматриваются атрибуты маршрутизации (при их наличии). Если будут найдены методы, в атрибутах которых определены маршруты, соответствующие запросу, то все остальные методы отбрасываются.
3. Далее просматриваются данные маршрута на наличие в них значения для сегмента action (если в маршруте определен такой сегмент). Если такое значение (которое передает имя метода) имеется, выбираются все методы, которые имеют данное имя. Остальные методы отбрасываются.
4. На следующем этапе проверяется, соответствуют ли методы соглашениям о наименовании Web API RESTful. Например, согласно этим условностям метод, обрабатывающий запрос GET, должно называться или начинаться с "Get", например, "GetBooks". Все методы, которые не соответствуют типу запроса (GET, POST, PUT) и не следуют условностям об наименовании, отбрасываются
5. Если ни один метод не использует соглашения о наименовании, то ApiControllerActionSelector просматривает наличие у методов атрибутов HttpGet, HttpPost, HttpDelete и HttpPut, которые соответствуют типу запроса.
6. На финальном этапе отбрасываются все методы, которые не соответствуют маршруту по числу и типу параметров.

Если не найдено ни одного метода или больше одного, то возникает ошибочная ситуация.

Методы контроллеров в Web API 2 могут возвращать различные типы:

* **void**: просто возвращает клиенту статусный код HTTP 204
* **HttpResponseMessage**: если метод возвращает объект HttpResponseMessage, то Web API преобразует возвращаемое значение в текст ответа HTTP
* **IHttpActionResult**: реализация этого интерфейса позволяет составить объекты, которые реализуют различные сценарии при обработке запроса
* любой другой тип: объекты любого другого типа сериализуются и добавляются в тело ответа, который также несет статусный код 200 (ОК)

## Статус коды HTTP

Классы **HttpRequestMessage** и **HttpResponceMessage** дают разработчику низко-уровневый контроль HTTPсообщения. С помощью этих классов, например, можно изменять заголовки, формировать нужный код ответа.

## Статус коды HTTP

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Шаблон | HTTP Метод | Описание | Статус | Содержимое |
| **/{key}** | **GET** | Получение источника по ключу | 200 –OK | Источник по ключу |
| 400 –Bad Request | Пустой, или текстошибки |
| 404 –Not Found | Пустой |
| **/** | **POST** | Добавление источника в коллекцию | 201 –Created | Добавленный источник |
| 204 –No Content | Пустой |
| 400 –Bad Request | Пустой, или текст ошибки |
| 409 –Conflict | Пустой, или текст ошибки |
| **/{key}** | **PUT** | Обновление источника в коллекции | 200 –OK | Измененный источник |
| 204 –No Content | Пустой |
| 400 –Bad Request | Пустой, или текст ошибки |
| 404 –Not Found | Пустой, или текст ошибки |
| 409 –Conflict | Пустой, или текст ошибки |
| **/{key}** | **DELETE** | Удаление источника из коллекции | 200 –OK | Удаленный ресурс |
| 204 –No Content | Пустой |
| 400 –Bad Request | Пустой, или текст ошибки |
| 404 –Not Found | Пустой, или текст ошибки |
| 409 –Conflict | Пустой, или текст ошибки |

Построим сервис для работы с классом

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public string Category { get; set; }

public decimal Price { get; set; }

}

public interface IProductRepository

{

IEnumerable<Product> GetAll();

Product Get(int id);

Product Add(Product item);

void Remove(int id);

bool Update(Product item);

}

В реализации интерфейса

public class ProductRepository : IProductRepository

{

private List<Product> products = new List<Product>();

private int \_nextId = 1;

public ProductRepository()

{

Add(new Product { Name = "Tomato soup", Category = "Groceries", Price = 1.39M });

Add(new Product { Name = "Yo-yo", Category = "Toys", Price = 3.75M });

Add(new Product { Name = "Hammer", Category = "Hardware", Price = 16.99M });

Add(new Product { Name = "Network Cards", Category = "Electronics", Price = 6.59M });

Add(new Product { Name = "Spotting Scopes", Category = "Optics", Price = 25.99M });

Add(new Product { Name = "Biometric Monitors", Category = "Health Care", Price = 100.0M });

Add(new Product { Name = "Perfume", Category = "Cosmetics", Price = 10.99M });

Add(new Product { Name = "Hair Coloring", Category = "Personal Care", Price = 16.99M });

}

public Product Add(Product item)

{

item.Id = \_nextId++;

products.Add(item);

return item;

}

public IEnumerable<Product> GetAll()

{

return products;

}

public Product Get(int id)

{

return products.Find(p => p.Id == id);

}

public void Remove(int id)

{

products.RemoveAll(p => p.Id == id);

}

public bool Update(Product item)

{

int index = products.FindIndex(p => p.Id == item.Id);

if (index == -1)

{

return false;

}

products.RemoveAt(index);

products.Add(item);

return true;

}

}

#region Такая реализация возвращает правильный ответ: 404 - Not Found

public HttpResponseMessage GetProducts(int id)

{

Product product = repository.Get(id);

if (product == null)

{

return Request.CreateResponse(HttpStatusCode.NotFound, "Item not found");

}

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

}

#endregion

HttpResponseMessage является основным способом для возврата результата из метода в контроллере Web API. Например, метод PostProduct, который создает и добавляет новый объект в БД

public HttpResponseMessage PostProduct(Product item)

{

item = repository.Add(item);

var response = Request.CreateResponse<Product>(HttpStatusCode.Created, item);

string uri = Url.Link("DefaultApi", new { id = item.Id });

response.Headers.Location = new Uri(uri);

return response;

}

**Работу сервиса можно протестировать с помощью Fiddler.**

Для добавления выбираем метод Post

Content-type: application/json

{

"Id":"10",

"Name":"Test",

"Category":"Hardware",

"Price": "1.10"

}

**Для того что бы HTTP методы могли работать при запуске в IIS Express необходимо поменять файл конфигурации, который находится по пути**

**C:\\Users\<UserName>\Documents\IISExpress\config\applicationhost.config**

**<add name="ExtensionlessUrl-Integrated-4.0" path="\*." verb="GET,HEAD,POST,DEBUG" type="System.Web.Handlers.TransferRequestHandler" preCondition="integratedMode,runtimeVersionv4.0" responseBufferLimit="0" />**

**В атрибут verb элемента указанного выше, необъодимо добавить необходимые глаголы PUT, DELETE.**

**Если на компьютере установлен WebDAV Publishing его нужно удалить через "программы и компоненты".**

**http://stackoverflow.com/questions/3946283/enabling-html-put-method-on-iis-7-5**

Потребителем нашего сервиса будет консольное приложение **AppTestWebApi** (Тестирование работы сервиса)

## Сопоставление классов WebAPI and MVC

[**http://metanit.com/sharp/aspnet\_webapi/1.4.php**](http://metanit.com/sharp/aspnet_webapi/1.4.php)

**web api 2 CRUD Operations with AngularJS???**

Домашнее задание.

1. Реализовать Web API сервис который работает с классом Product. Протестировать в консоли.
2. Реализовать веб интерфейс.