**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.

WCF – это один из основных компонентов Microsoft .NET Framework 3.x. Впервые WCF была включена в состав .NET 3.0

#### Что такое REST

**(Передача состояния представления - сильно завязан на протокол *HTTP*)**

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

В общем случае ***REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек***. ***Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL***. Каждый URI в свою очередь имеет строго заданный формат.

Отсутствие дополнительных внутренних прослоек означает передачу данных в том же виде, что и сами данные. Т.е. мы не заворачиваем данные в XML, как это делает SOAP и XML-RPC, не используем AMF, как это делает Flash и т.д. Просто отдаем сами данные.

## Основные принципы REST:

**Client-Server**–разделение ответственности

**Stateless**–общение между клиентом и сервером не подразумевает хранения состояния

**Cacheable**–клиенты могут кэшировать ответы. Ответ должен явно или не явно указывать клиенту может ли тот его кэшировать.

**Layered System** –клиент не может точно знать подключен ли он на прямую к серверу или пользуется промежуточным прокси сервером.

**Uniform Interface** –унифицированный интерфейс между клиентом и сервером основанный на URI и использования HTTP метода для определения операции, которую необходимо выполнить на сервере.

Как происходит управление информацией сервиса – это целиком и полностью основывается на протоколе передачи данных. Для asp.net - наиболее распространенный протокол конечно же HTTP. Так вот, для HTTP действие над данными задается с помощью методов:

* GET (получить);
* PUT (добавить, заменить);
* POST (добавить, изменить, удалить);
* DELETE (удалить).

Таким образом, действия CRUD (Create-Read-Updtae-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

Вот как это будет выглядеть на примере:

* GET /book/ — получить список всех книг
* GET /book/3/ — получить книгу номер 3
* PUT /book/ — добавить книгу (данные в теле запроса)
* POST /book/3 – изменить книгу (данные в теле запроса)
* DELETE /book/3 – удалить книгу

**ВАЖНОЕ ДОПОЛНЕНИЕ:** Существуют так называемые [REST-Patterns](http://wiki.developer.mindtouch.com/REST/REST_for_the_Rest_of_Us), которые различаются связыванием **HTTP-методов с тем, что они делают.** В частности, разные паттерны по-разному рассматривают POST и PUT. Однако, PUT предназначен для создания, реплейса или апдейта, для POST это не определено *(The POST operation is*

*very generic and no specific meaning can be attached to it)*.

Вообще, **POST** может использоваться одновременно для всех действий изменения:

* POST /book/ – добавить книгу (данные в теле запроса)
* POST /book/3 – изменить книгу (данные в теле запроса)
* POST /book/3 – удалить книгу (тело запроса пустое)

Это позволяет иногда обходить неприятные моменты, связанные с неприятием PUT и DELETE.

## Работа с Google Map API

## Инициализация карты

<script src="~/Scripts/jquery-2.2.1.min.js"></script>

<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"> </script>

<div style="width:1024px;height: 500px" id="map"></div>

<script type="text/javascript">

$(function () {

var latlng = new google.maps.LatLng(50.4492345, 30.5064439);

//var latlng = new google.maps.LatLng(p.lat, p.lng);

var centerlatlng = new google.maps.LatLng(50.447667, 30.508558);

//50.455264, 30.596319

var myOptions = {

zoom: 16,

center: centerlatlng,

mapTypeId: google.maps.MapTypeId.ROADMAP

};

var map = new google.maps.Map(document.getElementById("map"), myOptions);

var marker = new google.maps.Marker({

position: latlng,

map: map,

title: 'Hello World!'

});

</script>

## Автоматический маршрут

Нужно подключить функции маршрутов. Вставьте этот код в функцию инициализации карты:

var directionsDisplay = new google.maps.DirectionsRenderer();

var directionsService = new google.maps.DirectionsService();

directionsDisplay.setMap(map);

directionsDisplay.setOptions({ suppressMarkers: true, suppressInfoWindows: true });

Теперь можно использовать эти функции. Как именно и с какими координатами их запускать - на ваше усмотрение.

var request = {

origin: latlng,

destination: endPoint,

travelMode: google.maps.TravelMode.WALKING,

unitSystem: google.maps.UnitSystem.METRIC,

waypoints: [

{

location: latlng,

stopover: false

}, {

location: endPoint,

stopover: true

}

],

optimizeWaypoints: true,

provideRouteAlternatives: true,

avoidHighways: true,

avoidTolls: true

};

directionsService.route(request, function (result, status) {

if (status == google.maps.DirectionsStatus.OK) {

directionsDisplay.setDirections(result);

var routes = result.routes;

}

});

**origin** - (Обязательный параметр) координаты или адрес точки начала маршрута  
**destination** - (Обязательный параметр) координаты или адрес точки конца маршрута  
**travelMode** - (Обязательный параметр) способ перемещения (DRIVING - на автомобиле, BICYCLING - на велосипеде, TRANSIT - на общественном транспорте, WALKING - пешком)  
**unitSystem** - система измерения (METRIC - метрическая, IMPERIAL - британская)  
**waypoints** - массив с точками, через которые обязательно должен проходить маршрут  
**waypoints -> location** - координаты или адрес точки  
**waypoints -> stopover** - разделять ли маршрут на части в этой точке (true или false)  
**optimizeWaypoints** - включает оптимизацию маршрута с помощью точек в waypoints (true или false)  
**provideRouteAlternatives** – включает поиск нескольких альтернативных маршрутов (true или false)  
**avoidHighways** – пытаться ли избегать автомагистралей (true или false)  
**avoidTolls** – пытаться ли избегать платных дорого (true или false)  
**routes** - массив с вариантами маршрутов  
**leg** - массив с отрезками маршрута  
**lenght** - дистанция первого отрезка (или всего маршрута, если отрезок всего один)  
**duration** - продолжительность первого отрезка (или всего маршрута, если отрезок всего один)

## Построение маршрута по адресу

function () {

var address = $('#address').val();

$.getJSON('http://maps.googleapis.com/maps/api/geocode/json?address='

+ address + '&sensor=false', null, function (data) {

p = data.results[0].geometry.location;

var latlng = new google.maps.LatLng(p.lat, p.lng);

var directionsDisplay = new google.maps.DirectionsRenderer();

var directionsService = new google.maps.DirectionsService();

directionsDisplay.setMap(map);

directionsDisplay.setOptions({ suppressMarkers: true, suppressInfoWindows: true });

var travelModeId = $('#TravelMode').val();

var travelMode = GetTravelMode(travelModeId);

var endPoint = new google.maps.LatLng(50.4492345, 30.5064439); var request = {

origin: latlng,

destination: endPoint,

travelMode: travelMode, //google.maps.TravelMode.WALKING,

unitSystem: google.maps.UnitSystem.METRIC,

waypoints: [

{

location: latlng,

stopover: false

}, {

location: endPoint,

stopover: true

}

],

optimizeWaypoints: true,

provideRouteAlternatives: true,

avoidHighways: true,

avoidTolls: true

};

directionsService.route(request, function (result, status) {

if (status == google.maps.DirectionsStatus.OK) {

directionsDisplay.setDirections(result);

var routes = result.routes;

}

});

});

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

[**https://habrahabr.ru/company/dataart/blog/277419/**](https://habrahabr.ru/company/dataart/blog/277419/)

Закономерно, все же появился действительно простой подход — REST. Аббревиатура REST расшифровывается как representational state transfer — «передача состояния представления» или, ***лучше сказать, представление данных в удобном для клиента формате.*** Термин “REST” был введен Роем Филдингом в 2000 г. ***Основная идея REST в том, что каждое обращение к сервису переводит клиентское приложение в новое состояние.*** По сути, REST — не протокол и не стандарт, а подход, архитектурный стиль проектирования API.

<https://habrahabr.ru/post/252237/>

интсрументы для работы с API

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.

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.

***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 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.

Задание.

Создать карту Google Map. В центре – Ваш адрес.

На странице должны быть отображены – DropDownList с перечнем типа TravelMode, адрес и кнопка, по клику на которую происходит построение маршрута.