**MVC**

Web API vs WCF

|  |  |
| --- | --- |
| WCF | Web API |
| Enables building services that support multiple transport protocols (HTTP, TCP, UDP, and custom transports) and allows switching between them. | HTTP only. First-class programming model for HTTP. More suitable for access from various browsers, mobile devices etc enabling wide reach. |
| Supports building services with WS-\* standards like Reliable Messaging, Transactions, Message Security. | Uses basic protocol and formats such as HTTP, WebSockets, SSL, JSON, and XML. There is no support for higher level protocols such as Reliable Messaging or Transactions. |
| Supports Request-Reply, One Way, and Duplex message exchange patterns. | HTTP is request/response but additional patterns can be supported through [SignalR](https://github.com/SignalR/SignalR) and WebSockets integration. |
| Doesn’t provide support for MVC features like controller, routing, filter etc | Provides support for MVC features. |
| Ships with the .NET Framework. | Ships with .NET Framework but is open-source and is also available out-of-band as independent download. |

**REST**

Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services.

1. **Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
2. **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
3. **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
4. **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
5. **Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.
6. **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

**SOAP VS REST**

|  |  |
| --- | --- |
| SOAP | REST |
| SOAP stands for Simple Object Access Protocol | REST stands for Representational State Transfer |
| SOAP is a protocol. | REST is a architectural pattern. |
| SOAP uses service interfaces to expose its functionality to client applications. In SOAP, the WSDL file provides the client with the necessary information which can be used to understand what services the web service can offer. | REST use Uniform Service locators to access to the components on the hardware device. |
| SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot. | REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. |
| SOAP can only work with XML format. | REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON. |