API

Application Programming Interface, is a set of rules, protocols, and tools that allows different software applications to communicate with each other. APIs define the methods and data formats that developers can use to interact with a software component, service, or system.

Types of APIs:

Web APIs: These are APIs that are exposed over the web, typically using HTTP/HTTPS protocols. Web APIs are commonly used for communication between client-side and server-side applications. They can be further categorized into:

RESTful APIs: REST (Representational State Transfer) is an architectural style for designing networked applications. RESTful APIs use HTTP requests to perform CRUD (Create, Read, Update, Delete) operations on resources.

Soap vs Rest:

SOAP is a protocol specification for exchanging structured information in the implementation of web services in computer networks. It typically uses XML for message formatting and HTTP or SMTP for message transmission.

Soap messages are typically xml based.

It can maintain state between requests using session management mechanisms.

SOAP is more rigid, formal, and suited for enterprise-level applications with complex requirements,

REST: REST is an architectural style that uses standard HTTP methods (such as GET, POST, PUT, DELETE) for communication and typically uses JSON or XML for message formatting. REST is not tied to any specific protocol but commonly uses HTTP/HTTPS.

Rest messaging formats are both json and xml- based.

It can maintain state between requests using session management mechanisms.

REST is simpler, more lightweight, and suitable for simpler client-server interactions, especially in web and mobile applications.

Rest Principles:

StateLess:

The REST API is stateless meaning that every HTTP request happens in complete isolation. When the client makes the HTTP request, it will contain all the necessary information to make the server understand the requests sent from the client.

Client-Server

Both the client and the server are completely autonomous in every way. Their communication will only happen using the REST API. This improves the portability of the codebase across multiple platforms and also helps with making the server scalable.

Uniform Interface

REST API provides four interfaces to achieve uniformity.

Resource identification

Resource manipulation using representations

Self-descriptive messages

Hypermedia as the engine of application state

Cacheable

REST API is often made cacheable to improve the performance of the application. It reduces the server load and the client uses the cached response if the user is requesting the same resources. It also dynamically decreases the load time of your application.

Layered System

RESTful systems are organized into layers, with each layer responsible for a specific aspect of the system. Layers can include client-server communication, caching, load balancing, and security. This separation of concerns enhances scalability, flexibility, and maintainability.

Code on Demand (Optional):

REST allows for the transfer of executable code from the server to the client, enabling clients to extend their functionality dynamically. This principle is optional in REST and is not commonly used in practice, especially in public APIs.

Content-Type:

Definition: The Content-Type header specifies the media type of the request body sent to the server. It tells the server what type of data is being sent so that it can process it appropriately.

Example: Content-Type: application/json indicates that the request body is in JSON format.

Accept:

Definition: The Accept header specifies the media types that the client can understand or prefers to receive in the response. It tells the server what type of content the client expects.

Example: Accept: application/json indicates that the client prefers to receive the response in JSON format.

General Headers:

These headers apply to both requests and responses and provide general information about the message. Examples include:

Cache-Control: Specifies caching directives to be applied by caching mechanisms along the request-response chain.

Connection: Controls whether the network connection stays open after the current transaction finishes.

Date: Represents the date and time at which the message was sent.

Pragma: Used for backwards compatibility with HTTP/1.0 caches where Cache-Control is not yet present.

Request Headers:

These headers provide additional information about the client or the requested resource. Examples include:

Accept: Specifies the media types that are acceptable for the response.

Authorization: Contains credentials for authenticating the client with the server.

Host: Specifies the domain name of the server (useful for virtual hosting).

User-Agent: Provides information about the user agent (browser or client) making the request.

Response Headers:

These headers provide additional information about the server's response. Examples include:

Content-Type: Specifies the media type of the response body.

Expires: Specifies the date and time after which the response is considered stale.

Server: Provides information about the server software handling the request.

WWW-Authenticate: Indicates the authentication method that should be used to access a resource.

Entity Headers:

These headers are used to provide information about the message body, such as its length or encoding. Examples include:

Content-Length: Specifies the length of the message body in bytes.

Content-Encoding: Specifies the encoding algorithms applied to the message body.

Content-Language: Specifies the natural language(s) of the intended audience for the message body.

Content-Disposition: Provides a hint to the browser on how to display the response.

HTTP methods are the most commonly used::

GET: Used to retrieve information or data from the server. It is a safe and idempotent method, meaning it should not have any side effects on the server and can be repeated multiple times without changing the server's state.

POST: Data is sent to the server for processing. It is frequently used to create new server resources. POST, unlike GET, is not idempotent, as multiple requests may result in the generation of many resources.

PUT: Used to update or replace a resource on the server. It is idempotent in the sense that several identical PUT requests should produce the same results as a single request..

PATCH: Similar to PUT, PATCH is used to update a resource. However, while PUT typically replaces the entire resource, PATCH applies partial modifications. It is also idempotent.

DELETE: Used to request that a resource be removed from the server. Like GET, DELETE is also idempotent.

Hateos:

Hateoas:

HATEOAS stands for Hypermedia as the Engine of Application State

When using HATEOAS architecture, a client will be able to access the API for a network application through a simple, static, RESTful URL call. Now, any further actions, that the client may wish to take, will be enabled by the data, returned by the server, in the original call. This will enable the client to move from one application state to the next just by interacting with the details contained in the responses by the server.

The “data”, within the response, that enables this change of state is simple hypermedia links. This is how HATEOAS manages the change in application states through hypermedia.

**The basic differences between WCF and ASMX services:-**

1) ASMX services only support HTTP protocols where in WCF services supports other protocols like TCP, Namedpipes, MSMQ etc.

2) ASMX services have less security than WCF services.

3) ASMX services can be hosted in IIS only where in WCF services can be hosted in multiple ways e.g. Self Hosting, IIS Hosting , Hosting in Windows service, WAS Hosting.

4) ASMX uses XMLSerializer where in WCF uses DataContractSerializer which is faster than XMLSerializer. 5) The purpose of ASMX services is to send and receive data using SOAP over HTTP protocol. However, WCF services can send and receive data using any format over different protocols like HTTP, HTTPS, TCP, MSMQ etc.

Attributes:

attributes provide a powerful way to add metadata to types and members of your code. They can be used to provide additional information to the compiler, runtime, or other tools about how to process or treat your code

Compiler Instructions: Attributes can provide instructions to the compiler about how to process your code. For example, the [Obsolete] attribute marks a type or member as obsolete, causing the compiler to emit a warning or error when it is used.

Runtime Behavior: Attributes can affect the behavior of your code at runtime. For example, the [Serializable] attribute indicates that a class can be serialized, allowing its instances to be converted to a byte stream for storage or transmission.

Code Generation: Attributes can be used by code generation tools or frameworks to customize the code that is generated. For example, ASP.NET uses attributes like [WebMethod] to mark methods that should be exposed as web service endpoints.

Reflection: Attributes can be queried at runtime using reflection to provide information about types and members. This can be useful for implementing custom frameworks or tools that need to analyze your code.

Content negotiation

Content negotiation allows for more flexible and efficient communication between clients and servers, as it enables clients to receive responses in formats they can process and servers to provide representations that best suit the clients' needs. It also enables the use of different representations for the same resource, such as providing different languages or compressions based on client preferences.

Swagger:

Swagger is an open-source framework for designing, building, documenting, and consuming RESTful web services. It provides a set of tools and specifications that enable developers to describe the structure of their APIs in a standard and machine-readable format, making it easier to understand and interact with the API.

Web API

. Web API stands for Web Application Programming Interface. It is a framework for building HTTP services that can be consumed by a broad range of clients, including browsers, mobile devices, and desktop applications. Web APIs allow you to expose the functionality of your application as a set of HTTP endpoints that clients can interact with.

Key features of Web APIs include:

HTTP-based: Web APIs are built on top of the HTTP protocol, using standard HTTP methods like GET, POST, PUT, DELETE to perform operations on resources.

RESTful: Many Web APIs are designed according to the principles of REST (Representational State Transfer), which emphasizes a stateless client-server architecture, uniform interfaces, and the manipulation of resources through representations.

Data Formats: Web APIs typically use standard data formats like JSON (JavaScript Object Notation) or XML (eXtensible Markup Language) for representing data exchanged between the client and server.

Authentication and Authorization: Web APIs often support authentication and authorization mechanisms to control access to resources and protect sensitive data.

Scalability: Web APIs are designed to be scalable, allowing them to handle a large number of concurrent requests from clients.

Action filters:

Filters means changing the flow of execution

They are attributes that can be applied on a controller action method or whole controller.

This allow us to add pre and post processing logic to an action method.they allow us to modify the way in which an action is executed.

having the piece of code which want to be shared across multiple action methods of same or different controllers.

Same piece of code which is used for action methods written in action filters

Action filters:

Authorize

Which is ensuring the certain members only the login members only validate for excecution of that action

Child action only

Handler

Output cache

[output cache duration=10)]

Before the Action method only the html page rendor to browser and page processing is cancelled for every one second

validateInput “”” under default situations asp.net doesn’t take as input any content like tags so to allow that we write validateinput[false]

validateantiforeign token

user is not able to submit the form from one site other than the one from where the form was copied or download from server.

Bcz of attribute the cookie is checking by having this attribute.

TextBox:

The TextBox helper is used when you want to manually specify the name and value of the input element.

It is generally used when you are not strongly typing your view to a model or when you need more control over the input element's name and value.

The TextBoxFor helper is used when you want to bind the input element to a model property.

It is used in conjunction with a model expression (Expression<Func<TModel, TValue>>) to automatically generate the input element's name and value based on the model property.

WCF:

In WCF, you define a service contract interface (IService1) that contains operation contracts ([OperationContract]) defining the methods that clients can call. The service class (Service1) implements this interface and provides the actual implementation for the operations.

Data contracts ([DataContract] and [DataMember]) define the types that are exchanged between the client and the service. IService.cs (Service Contract Interface):

This is the interface where you define the operations (methods) that your service will expose.

It is decorated with the [ServiceContract] attribute to indicate that it defines a service contract.

Each method in the interface is decorated with the [OperationContract] attribute to indicate that it is an operation that can be called remotely.

The main use of WCF is to facilitate communication between applications and services in a distributed environment. It allows you to create services that expose endpoints for various communication protocols (e.g., HTTP, TCP, MSMQ) and message formats (e.g., XML, JSON).

clientid

793712155718-d35nd6q5mhksd09m4vhp07931lqqc185.apps.googleusercontent.com

Client secret

GOCSPX-wQWkhSt26tZiDHVtTkR60s-m\_fKH