

Unit - 4 Web Services

Why web service?







- Web Services make the communication between incompatible applications possible.
- The functionality that one application is trying to consume from the other is exposed as a Web Service.

Ex: Real time scenarios

1. A lot of companies like Amazon leverage different payment gateway options to pay for the purchases.
 - a. It is not guaranteed that all the vendor-specific applications and the payment gateway applications are compatible with Amazon in terms of technology and platform where they get executed.
 - b. Even then, the communication between these applications and Amazon happens successfully due to use of web services.

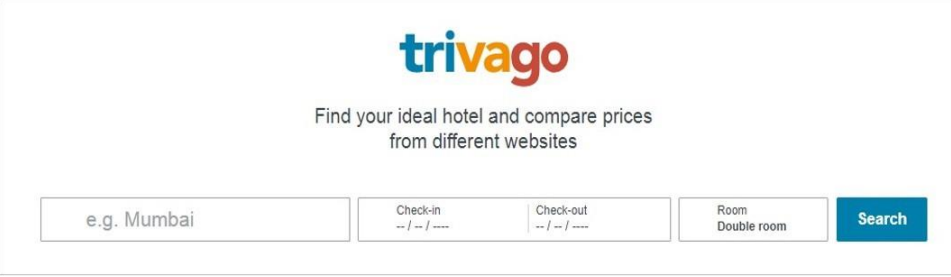
Edit Payment Information

Payment Type



2. Travelling is made easier and comfortable through different forms of transportation.
 - a. In order to make our stay comfortable and pleasant, we need to search for a good hotel that fits in our budget. We all know that different online hotel booking applications are there to make this search possible in a single click.
 - b. Trivago, a familiar online hotel booking site/application will give us the details of the hotels that are available in a particular locality based on our preference. As well, it will allow us to book rooms in a hotel for our stay.

- c. It is not possible for trivago, an outsider, to get access to the hotel databases because of security reasons. The hotels will also not push their room details to Trivago periodically, simply because when the updates happen periodically, there are situations where the end-customer gets misguided as Trivago and the partner hotels' databases are not in sync.
- d. The correct mechanism behind this is, Trivago gives calls to the associated hotels' application services when the end customer requests. But Trivago may not be compatible with all the associated applications. The communication still happens due to web services.



The image shows the Trivago search interface. At the top is the Trivago logo in blue and orange. Below it is the text "Find your ideal hotel and compare prices from different websites". The search area contains four input fields: a location field with the example "e.g. Mumbai", a "Check-in" field with a date range "-- / -- / --", a "Check-out" field with a date range "-- / -- / --", and a "Room" field with the example "Double room". To the right of these fields is a blue "Search" button.

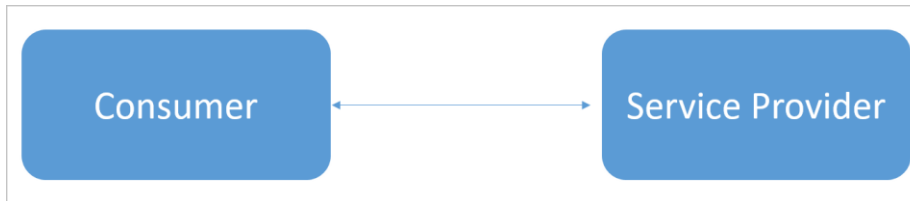
3. Quora gives us many sign-in options like Google and Facebook apart from its standard authentication.
 - a. If we have signed in to our Facebook already, it is not required for us to sign in to Quora individually. Rather, we can make use of an option called, "Continue with Facebook".
 - b. Here, it is implicitly understood that Quora avails the login service of Facebook. And, we see a lot of applications work this way in our day-to-day life like LinkedIn or InfySpringboard.
 - c. But it doesn't mean that Google and Facebook are compatible with Quora, the communication happens with the use of Web Services.



The image shows the Quora login interface. At the top is the Quora logo in red. Below it is the tagline "A place to share knowledge and better understand the world". The login area has two columns. The left column contains two buttons: "Continue with Google" (red) and "Continue with Facebook" (blue). Below these is a link "Continue With Email" with a note "By signing up you indicate that you have read and agree to Quora's Terms of Service and Privacy Policy." The right column is titled "Login" and contains a text input field for the username, a text input field for the password, and a "Forgot Password?" link. At the bottom of the right column is a blue "Login" button. Below the login area is a line of text: "New: Dutch, Danish, Finnish, Norwegian, Swedish, Marathi, Bengali, and Tamil". At the very bottom is a footer with links: "About", "Languages", "Careers", "Businesses", "Privacy", "Terms", "Contact", and "© Quora Inc. 2019".

SOA - Service Oriented Architecture

- SOA is a software model designed for achieving communication among distributed application components, irrespective of the differences in terms of technology, platform, etc.,
- The application component that requests for a service is the consumer/client and the one that renders the service is the producer/server.



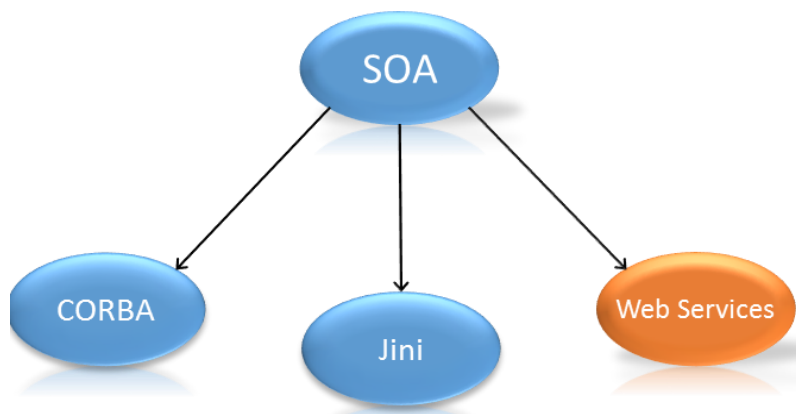
- SOA is an architecture that can help to build an enterprise application by availing the services/functionalities from different third-party entities without reinventing the existing functionalities.

SOA principles:

Principle	Explanation
Loose coupling	SOA aims at structuring procedures or software components as services. And, these services are designed in such a way that they are loosely coupled with the applications. So, these loosely-coupled services will come into picture only when needed.
Publishing the services	There should be a mechanism for publishing the services that include the functionality and input/output specifications of those services. In short, services are published in such a way that the developers can easily incorporate them into their applications.
Management	Finally, software components should be managed so that there will be a centralized control on the usage of the published services.

These principles are the driving factors behind SOA, ensuring a high level of abstraction, reliability, scalability, reusability, and maintainability.

Implementing SOA:



CORBA: Common Object Request Broker Architecture is a standard that achieves interoperability among the distributed objects. Using CORBA, it is much possible to make the distributed application components communicate with each other irrespective of the location (the place where the components are available), language and platform.

Jini: Jini or Apache River is a way of designing distributed applications with the help of services that interact and cooperate.

Web Services: Web services are a way of making applications interact with each other on the web. Web services leverage the existing protocols and technologies but use certain standards. Web Service is the most popular solution as it eliminates the issues related to interoperability with pre-defined standards and processes in place.

COBRA and Jini/Apache River : Tight coupling as objects are used
Web services : Loose coupling

What are Web services?

W3C defines Web Services as follows: "A web service is a software system designed to support interoperable machine-to-machine interaction over a network".

- Web services are client-server applications that communicate over HyperText Transfer Protocol (HTTP)
- Web services are a standard means of achieving interoperability between software applications running on diverse platforms and frameworks.
- Web Services are highly extensible. Services can be loosely coupled to solve complex operations.
- Web Service is a way of realizing Service Oriented Architecture (SOA). SOA is a style of software design where services are provided by the application components through a communication protocol. for example, retrieving a banking card statement online

Difference between WebApplication and WebService:

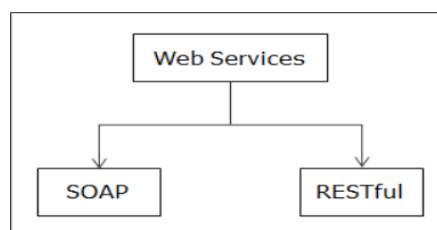
Web Application	Web Services
A web application is meant for humans	Web Services are used to exchange data between two incompatible applications
The Web application is a complete application with GUI(Graphical User Interface)	Web Services do not necessarily have a user interface since it is used as a component in an end to end application
A typical web application, for example, a Java web application can be accessed by Java Clients (Web/Desktop) alone. And, a web application is generally accessed through browsers	Web Services can be accessed by applications of different languages and platforms
Reusing the functionalities of a web application is difficult	Reusing the functionalities that are exposed as web services is highly possible

Benefits of Web Services:

- Integrates with other systems easily irrespective of the difference in technology or platform - Interoperability
- Creates reusable components - Reusability
- Saves cost, effort and time - Ease of

Types of Web Services

There are two types of web services:



- RESTful Web Services
- SOAP Web Services

RESTful Web Services

REST stands for **REpresentational State Transfer**. It is developed by **Roy Thomas Fielding** who also developed HTTP. The main goal of RESTful web services is to make web services **more effective**. RESTful web services try to define services using the different concepts that are already present in HTTP. REST is an **architectural approach**, not a protocol.

It does not define the standard message exchange format. We can build REST services with both XML and JSON. JSON is more popular format with REST. The **key abstraction** is a resource in REST. A resource can be anything. It can be accessed through a **Uniform Resource Identifier (URI)**. For example:

The resource has representations like XML, HTML, and JSON. The current state is captured by representational resource. When we request a resource, we provide the representation of the resource. The important methods of HTTP are:

- **GET:** It reads a resource.
- **PUT:** It updates an existing resource.
- **POST:** It creates a new resource.
- **DELETE:** It deletes the resource.

For example, if we want to perform the following actions in the social media application, we get the corresponding results.

POST /users: It creates a user.

GET /users/{id}: It retrieve the detail of one user.

GET /users: It retrieve the detail of all users.

DELETE /users: It delete all users.

DELETE /users/{id}: It delete a user.

GET /users/{id}/posts/post_id: It retrieve the detail of a specific post.

POST / users/{id}/ posts: It creates a post for a user.

GET /users/{id}/post: Retrieve all posts for a user

HTTP also defines the following standard status code:

- **404:** RESOURCE NOT FOUND
- **200:** SUCCESS
- **201:** CREATED
- **401:** UNAUTHORIZED
- **500:** SERVER ERROR

RESTful Service Constraints

- There must be a service producer and service consumer.
- The service is stateless.
- The service result must be cacheable.
- The interface is uniform and exposing resources.
- The service should assume a layered architecture.

Advantages of RESTful web services

- RESTful web services are **platform-independent**.
- It can be written in any programming language and can be executed on any platform.
- It provides different data format like **JSON, text, HTML, and XML**.
- It is fast in comparison to SOAP because there is no strict specification like SOAP.
- These are **reusable**.
- These are language neutral.

SOAP Web Services

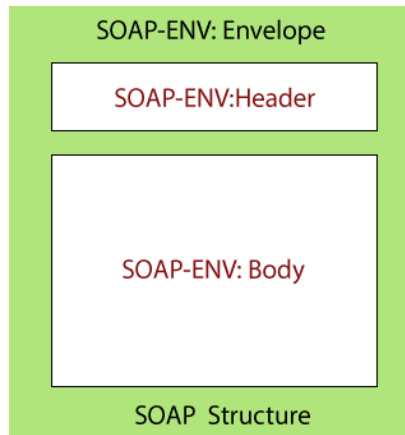
REST defines an architectural approach whereas SOAP poses a restriction on the format of the XML. XML transfer data between the service provider and service consumer. Remember that SOAP and REST are not **comparable**.

SOAP: SOAP acronym for **Simple Object Access Protocol**. It defines the standard XML format. It also defines the way of building web services. We use Web Service Definition Language (WSDL) to define the format of **request XML** and the **response XML**.

For example, we have requested to access the **Todo** application from the **Facebook** application. The Facebook application sends an XML request to the Todo application. Todo application processes the request and generates the XML response and sends back to the Facebook application.



If we are using SOAP web services, we have to use the **structure** of SOAP.



In the above figure, the **SOAP-Envelope** contains a **SOAP-Header** and **SOAP-Body**. It contains meta-information needed to identify the request, for example, authentication, authorization, signature, etc. SOAP-Header is optional. The **SOAP-Body** contains the real XML content of request or response. In case of an error, the response server responds back with SOAP-Fault.

Let's understand the SOAP XML request and response structure.

XML Request

1. `<Envelop xmlns=?http://schemas.xmlsoap.org/soap/envelop/?>`
2. `<Body>`
3. `<getCourseDetailRequest xmlns=?http://udemy.com/course?>`
4. `<id>course1</id>`
5. `<getCourseDetailRequest>`
6. `</Body>`
7. `</Envelop>`

XML Response

1. `<SOAP-ENV:Envelope xmlns:SOAP-ENV=?http://schemas.xmlsoap.org/soap/envelope/?>`
2. `<SOAP-ENV:Header />` `<!--empty header-->`
3. `<SOAP-ENV:Body>` `<!--body begin-->`
4. `<ns2:getCourseDetailsResponse xmlns:ns2=?http://in28mi>` `<!-- content of the response-->`
5. `<ns2:course>`


```

6.          <ns2:id>Course1</ns2:id>
7.          <ns2:name>Spring</ns2:name>
8.          <ns2:description>10 Steps</ns1:description>
9.          </ns2:course>
10.         </ns2:getCourseDetailResponse>
11.         </SOAP-ENV:Body>          <!?body end-->
12. </SOAP-ENV:Envelope>

```

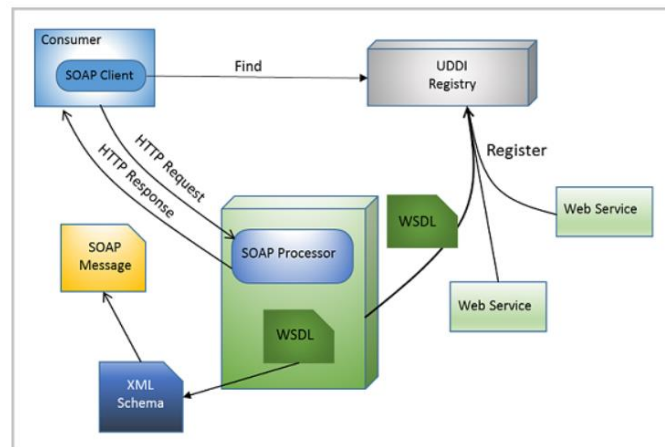
Points to remember

- SOAP defines the format of **request** and **response**.
- SOAP does not pose any restriction on transport. We can either use **HTTP** or **MQ** for communication.
- In SOAP, service definition typically done using **Web Service Definition Language (WSDL)**. WSDL defines **Endpoint**, **All Operations**, **Request Structure**, and **Response Structure**.

SOAP Web Services

- SOAP stands for Simple Object Access Protocol.
- It is a XML-based protocol for accessing web services over HTTP.
- SOAP is a W3C (the World Wide Web consortium) recommendation for communication between two applications.
- The WC3 is committed to improving the web by setting and promoting web-based standards. The W3C's goal is to create technical standards and guidelines for web technologies worldwide.
- It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications.
- SOAP is a communication protocol designed to communicate via Internet.
- SOAP can extend HTTP for XML messaging.
- SOAP provides data transport for Web services.
- SOAP can exchange complete documents or call a remote procedure.
- SOAP can be used for broadcasting a message.
- SOAP is the XML way of defining what information is sent and how.
- SOAP enables client applications to easily connect to remote services and invoke remote methods.

SOAP Web service working



The flow diagram shows how a SOAP client sends a message to a SOAP processor, which then invokes the appropriate web service and returns the response to the SOAP client.

Steps:

1. The SOAP client creates a SOAP message containing the request and sends it to the SOAP processor.
2. The SOAP processor receives the SOAP message and parses it to determine the requested web service and its parameters.
3. The SOAP processor invokes the requested web service and passes it the parameters specified in the SOAP message.
4. The web service performs the requested operation and returns a response to the SOAP processor.
5. The SOAP processor parses the response from the web service and creates a SOAP message containing the response.
6. The SOAP processor sends the SOAP message containing the response back to the SOAP client.
7. The SOAP client receives the SOAP message containing the response and parses it to extract the results.

UDDI Registry- (Universal Description, Discovery, and Integration):

The UDDI Registry is a directory of web services. It can be used by SOAP clients to find the WSDL for a web service. The WSDL describes the web service's operations and the parameters that each operation accepts and returns.

WSDL-(Web Services Description Language):

The Web Services Description Language (WSDL) is a XML-based language that is used to describe web services. It describes the web service's operations, the parameters that each operation accepts and returns, and the data types of the parameters.

SOAP Processor:

A SOAP processor is a software component that mediates between SOAP clients and web services. It receives SOAP messages from SOAP clients, invokes the appropriate web services, and returns the responses to the SOAP clients.

Web Service:

A web service is a software application that is exposed over the web. It can be invoked by SOAP clients to perform various operations.

SOAP Message:

A SOAP message is an XML-based message that is used to exchange data between SOAP clients and web services. It contains the request or response for a web service operation.

XML Schema:

XML Schema is a language that is used to define the structure and data types of XML documents. It is often used to define the structure of SOAP messages.

RESTful Web Services

- **REST-REpresentational State Transfer** is an architectural style for developing web services which uses HTTP for communication.
- The term, **REST** was coined by Roy Fielding in his doctoral dissertation in 2000.
- It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

- In **REST architecture**, a REST Server simply provides access to resources and REST client accesses and modifies the resources.
- Here each resource is identified by **URIs/ global IDs**.
- REST uses various representations to represent a resource like text, JSON, XML. JSON is the most popular one.

HTTP methods:

- **GET** – Provides a read only access to a resource.
- **POST** – Used to create a new resource.
- **DELETE** – Used to remove a resource.
- **PUT** – Used to update an existing resource or create a new resource.
- RESTful Web Services make use of **HTTP protocols** as a medium of communication between client and server.
- A client sends a message in form of a **HTTP Request** and the server responds in the form of an **HTTP Response**.
- This technique is termed as **Messaging**

In REST, everything is a **Resource** (data/functionality/web page) that is uniquely identified by URI (Uniform Resource Identifier).

Example:

Resource	URI
Data of an employee	http://www.infy.com/employees/john
Web Page	http://www.infy.com/about/infosys.html
Functionality that validates a credit card	http://www.infy.com/creditcards/1000001

Constructing a Standard URI

The following are important points to be considered while designing a URI –

- **Use Plural Noun** – Use plural noun to define resources. For example, we've used users to identify users as a resource.
- **Avoid using spaces** – Use underscore (_) or hyphen (-) when using a long resource name. For example, use `authorized_users` instead of `authorized%20users`.
- **Use lowercase letters** – Although URI is case-insensitive, it is a good practice to keep the url in lower case letters only.
- **Maintain Backward Compatibility** – As Web Service is a public service; a URI once made public should always be available. In case, URI

gets updated, redirect the older URI to a new URI using the **HTTP Status code, 300**.

- **Use HTTP Verb** – Always use HTTP Verb like GET, PUT and DELETE to do the operations on the resource.

Good Resources Representation

- REST does not impose any restriction on the format of a resource representation.
- A client can ask for JSON representation whereas another client may ask for XML representation of the same resource to the server and so on.
- It is the responsibility of the REST server to pass the client the resource in the format that the client understands.

Following are some important points to be considered while designing a representation format of a resource in RESTful Web Services.

- **Understandability** – Both the Server and the Client should be able to understand and utilize the representation format of the resource.
- **Completeness** – Format should be able to represent a resource completely. For example, a resource can contain another resource. Format should be able to represent simple as well as complex structures of resources.
- **Linkability** – A resource can have a linkage to another resource; a format should be able to handle such situations.

However, at present most of the web services are representing resources using either XML or JSON format. There are plenty of libraries and tools available to understand, parse, and modify XML and JSON data.

The resources of REST can be **represented** in many ways that include JSON, XML, HTML, etc.,

For example, if the URI, <http://www.infy.com/employees/john>, is hit using HTTP GET method, the server may respond back with John's details in JSON format.

```
1. {  
2.   "name": "John",  
3.   "Level": 5,  
4.   "address": "Hyderbad",
```

```
5.   "phoneno":998765432
6. }
```

There are possibilities to expose John's details in XML format as well.

```
1. <employee>
2. <name>John</name>
3. <address>hyderabad</address>
4. <level>5</level>
5. <phoneno>998765432</phoneno>
6. </employee>
```

This means the server can represent the resource in many ways.

So now what does **State Transfer** mean?

When we make a request to a resource like <http://www.infy.com/employees/john> using HTTP, we may be asking for the **current "state" of this resource or its desired "state"**.

We can use the same URI to indicate that, we want to:

- Retrieve the current value of an Employee John (current state)
- Perform Update operation on employee John's data (desired state) etc.

The server will understand which operation has to be invoked based on the HTTP method that was used to make a call.

In short it means that a client and server exchange, representation of a resource, which reflects its current state(GET) or its desired state(PUT).

Why RESTful Web Services?

Sample SOAP Request to GreetingService:

```
1. Sample SOAP Request for GreetingService:
```

```

2. <?xml version="1.0" encoding="UTF-8"?>
3. <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
4.   <SOAP-ENV:Header/>
5.   <S:Body>
6.     <ns2:hello xmlns:ns2="http://infy.com/">
7.       <name>Rekha</name>
8.     </ns2:hello>
9.   </S:Body>
10.</S:Envelope>

```

This XML will be embedded as "payload" inside the **SOAP** request envelope which in turn will be wrapped using HTTP request and then, will be sent using HTTP POST.

The SOAP Response would be:

```

1. Sample SOAP Response for GreetingService:
2. <?xml version="1.0" encoding="UTF-8"?><S:Envelope
   xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
   ENV="http://schemas.xmlsoap.org/soap/envelope/">
3.   <SOAP-ENV:Header/>
4.   <S:Body>
5.     <ns2:helloResponse xmlns:ns2="http://infy.com/">
6.       <return>Hello Rekha,Welcome to the world of Web Services
       !</return>
7.     </ns2:helloResponse>
8.   </S:Body>
9. </S:Envelope>
10.

```

If the same functionality is developed as a RESTful Web service, the request will be something like this,

```
GET http://<server name>:<portno>/GreetingService/greetings?name=Rekha
```

It is an HTTP GET request to a web service or endpoint at the URL
`http://<server name>:<portno>/GreetingService/greetings`.

The request includes a query parameter, `name=Rekha`, which is typically used to pass data to the web service for processing.

- **GET**: specifies the HTTP method being used, which is "GET" in this case. A GET request is used to retrieve data from the specified URL.
- **<server name>**: It is a placeholder for the actual server name or hostname where the web service is hosted.
- **<portno>**: It is a placeholder for the port number on which the web service is listening.
- **/GreetingService/greetings**: This is the path or endpoint of the web service. It specifies the location of the resource you want to access on the server. In this case, it seems to be related to a "GreetingService" and "greetings" resource.
- **?name=Rekha**: This is the **query parameter** included in the URL. Query parameters are used to pass data to the web service. In this case, it's passing a parameter named "name" with the value "Rekha."

And, the response will be like the one that follows.

```
{ "response": "Hello Rekha, Welcome to the world of Web Services" }
```

SOAP vs REST

Both SOAP and REST provide support for building applications based on Service Oriented Architecture.

SOAP-based Web Service

SOAP is a protocol (defines the way how messages have to be sent) which is transported over a transport layer Protocol, mostly HTTP.

SOAP messages can be **transported** using other transport layer protocols (SMTP, FTP) as well.

Data (XML/JSON) wrapped in a SOAP message, comes with an additional overhead of XML processing at the client and server side as well.

SOAP-based reads **cannot be cached** because SOAP messages are sent using the HTTP POST method.

A predefined formal contract is required between the consumer and the provider.

RESTful Web Service

REST is an architectural style.

REST leverages HTTP Protocol.

REST is straight forward in handling requests and responses. Since REST supports **multiple data formats** other than XML, there is no overhead of wrapping the request and response data in XML. Also, REST uses the existing HTTP protocol for communication. All these factors together make REST, **lightweight**.

REST-based reads can be cached.

Formal contract is not mandatory. Service description can be done if needed, using a less verbose way with the help of WADL.

The **Web Application Description Language (WADL)** is a machine-readable XML description of HTTP-based web services.

When REST?

To better understand REST and SOAP, let us take a simple **analogy of mailing a letter**.

- SOAP is like an **envelope** which is used to send/receive messages
- REST is like a **postcard**
 - Easier to handle (by the receiver)
 - Less usage of paper (i.e., consume less bandwidth)
 - Has a short content (REST requests are not really limited in length, especially, if POST is used instead of GET)

So eventually, if we want to develop an application which

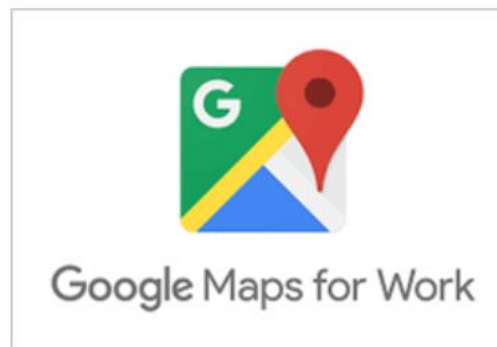
- is light-weight
- can be accessed by any other application irrespective of the differences in terms of language and platform.
- leverages HTTP protocol
- is stateless
- does not require a specific message format to carry the request and response (like SOAP)
- represents data using various MIME (MultiPurpose Internet Mail Extension) types which all applications can consume

then, **RESTful** Web Service is the choice.

REST APIs are **stateless**, meaning that calls can be made independently of one another, and each call contains all of the data necessary to complete itself successfully.

REST in Real World

The **Google Maps API** provides web services as an interface for requesting Maps API data from external services.



PayPal: Uses REST based payment services to accept online and mobile payments in an easier and secure way.



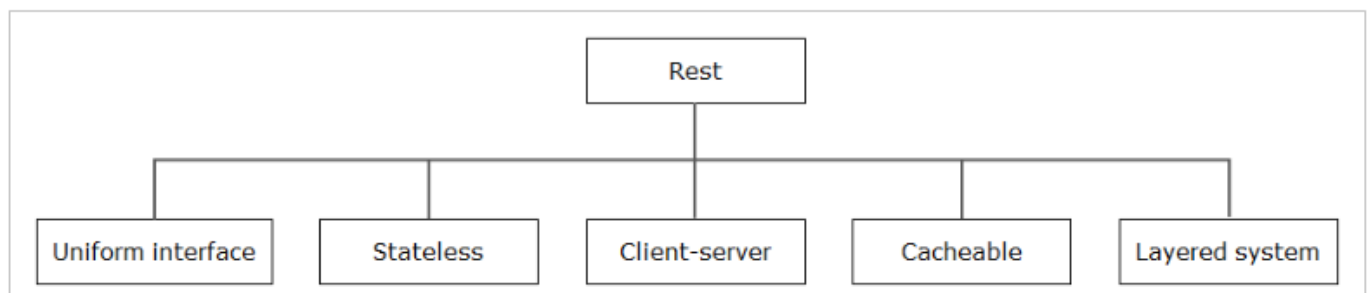
YouTube: In order to fetch the list of videos available on YouTube, A REST API service named "search" is provided by YouTube.



One example is the **GitHub API**. GitHub is a web-based platform for version control and collaboration that provides an API for developers to interact with its data programmatically. The GitHub API is well-designed and adheres to RESTful principles.

REST Principles

RESTful web service or REpresentational State Transfer (REST) adheres to a set of architectural principles which makes RESTful web services **fast, scalable** and **modifiable**.



Uniform Interface

- In REST, data and functionality are considered as resources. Every resource has a representation (XML/JSON) and a URI to identify the same.
- Resources are manipulated using HTTP's GET, POST, PUT and DELETE methods.

We can use the HTTP Methods for the following operations on the resources.

HTTP Method	CRUD Operation	Description
GET	Read	Fetches a resource
POST	Create	Adds an existing resource to a server
PUT	Update	Transfers a resource to the server and overwrites the existing resource
DELETE	Delete	Discards resources

Example: If Customer is a resource, then Customer details can be added, deleted, updated or retrieved.

If the blog is another resource, then the operations that we can perform on the blog will be read, create, update or delete.

Advantage: Single URI can perform multiple operations on the resource.

Stateless

- The interaction that takes place between the service provider and consumer is stateless.
- Being stateless, the server will never store the state information of its clients. Rather, every request from a service consumer should carry all the necessary information to make the server understand and process the request.

Advantage: Statelessness ensures that the requests from a consumer should get treated independently. This helps to **scale the APIs** (making them available in multiple servers) to support a large number of clients concurrently. Any server is free to serve any client since there is **no session related dependency**. Also, REST APIs stay less complex as server-side state synchronization logic is removed. On top, the performance of APIs is improvised because of less space occupation (no space is required for state-related info).

Client-Server

- Enforces the separation of concerns that helps to establish a **distributed architecture**. And, this is the most foundational constraint. Because of this distributed architecture with separation of concerns, the change in one component will not affect other components. In simple words, the components stay loosely coupled making them more manageable.

Advantage: Supports the independent evolution of the client and server side logic.

Cacheable

- Service consumers can cache the response data and reuse the same. For example, HTTP GET and HEAD operations are cacheable.
- RESTful Web Services use the HTTP protocol as a carrier for the data. And, they can make use of the metadata that is available in the HTTP headers to enable caching.
- For example, Cache-Control headers are used to optimize caching to enhance performance.

Advantage: Enhances performance

Layered System

- REST based solution comprises of multiple architectural layers.
- The Consumer may interact directly with the actual provider or through the service that is residing in the middleware layer.

Advantage: Information hiding with a layered architecture helps to simplify the design of distributed architecture. Also, individual layers can be deployed and evolved independently.

Code-On-Demand

- This is an optional constraint which is primarily intended to allow client-side logic to be modified without doing changes in the server-side logic.
- For example, a service can be designed in such a way that it dynamically pushes part of the logic to the consumers.

Advantage: Logic that is present in the client end (such as Web browsers) can be modified/ maintained independently without affecting the logic at the server side.

Advantages of REST API:

- REST API is easy to understand and learn, due to its simplicity, known API.
- With REST API, being able to organize complicated applications & makes it easy to use resources.
- The high load can be managed with help out of HTTP proxy server & cache.
- Use standard HTTP procedure call- outs to retrieve data and requests.
- Brings flexibility formats by serializing data in XML or JSON format.

Disadvantages or Challenges in REST:

- **Lack of state:** most web applications require stateful mechanisms. Suppose you purchase a website which has a mechanism to have a shopping cart. It is required to know the number of items in the shopping cart before the actual purchase is made. This burden of maintaining the state lies on the client, which makes the client application heavy and difficult to maintain.
- **Lack of security:** REST doesn't impose security such as SOAP. That is the reason REST is appropriate for public URLs, but it is not good for confidential data passage between client and server.

How to create RESTful Services

RESTful Web services can be developed and consumed in Java using **JAX-RS API** or **Spring REST**.



JAX-RS stands for JAVA API for RESTful Web Services.

JAX-RS is a JAVA based programming language API and specification to provide support for created RESTful Web Services

JAX-RS 2.0 and above are annotation-driven which eases the development of Java-based RESTful services.

JAX-RS uses annotations available from Java SE 5 to simplify the development of JAVA based web services creation and deployment.

By using JAX-RS technology, REST applications are simpler to develop, simpler to consume, and simpler to scale when compared to other types of distributed systems

It also provides supports for creating clients for RESTful Web Services.

Specifications

Following are the most commonly used annotations to map a resource as a web service resource.

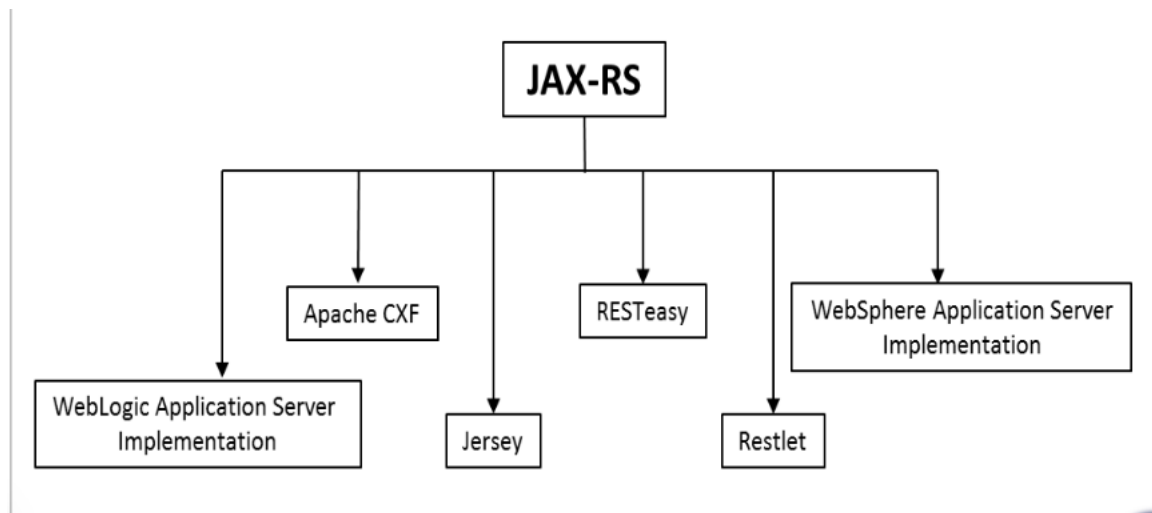
Annotation	Description
@Path	<p>The @Path annotation's value is a relative URI path indicating where the Java class will be hosted:</p> <p>for example, /helloworld. You can also embed variables in the URIs to make a URI path template. For example, you could ask for the name of a user and pass it to the application as a variable in the URI: /helloworld/{username}.</p>
@GET	<p>The @GET annotation is a request method designator and corresponds to the similarly named HTTP method.</p> <p>The Java method annotated with this request method designator will process HTTP GET requests.</p>
@POST	<p>The @POST annotation is a request method designator and corresponds to the similarly named HTTP method.</p> <p>The Java method annotated with this request method designator will process HTTP POST requests.</p>
@PUT	<p>The @PUT annotation is a request method designator and corresponds to the similarly named HTTP method.</p> <p>The Java method annotated with this request method designator will process HTTP PUT requests.</p>
@DELETE	<p>The @DELETE annotation is a request method designator and corresponds to the similarly named HTTP method.</p> <p>The Java method annotated with this request method designator will process HTTP DELETE requests.</p>
@HEAD	<p>The @HEAD annotation is a request method designator and</p>

Annotation	Description
	<p>corresponds to the similarly named HTTP method.</p> <p>used to get status of method availability.</p> <p>The Java method annotated with this request method designator will process HTTP HEAD requests.</p>
@PathParam	The @PathParam annotation is a type of parameter that you can extract for use in your resource class.
@QueryParam	<p>The @QueryParam annotation is a type of parameter that you can extract for use in your resource class.</p> <p>Query parameters are extracted from the request URI query parameters.</p>
@Consumes	The @Consumes annotation states the HTTP Request type.
@Produces	The @Produces annotation states the HTTP Response generated by web service.

JAX-RS implementations

JAX-RS is simply a specification and we need actual implementations to write web services. There are many vendors in the market who adhere to the standards of JAX-RS such as, Jersey and RESTEasy.

Below are some of the JAX-RS implementations.



Spring REST

Spring is an end to end framework which has a lot of modules in a highly organized way.

One among such modules is, Spring MVC that provides the support for REST in addition to the standard MVC support.

The developers who use Spring MVC can go for constructing RESTful services in an effortless way. Also, it is guaranteed that the application stays lightweight as the build path is not polluted with many external dependencies, just to support REST.

Spring Web MVC is the source of Spring REST.

Spring provides support for creating RESTful web services using Spring MVC. Availing this support requires, Spring version 3.0 and above.