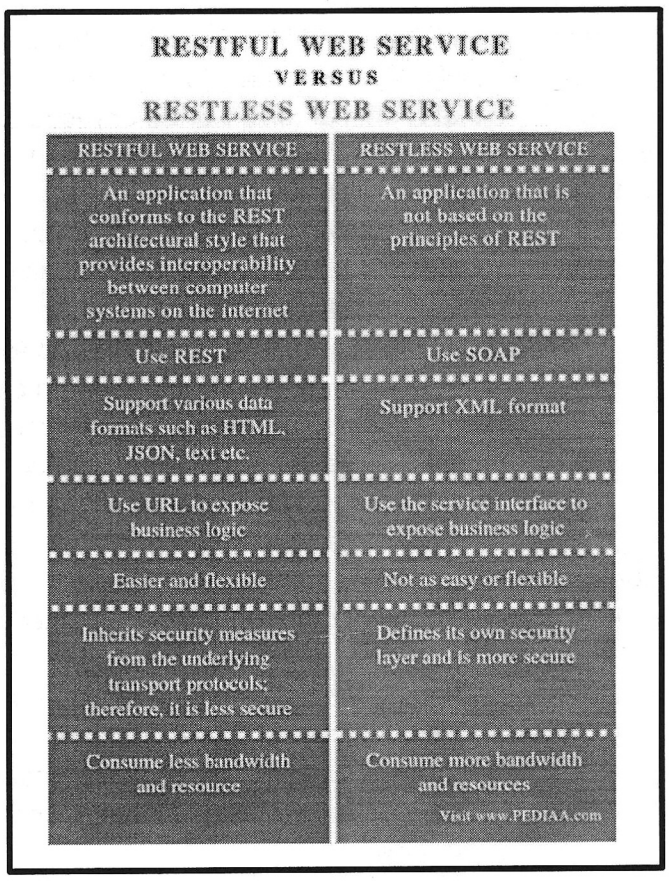
REST & Web Services

53. What is a web service?
    a. A collection of open protocols and standards used for exchanging data between applications or systems.
    b. A web service is any service that:
        i. Is available over the Internet
        ii. Uses a standardized XML messaging system
        iii. Is not tied to any one operating system or programming language
        iv. Is self-describing via a common XML grammar
        v. Is discoverable via a simple find mechanism
54. How does a RESTful web service differ from a normal web service?
    a. A RESTful Web Service is a lightweight, maintainable, and scalable service that is built on the REST architecture
    b. Differences:
        i. A RESTful Web Service will expose an API from your application in a secure, uniform, stateless manner to the calling client.
        ii. The calling client can perform predefined operations using the RESTful service.
        iii. The underlying protocol for REST is HTTP.
        iv. Chart for comparsion between RESTful and RESTless web services:



55. What is REST? What are its constraints?
    a. REST stands for REpresentational State Transfer.
    b. REST is a way to access resources which lie in a particular environment.
    c. Roy Fielding describes how a well-designed web application is supposed to behave like with these 6 constraints:
        i. Uniform interface
            1. Having a uniform way of interacting with a given server regardless of device or type of application
            2. Guidelines:
                a. Resource-Based: Individual resources are identified in requests. For example: API/users.
                b. Manipulation of Resources Through Representations: Client has representation of resource and it contains enough information to modify or delete the resource on the server, provided it has permission to do so. Example: Usually user get a user id when user request for a list of users and then use that id to delete or modify that particular user.
                c. Self-descriptive Messages: Each message includes enough information to describe how to process the message so that server can easily analyses the request.

d. Hypermedia as the Engine of Application State (HATEOAS): It need to include links for each response so that client can discover other resources easily.

ii. Client-server
1. Should have a client-server architecture
2. Neither client nor server are concerned with what their counterpart is doing and thus each evolve independently
a. REMEMBER:
i. A Client is someone who is requesting resources and are not concerned with data storage, which remains internal to each server
ii. A Server is someone who holds the resources and are not concerned with the user interface or user state.

iii. Stateless
1. The state to handle the request is contained within the request itself
a. Meaning the server would not store anything related to the session.
b. Think of it as "Browser State Amnesia" as the browser will forget the current session state after the request is sent to the server.
c. Advantage: enables greater availability since the server does not have to maintain, update or communicate that session state with every request
d. Drawback: Reduction of the scope of network optimization the larger the client's request for data becomes

iv. Cacheable
1. Every response should include whether the response is cacheable or not and for how much duration responses can be cached at the client side.
2. Client will return the data from its cache for any subsequent request and there would be no need to send the request again to the server.
3. A well-managed caching partially or completely eliminates some client–server interactions, further improving availability and performance. But sometime there are chances that user may receive stale data.

v. Layered system
1. An application architecture needs to be composed of multiple layers.
2. Each layer doesn't know anything about any layer other than that of immediate layer and there can be lot of intermediate servers between client and the end server.

3. Intermediary servers may improve system availability by enabling load-balancing and by providing shared caches.

vi. Code on demand (optional)

1. Servers can provide executable code (in the form of snippets or scripts) to the client.

56. What is SOAP? Name and describe its message elements.

a. SOAP = Simple Object Access Protocol

i. A standards-based web services access protocol that been around longer than REST

ii. Similar to REST, SOAP is used to "expose" and "consume" webservices

b. XML-based web service protocol originally developed by Microsoft

c. Platform and language independent

d. Legacy protocol: most companies switching to RESTful web services

e. Can be used in conjunction with any transport-layer protocol (HTTP, SMTP, FTP, etc)

i. When used with HTTP, POST requests are used

ii. HTTP must set that content type to XML

f. Uses a contract (WSDL)

i. WSDL = Web service description language

1. Web Service Description Language
2. XML file that describes the service
3. "Contract" or "Endpoint"
4. WSDL elements are placed within the namespaces like this:
5. xmlns:xsd, xmlns:soap
6. WSDL specifies ports, messages, operations, services

ii. Contract first vs contract last development

g. Built-in security

h. SOAP message elements

i. envelope - (mandatory) defines the start and end of a message

ii. header - (optional) specifies attributes)

iii. body - (mandatory) this is the XML data

iv. fault - (optional) describes error that may have occurred

57. What are the differences between SOAP and REST?

Difference	SOAP	REST
Style	Protocol	Architectural style
Function	Function-driven: transfer structured information	Data-driven: access a resoruce for data
Data format	Only uses XML	Permits many data formats, including plain text, HTML, XML, and JSON
Security	Supports WS-Security and SSL	Supports SSL and HTTPS
Bandwidth	Requires more resources and bandwidth	Requires fewer resources and is lightweight
Data cache	Can not be cached	Can be cached
Payload handling	Has a strict communication contract and needs knowledge of everything before any interaction	Needs no knowledge of the API
ACID compliance	Has built-in ACID compliance to reduce anomalies	Lacks ACID compliance