

UNIT 2

1.

What is REST? Explain RESTful services.

ANS 1:

- Just like SOAP (Simple Object Access Protocol), which is used to develop web services by XML method, RESTful web services use web protocol i.e. HTTP protocol method.
- They have the feature like scalability, maintainability, help multiple application communication built on various programming languages etc.
- RESTful web service implementation defines the method of accessing various resources which are required by the client and he has sent the request to the server through the web browser.
- The important aspects of this implementation include:
 - Resources
 - Request Headers
 - Request Body
 - Response Body
 - Status codes

2.	List and Explain the various HTTP Methods required for creating RESTful Web Services.																				
ANS 2:	<table border="1" data-bbox="225 185 1353 1144"> <thead> <tr> <th data-bbox="225 185 408 237">Method</th><th data-bbox="408 185 1353 237">Description</th></tr> </thead> <tbody> <tr> <td data-bbox="225 237 408 322">GET</td><td data-bbox="408 237 1353 322">This method is used for retrieving resources from the server by using the given URI.</td></tr> <tr> <td data-bbox="225 322 408 407">HEAD</td><td data-bbox="408 322 1353 407">This method is the same as the GET request, but it only transfers the status line and the header section without the response body.</td></tr> <tr> <td data-bbox="225 407 408 573">POST</td><td data-bbox="408 407 1353 573">This method is used for posting data to the server. The server stores the data (entity) as a new subordinate of the resource identified by the URI. If you execute POST multiple times on a resource, it may yield different results.</td></tr> <tr> <td data-bbox="225 573 408 698">PUT</td><td data-bbox="408 573 1353 698">This method is used for updating the resource pointed at by the URI. If the URI does not point to an existing resource, the server can create the resource with that URI.</td></tr> <tr> <td data-bbox="225 698 408 750">DELETE</td><td data-bbox="408 698 1353 750">This method deletes the resource pointed at by the URI.</td></tr> <tr> <td data-bbox="225 750 408 875">TRACE</td><td data-bbox="408 750 1353 875">This method is used for echoing the contents of the received request. This is useful for the debugging purpose with which the client can see what changes (if any) have been made by the intermediate servers.</td></tr> <tr> <td data-bbox="225 875 408 963">OPTIONS</td><td data-bbox="408 875 1353 963">This method returns the HTTP methods that the server supports for the specified URI.</td></tr> <tr> <td data-bbox="225 963 408 1048">CONNECT</td><td data-bbox="408 963 1353 1048">This method is used for establishing a connection to the target server over HTTP.</td></tr> <tr> <td data-bbox="225 1048 408 1144">PATCH</td><td data-bbox="408 1048 1353 1144">This method is used for applying partial modifications to a resource identified by the URI.</td></tr> </tbody> </table>	Method	Description	GET	This method is used for retrieving resources from the server by using the given URI.	HEAD	This method is the same as the GET request, but it only transfers the status line and the header section without the response body.	POST	This method is used for posting data to the server. The server stores the data (entity) as a new subordinate of the resource identified by the URI. If you execute POST multiple times on a resource, it may yield different results.	PUT	This method is used for updating the resource pointed at by the URI. If the URI does not point to an existing resource, the server can create the resource with that URI.	DELETE	This method deletes the resource pointed at by the URI.	TRACE	This method is used for echoing the contents of the received request. This is useful for the debugging purpose with which the client can see what changes (if any) have been made by the intermediate servers.	OPTIONS	This method returns the HTTP methods that the server supports for the specified URI.	CONNECT	This method is used for establishing a connection to the target server over HTTP.	PATCH	This method is used for applying partial modifications to a resource identified by the URI.
Method	Description																				
GET	This method is used for retrieving resources from the server by using the given URI.																				
HEAD	This method is the same as the GET request, but it only transfers the status line and the header section without the response body.																				
POST	This method is used for posting data to the server. The server stores the data (entity) as a new subordinate of the resource identified by the URI. If you execute POST multiple times on a resource, it may yield different results.																				
PUT	This method is used for updating the resource pointed at by the URI. If the URI does not point to an existing resource, the server can create the resource with that URI.																				
DELETE	This method deletes the resource pointed at by the URI.																				
TRACE	This method is used for echoing the contents of the received request. This is useful for the debugging purpose with which the client can see what changes (if any) have been made by the intermediate servers.																				
OPTIONS	This method returns the HTTP methods that the server supports for the specified URI.																				
CONNECT	This method is used for establishing a connection to the target server over HTTP.																				
PATCH	This method is used for applying partial modifications to a resource identified by the URI.																				
3.	What is a Resource? How is it represented?																				
ANS 3:	<ul data-bbox="320 1196 1414 1529" style="list-style-type: none"> • A RESTful resource is anything that is addressable over the Web. • By addressable, we mean resources that can be accessed and transferred between clients and servers. • Subsequently, a resource is a logical, temporal mapping to a concept in the problem domain for which we are implementing a solution. • Here are some examples of the REST resources: <ul data-bbox="416 1429 991 1529" style="list-style-type: none"> ○ A news story ○ The temperature in NY at 4:00 p.m. EST ○ A tax return stored in the IRS database <p data-bbox="225 1536 592 1570"><i>Representation of Resources</i></p> <ul data-bbox="272 1576 1422 2132" style="list-style-type: none"> • The representation of resources is what is sent back and forth between clients and servers in a RESTful system. • A representation is a temporal state of the actual data located in some storage device at the time of a request. • In general terms, it is a binary stream together with its metadata that describes how the stream is to be consumed by the client. • The metadata can also contain extra information about the resource, for example, validation, encryption information, or extra code to be executed at runtime. • Throughout the life of a web service, there may be a variety of clients requesting resources. • Different clients can consume different representations of the same resource. • Therefore, a representation can take various forms, such as an image, a text file, an XML, or a JSON format. • However, all clients will use the same URI with appropriate Accept header values for accessing the same resource in different representations. 																				

	<ul style="list-style-type: none"> • For the human-generated requests through a web browser, a representation is typically in the form of an HTML page. • For automated requests from the other web services, readability is not as important and a more efficient representation, such as JSON or XML, can be used.
4.	Explain the characteristics required for Good Resource Representation.
ANS 4:	<ul style="list-style-type: none"> • REST components perform actions on a resource by using a representation to capture the current or intended state of that resource and transferring that representation between components. • The state of resource at any timestamp is known as resource representation. • A representation consists of data, metadata describing the data, and, on occasion, metadata to describe the metadata. • The representation of resources is what is sent back and forth between clients and servers in a RESTful system. • A representation is a temporal state of the actual data located in some storage device at the time of a request. • In general terms, it is a binary stream together with its metadata that describes how the stream is to be consumed by the client. • The metadata can also contain extra information about the resource, for example, validation, encryption information, or extra code to be executed at runtime. • Throughout the life of a web service, there may be a variety of clients requesting resources. • Different clients can consume different representations of the same resource. • Therefore, a representation can take various forms, such as an image, a text file, an XML, or a JSON format. • However, all clients will use the same URI with appropriate Accept header values for accessing the same resource in different representations. • For the human-generated requests through a web browser, a representation is typically in the form of an HTML page. • For automated requests from the other web services, readability is not as important and a more efficient representation, such as JSON or XML, can be used. • 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 can understand. • Following points can be considered while designing a representation format of a resource in RESTful Web Services. <ul style="list-style-type: none"> ➤ Understandability - Both the Server and the Client should be able to understand and utilize the representation format of the resource. ➤ Completeness – The format should be able to represent a resource completely such that it can 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 as defined in the HATEOAS.

5.	<p>Explain with the help of a diagram the various parts of</p> <ol style="list-style-type: none"> HTTP Request HTTP Response
ANS 5:	<ul style="list-style-type: none"> The request and response messages sent between the client and the server: <div data-bbox="322 259 1342 1019" data-label="Diagram"> <pre> sequenceDiagram participant Client participant WebServer as Web Server Note over Client,WebServer: Request Client->>WebServer: GET /index.html HTTP/1.1 Host: www.example.com User-Agent: Mozilla/5.0 Accept: text/html Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Connection: keep-alive Note over Client,WebServer: Response WebServer-->>Client: HTTP/1.1 200 OK Accept-Ranges: bytes Cache-Control: max-age=604800 Content-Type: text/html Date: Wed, 03 Dec 2014 15:05:59 GMT Content-Length: 1270 <html> <head> <title>An Example Page</title> </head> <body> Hello World ! </body> </html> </pre> </div> <ol style="list-style-type: none"> HTTP Request <ul style="list-style-type: none"> The sample request looks like the following: <pre> GET /index.html HTTP/1.1 Host: www.example.com User-Agent: Mozilla/5.0 Accept: text/htmlAccept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Connection: keep-alive </pre> The general format for the request line is an HTTP command, followed by the resource to retrieve, and the HTTP version supported by the client. The client can be any application that understands HTTP, although this example refers to a web browser as the client. The request line and other header fields must end with a carriage return character followed by a line feed character. In the preceding example, the browser instructs the server to get the index.html file through the HTTP 1.1 protocol. The rest of the information that you may see in the request message is the HTTP header values for use by the server. The header fields are colon-separated key-value pairs in the plain-text format, terminated by a carriage return followed by a line feed character. The header fields in the request, such as the acceptable content types, languages, and connection type, are the operating parameters for an HTTP transaction. The server can use this information while preparing the response for the request.

	<ul style="list-style-type: none"> • A blank line is used at the end of the header to indicate the end of the header portion in a request. • The last part of an HTTP request is the HTTP body. Typically, the body is left blank unless the client has some data to submit to the server. In our example, the body part is empty as this is a GET request for retrieving a page from the server. <p><i>b. HTTP Response</i></p> <ul style="list-style-type: none"> • The response is made up of the reply status code from the server, followed by the HTTP header and a response content body: • <pre> HTTP/1.1 200 OK Accept-Ranges: bytes Cache-Control: max-age=604800 Content-Type: text/html Date: Wed, 03 Dec 2014 15:05:59 GMT Content-Length: 1270 <html> <head> <title>An Example Page</title> </head> <body> Hello World ! </body> </html>.</pre> • The first line in the response is a status line. It contains the HTTP version that the server is using, followed by a numeric status code and its associated textual phrase. • The status code indicates one of the following parameters: informational codes, success of the request, client error, server error, or redirection of the request. • In our example, the status line is as follows: HTTP/1.1 200 OK • The next item in the response is the HTTP response header. Similar to the request header, the response header follows the colon-separated name-value pair format terminated by the carriage return and line feed characters. • The HTTP response header can contain useful information about the resource being fetched, the server hosting the resource, and some parameters controlling the client behaviour while dealing with the resource, such as content type, cache expiry, and refresh rate. • The last part of the response is the response body. Upon the successful processing of the request, the server will add the requested resource in the HTTP response body. It can be HTML, binary data, image, video, text, XML, JSON, and so on. • Once the response body has been sent to the requestor, the HTTP server will disconnect if the connection created during the request is not of the keep-alive type (using the Connection: keep-alive header).
6.	Explain the important points considered when constructing a standard URI.
ANS 6:	<ul style="list-style-type: none"> • <i>Constructing a Standard URI</i> <ul style="list-style-type: none"> → The following are important points to be considered while designing a URI – <ul style="list-style-type: none"> * Use Plural Noun – Use plural noun to define resources. For example, we've used users to identify users as a resource.

	<ul style="list-style-type: none"> * Avoid using spaces – Use underscore () or hyphen (-) when using a long resource name. For example, use <code>authorized_users</code> instead of <code>authorized%20users</code>. * 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. It is not good to use operations name in the URI. <p>– Example</p> <ul style="list-style-type: none"> ❖ Following is an example of a poor URI to fetch a user. <i><code>http://localhost:8080/UserManagement/rest/UserService/getUser/1</code></i> ❖ Following is an example of a good URI to fetch a user. <i><code>http://localhost:8080/UserManagement/rest/UserService/users/1</code></i>
7.	Write a short note on Statelessness
ANS 7:	<ul style="list-style-type: none"> • In REST, ST itself defines State Transfer and Statelessness means complete isolation. • This means, the state of the client's application is never stored on the server and is passed on. • In this process, the clients send all the information that is required for the server to fulfil the HTTP request that has been sent. • Thus, every client request and the responses are independent of the other with complete assurance of providing required information. • Every client passes a 'session identifier' which also acts as an identifier for each session.
8.	List the advantages and disadvantages of Statelessness.
ANS 8:	<p><i>Advantages:</i></p> <ul style="list-style-type: none"> • Every method required for communication is identified as an independent method i.e. there are no dependencies to other methods. • Any previous communication with the client and server is not maintained and thus the whole process is very much simplified. • If any information or metadata used earlier is required in another method, then the client sends again that information with HTTP request. • HTTP protocol and REST web service, both share the feature of statelessness. <p><i>Disadvantages:</i></p> <ul style="list-style-type: none"> • In every HTTP request from the client, the availability of some information regarding the client state is required by the web service.
9.	Write a short note on Caching
ANS 9:	<ul style="list-style-type: none"> • Caching is the process in which server response is stored so that a cached copy can be used when required and there is no need of generating the same response again. • This process not only reduces the server load but in turn increases the scalability and performance of the server. • Only the client can cache the response and that too for a limited period of time.

	<ul style="list-style-type: none"> • Mentioned below are the header of the resources and their brief description so that they can be identified for the caching process: <ul style="list-style-type: none"> + Time and Date of resource creation + Time and date of resource modification that usually stores the last detail. + Cache control header + Time and date at which the cached resource will expire. + The age which determines the time from when the resource has been fetched.
10.	Write a note on best practices for security while designing RESTful Web Services.
ANS 10:	<ul style="list-style-type: none"> • To design a secure RESTful web service, there are some best practices or say points that should be considered. • These are explained as follows: <ul style="list-style-type: none"> + Every input on the server should be validated. + Input should be well formed. + Never pass any sensitive data through URL. + For any session, the user should be authenticated. + Only HTTP error messages should be used for indicating any fault. + Use message format that is easily understood and is required by the client. + Unified Resource Identifier should be descriptive and easily understood.
11.	Write a short note on “JAX-RS”.
ANS 11:	<ul style="list-style-type: none"> • JAX-RS is defined as the Java API for RESTful web service. • Among multiple libraries and framework, this is considered as the most suitable Java programming language-based API which supports RESTful web service. • Some of the implementations of JAX-RS are: <ul style="list-style-type: none"> ○ Jersey ○ RESTEasy ○ Apache CFX ○ Play • Example: - <pre> import javax.ws.rs.GET; import javax.ws.rs.Path; import javax.ws.rs.Produces; import javax.ws.rs.core.MediaType; // The browser requests per default the HTML MIME type. //Sets the path to base URL + /hello @Path("/hello") public class Hello { // This method is called if TEXT_PLAIN is request @GET @Produces(MediaType.TEXT_PLAIN) public String sayPlainTextHello() { return "Hello Jersey"; } // This method is called if XML is request @GET @Produces(MediaType.TEXT_XML) </pre>

	<pre> public String sayXMLHello() { return "<?xml version='1.0'?" + "<hello> Hello Jersey" + "</hello>"; } // This method is called if HTML is request @GET @Produces(MediaType.TEXT_HTML) public String sayHtmlHello() { return "<html> " + "<title>" + "Hello World" + "</title>" + "<body><h1>" + "Hello World" + "</body></h1>" + "</html>"; } } </pre>
12.	Explain the HTTP and its working with diagram.
ANS 12:	<ul style="list-style-type: none"> • HTTP is a TCP/IP based communication protocol, that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. • The default port is TCP 80, but other ports can be used as well. • It provides a standardized way for computers to communicate with each other. • HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests. • HTTP protocol functions like a combination of FTP and SMTP. • It is similar to FTP because it transfers files and uses the services of TCP. • However, it is much simpler than FTP because it uses only one TCP connection. • There is no separate control connection; only data is transferred between the client and the server. • HTTP protocol is like SMTP protocol because the data transferred between the client and the server look like SMTP messages. • In addition, MIME-like headers control the format of the messages. • However, HTTP differs from SMTP in the way the messages are sent from the client to the server and from the server to the client. • Unlike SMTP, the HTTP messages are not destined to be read by humans; they are read and interpreted by the HTTP server and HTTP client (browser). • SMTP messages are stored and forwarded, but HTTP messages are delivered immediately. • The idea of HTTP protocol is very simple. • A client sends a request, which looks like mail, to the server. • The server sends the response, which looks like a mail reply, to the client. • The request and response messages carry data in the form of a letter with MIME-like format. • Working: <ul style="list-style-type: none"> ○ The three main HTTP message types are GET, POST, and HEAD. <ul style="list-style-type: none"> + HTTP GET messages sent to a server contain only a <u>URL</u>. Zero or more optional data parameters may be appended to the end of the URL. The server processes the optional data portion of the URL, if present, and returns the result (a web page or element of a web page) to the browser. + HTTP POST messages place any optional data parameters in the body of the request message rather than adding them to the end of the URL. + HTTP HEAD request works the same as GET requests. Instead of replying with the full contents of the URL, the server sends back only the header information (contained inside the HTML section).



Hypertext Transfer Protocol (HTTP) Request and Response.

13. Differentiate between HTTP and HTTPS

ANS 13:

HTTP	HTTPS
HTTP URLs begin with “http://”	HTTPS URLs begin with “https://”.(That “S” in the abbreviation comes from the word Secure)
HTTP uses port number 80 for communication	HTTPS uses 443 for communication
HTTP is considered to be unsecure since it remains focused on presenting the information, but cares less about the way this information travels from one place to another.	HTTPS is secure,uses[<u>Secure Sockets Layer</u> is the standard security technology that establishes an encrypted connection between a web server and a browser.]
No certificates required for validation in case of HTTP	HTTPS requires SSL Digital Certificate
HTTP Works at Application Layer	HTTPS works at Transport Layer

14. What is REST and RESTful webservices? Explain.

ANS 14:

REST-

- REST stands for **REpresentational State Transfer**.
- REST is an architecture all about client-server communication.
- REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP.
- REST uses URI to expose business logic.
- REST **permits different** data format such as Plain text, HTML, XML, JSON etc.

RESTful Web Service-

- Just like SOAP (Simple Object Access Protocol), which is used to develop web services by XML method, RESTful web services use web protocol i.e. HTTP protocol method. They have the feature like scalability, maintainability, help multiple application communication built on various programming languages etc.

	<ul style="list-style-type: none"> • RESTful web service implementation defines the method of accessing various resources which are required by the client and he has sent the request to the server through the web browser. • The important aspects of this implementation include: <ul style="list-style-type: none"> + Resources + Request Headers + Request Body + Response Body + Status codes
15.	List and explain the core constraint of RESTful system.
ANS 15:	<ul style="list-style-type: none"> □ Six guiding constraints define a RESTful system. □ These constraints restrict the ways that the server can process and respond to client requests so that, by operating within these constraints, the service gains desirable <u>non-functional properties</u>, such as performance, scalability, simplicity, modifiability, visibility, portability, and reliability. □ If a service violates any of the required constraints, it cannot be considered RESTful. □ The formal REST constraints are as follows: □ <u>CLIENT SERVER ARCHITECTURE</u> <ul style="list-style-type: none"> + The principle behind the client–server constraints is the separation of concerns. + Separating the user interface concerns from the data storage concerns improves the portability of the user interface across multiple platforms. + It also improves scalability by simplifying the server components. + Perhaps most significant to the Web, however, is that the separation allows the components to evolve independently, thus supporting the Internet-scale requirement of multiple organizational domains. □ <u>STATELESSNESS</u> <ul style="list-style-type: none"> + The client–server communication is constrained by no client context being stored on the server between requests. + Each request from any client contains all the information necessary to service the request, and session state is held in the client. + The session state can be transferred by the server to another service such as a database to maintain a persistent state for a period and allow authentication. + The client begins sending requests when it is ready to make the transition to a new state. + While one or more requests are outstanding, the client is considered to be <i>in transition</i>. + The representation of each application state contains links that can be used the next time the client chooses to initiate a new state-transition. □ <u>CACHEABILITY</u> <ul style="list-style-type: none"> + As on the World Wide Web, clients and intermediaries can cache responses. + Responses must therefore, implicitly or explicitly, define themselves as cacheable or not to prevent clients from getting stale or inappropriate data in response to further requests. + Well-managed caching partially or eliminates some client–server interactions, further improving scalability and performance. □ <u>LAYERED SYSTEM</u> <ul style="list-style-type: none"> + A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. + Intermediary servers can improve system scalability by enabling load balancing and by providing shared caches.

	<ul style="list-style-type: none"> + They can also enforce security policies. • <u>CODE ON DEMAND [OPTIONAL]</u> <ul style="list-style-type: none"> + Servers can temporarily extend or customize the functionality of a client by transferring executable code. + For example, compiled components such as <u>Java applets</u>, and client-side scripts such as <u>JavaScript</u>. • <u>UNIFORM INTERFACE</u> <ul style="list-style-type: none"> + The uniform interface constraint is fundamental to the design of any REST service. + It simplifies and decouples the architecture, which enables each part to evolve independently. + The four constraints for this uniform interface are: <ul style="list-style-type: none"> * Resource identification in requests <ul style="list-style-type: none"> – Individual resources are identified in requests, for example using <u>URIs</u> in Web-based REST systems. – The resources themselves are conceptually separate from the representations that are returned to the client. For example, the server could send data from its database as <u>HTML</u>, <u>XML</u> or as <u>JSON</u>—none of which are the server's internal representation. * Resource manipulation through representations <ul style="list-style-type: none"> – When a client holds a representation of a resource, including any <u>metadata</u> attached, it has enough information to modify or delete the resource. * Self-descriptive messages <ul style="list-style-type: none"> – Each message includes enough information to describe how to process the message. For example, which parser to invoke can be specified by a <u>media type</u>. * Hypermedia as the engine of application state (<u>HATEOAS</u>) <ul style="list-style-type: none"> – Having accessed an initial URI for the REST application—analogueous to a human Web user accessing the home page of a website—a REST client should then be able to use server-provided links dynamically to discover all the available actions and resources it needs. – As access proceeds, the server responds with text that includes hyperlinks to other actions that are currently available. – There is no need for the client to be hard-coded with information regarding the structure or dynamics of the REST service.
16.	Explain the role of UDDI and SOAP in description and discovery of RESTful web services?
ANS 16:	<p><i>SOAP</i></p> <ul style="list-style-type: none"> • SOAP Services can be located and invoked by client applications over a network automatically without human interaction. • SOAP facilitates interoperability among a wide range of programs and platforms, making existing applications accessible to a broader range of users. <ul style="list-style-type: none"> ○ To make the automatic locating and invocation possible SOAP web services were designed as self-describing using a message protocol called SOAP, service definition called WSDL and a way to locate other web services using UDDI. ○ SOAP web services are self-describing and self-contained, as the definition of the message format (WSDL and/or XSD) travels with the message or is accessible to the client, and the server and client requires only minimal

	<p>software an HTTP server and/or a SOAP server on the server side and on the client side no additional s/w is required.</p> <ul style="list-style-type: none"> <input type="checkbox"/> SOAP is based on rules whereas REST is based on guidelines. <input type="checkbox"/> There are no formal definitions for REST like the WSDL for SOAP. <input type="checkbox"/> There are some definition formats like WADL. <p>UDDI:</p> <ul style="list-style-type: none"> <input type="checkbox"/> To address the challenges of service registration and discovery, the Universal Description, Discovery, and Integration specification was created. <input type="checkbox"/> UDDI is a cross-industry initiative to create a registry standard for Web service description and discovery together with a registry facility that supports the publishing and discovery processes. • UDDI provides a global, platform independent, open framework making it easier to publish an enterprise's preferred means of conducting business, find trading partners, and interoperate with these trading partners over the Internet. <input type="checkbox"/> REST web services are not automatically located using an UDDI registry as in case of SOAP web services.
--	---

17.	Explain any five JAVA frameworks for building RESTful web services.
------------	--

ANS 17:

Sr. No	Name	Description
1.	Apache Axis	Apache Axis is an open-source, XML based Web service framework. It consists of a Java and a C++ implementation of the SOAP server, and various utilities and APIs for generating and deploying Web service applications. Using Apache Axis, developers can create interoperable, distributed computing applications. Axis development takes place under the auspices of the Apache Software Foundation.
2.	Apache Axis2	Apache Axis2 is a core engine for Web services. It is a complete re-design and re-write of the widely used Apache Axis SOAP stack. Implementations of Axis2 are available in Java and C.
3.	Java Web Services Development Pack	The Java Web Services Development Pack(JWS DP) is a free software development kit (SDK) for developing Web Services, Web applications and Java applications with the newest technologies for Java.

	4.	Jersey	Jersey RESTful Web Services framework is an open source framework for developing RESTful Web Services in Java. It provides support for JAX-RS APIs and serves as a JAX-RS Reference Implementation.
	5.	Web Services Interoperability Technology	Web Services Interoperability Technology (WSIT) is an open-source project started by Sun Microsystems to develop the next-generation of Web service technologies. It provides interoperability between Java Web Services and Microsoft's Windows Communication Foundation (WCF).
	6.	Web Services Invocation Framework	The Web Services Invocation Framework (WSIF) supports a simple Java API for invoking Web services, no matter how or where the services are provided. The framework allows maximum flexibility for the invocation of any Web Services Description Language (WSDL)-described service.
	7.	XML Interface for Network Services	XML Interface for Network Services (XINS) is an open source technology for definition and implementation of internet applications, which enforces a specification-oriented approach.
18.	Write a short note on JSON Array.		
ANS 18:	<ul style="list-style-type: none"> • The square brackets [] are used to declare JSON array. • JSON array are ordered list of values. • JSON array can store multiple value types. • JSON array can store string, number, boolean, object or other array inside JSON array. • In JSON array, values must be separated by comma. • Arrays in JSON are almost the same as arrays in JavaScript. • e.g. <pre> { "name" : "Admin", "age" : 36, "rights" : ["admin", "editor", "contributor"] } </pre> 		

Get value at specific index location in array

You can access the array values by using the index number:

```
x = myObj.rights[0];
```

```
//Output
```

```
admin
```

Loop through array values

You can access array values by using a for-in loop:

```
for (i in myObj.rights) {  
  x = myObj.rights[i];  
  console.log(x);  
}
```

```
//Output:
```

```
admin
```

```
editor
```

```
contributor
```

Delete array item

Use the delete keyword to delete items from an array:

```
delete myObj.rights[1];
```

Update array value at index location

Use the index number to modify an array:

```
myObj.rights[1] = "blogger";
```

Multi-dimensional array

We can store array inside JSON array, it is known as array of arrays or multidimensional array.

```
{  
  "name" : "Admin",  
  "age" : 36,  
  "rights" : [  
    {"roleName" : "admin", "roleIds" : [1,2,3] },  
    {"roleName" : "editor", "roleIds" : [4,5,6] },  
  ] {"roleName" : "contributor", "roleIds" : [7,8,9]}  
}
```

Iterate over multi-dimentional array

```
for (i in myObj.rights) {  
  
  for (j in myObj.rights[i].roleIds) {  
  
    x = myObj.rights[i].roleIds[j];  
  
    console.log(x);  
  }  
}
```

Output:

```
1  
2  
3  
4  
5  
6  
7  
8  
9
```


19.	Write a short note on JSON Object
ANS 19:	<ul style="list-style-type: none">• JSON objects are very much like javascript objects.• JSON objects are written in key/value pairs.• JSON objects are surrounded by curly braces { }.• Keys must be strings, and values must be a valid JSON data type (string, number, object, array, boolean or null).• Keys and values are separated by a colon.• Each key/value pair is separated by a comma.• For example –<pre>{ "name" : "Admin", "age" : 36, "rights" : ["admin", "editor", "contributor"] }</pre> <p>Access object values</p> <p>You can access object values in two ways :</p> <p><i>1) Using dot (.) notation</i></p> <pre>var author = { "name" : "Admin", "age" : 36, "rights" : ["admin", "editor", "contributor"] }</pre> <pre>console.log(author.name);</pre> <p>//Output</p> <p>Admin</p>

2) Using bracket ([]) notation

```
var author = {  
    "name" : "Admin",  
    "age" : 36,  
    "rights" : [ "admin", "editor", "contributor"  
]  
}
```

```
console.log( author [ "name" ] );  
console.log( author [ "age" ] );
```

//Output

Admin
36

Looping object values

You can loop through object values using for loop, just like looping through an array.

```
var author = {  
    "name" : "Admin",  
    "age" : 36,  
    "rights" : [ "admin", "editor", "contributor" ]  
}
```

```
for (x in author)  
{  
    console.log(x + " - " + (author[x]));  
}
```

//Output

name - Admin
age - 36
rights - admin,editor,contributor

Modify object values

To modify object values, use any of given two ways:

1) Using dot (.) notation

```
var author = {  
    "name" : "Admin",  
    "age" : 36,  
    "rights" : [ "admin", "editor", "contributor" ]  
}
```

```
author.name = "Lokesh";
```

```
console.log( author.name );
```

//Output

Lokesh

2) Using bracket ([]) notation

```
var author = {  
    "name" : "Admin",  
    "age" : 36,  
    "rights" : [ "admin", "editor",  
    "contributor" ]  
}
```

```
author["name"] = "Lokesh";
```

```
author["age"] = 35;
```

```
console.log( author [ "name" ] );
```

```
console.log( author [ "age" ] );
```

//Output

Lokesh

35

Delete object values

Use the delete keyword to delete properties from a JSON object:

```
delete author.name;
```

20.	How does data exchange happen using JSON? Explain.
ANS 20:	<ul style="list-style-type: none"> • If you use Java RESTful web service frameworks, such as JAX-RS, for building RESTful web APIs, the serialization and deserialization of the request and response messages will be taken care of by the framework. • The JSON structure and tools for processing JSON will definitely help you when the default offering by the framework does not meet your requirements. • The following diagram illustrates the role of the JSON marshalling and unmarshalling components in a typical Java RESTful web service implementation: <div data-bbox="248 396 941 795" data-label="Diagram"> <pre> graph LR subgraph "RESTful web service application" direction LR Request[Request] --> JSON[JSON Reader & Writer] JSON --> API[Java REST API] API --> JSON JSON --> Response[Response] end </pre> <p>The diagram illustrates the data flow within a RESTful web service application. It shows a 'Request' entering from the left, passing through a 'JSON Reader & Writer' component, then to a 'Java REST API' component. The 'Java REST API' component sends data back to the 'JSON Reader & Writer' component, which then outputs a 'Response' to the right. The entire process is contained within a box labeled 'RESTful web service application'.</p> </div> <ul style="list-style-type: none"> • The term processing, we mean reading, writing, querying, and modifying JSON data. • Two widely adopted programming models for processing JSON are as follows: <ul style="list-style-type: none"> <i>Object model:</i> <ul style="list-style-type: none"> ○ In this model, the entire JSON data is read into memory in a tree format. ○ This tree can be traversed, analyzed, or modified with the appropriate APIs. ○ As this approach loads the entire content into the memory first and then starts parsing, it ends up consuming more memory and CPU cycles. ○ However, this model gives more flexibility while manipulating the content. <i>Streaming model:</i> <ul style="list-style-type: none"> ○ The term streaming is very generic in meaning and can be used in many aspects. ○ This term means that data can be read or written in blocks. ○ This model does not read the entire JSON content into the memory to get started with parsing; rather, it reads one element at a time. ○ For each token read, the parser generates appropriate events indicating the type of token, such as the start or end of an array, or the start or end of the object and attribute values. ○ A client can process the contents by listening for appropriate events. ○ The most important point is that instead of letting the parser push the content to the client (push parser), the client can pull the information from the parser as it needs (pull parser). ○ In this model, the client is allowed to skip or stop reading contents in the middle of the process if it has finished reading the desired elements. ○ This model is also useful when you write contents to an output source in blocks. • API's are streamed in the following cases: <ul style="list-style-type: none"> ○ When the data is huge in size and it is not feasible to load the entire content into the memory for processing the content ○ When the partial processing is needed and the data model is not fully available yet.

21.	How to build RESTful web services with JAX-RS APIs?
ANS 21:	<ul style="list-style-type: none"> • <u>Specifying the dependency of the JAX-RS API</u> <ul style="list-style-type: none"> – To use JAX-RS APIs in your project, you need to add the javax.ws.rs-api JAR file to the class path. • <u>Using JAX-RS annotations to build RESTful web services</u> <ul style="list-style-type: none"> – Java annotations provide the metadata for your Java class, which can be used during compilation, during deployment, or at runtime in order to perform designated tasks. – The use of annotations allows us to create RESTful web services as easily as we develop a POJO class. – Here, we leave the interception of the HTTP requests and representation negotiations to the framework and concentrate on the business rules necessary to solve the problem at hand. • <u>Annotations for defining a RESTful resource</u> • REST resources are the fundamental elements of any RESTful web service. • A REST resource can be defined as an object that is of a specific type with the associated data and is optionally associated to other resources. • It also exposes a set of standard operations corresponding to the HTTP method types such as the HEAD, GET, POST, PUT, and DELETE methods. <ul style="list-style-type: none"> + <u>@Path</u> <ul style="list-style-type: none"> – The @javax.ws.rs.Path annotation indicates the URI path to which a resource class or a class method will respond. The value that you specify for the @Path annotation is relative to the URI of the server where the REST resource is hosted. – This annotation can be applied at both the class and the method levels. + <u>Specifying the @Path annotation on a resource class</u> <ul style="list-style-type: none"> – The following code snippet illustrates how you can make a POJO class respond to a URI path template containing the /departments path fragment: <pre>import javax.ws.rs.Path; @Path("departments") public class DepartmentService { //Rest of the code goes here }</pre> – The /department path fragment that you see in this example is relative to the base path in the URI. – The base path typically takes the following URI pattern: http://host:port/<context-root>/<application-path>. + <u>Specifying the @Path annotation on a resource class method</u> <ul style="list-style-type: none"> – The following code snippet shows how you can specify @Path on a method in a REST resource class. – Note that for an annotated method, the base URI is the effective URI of the containing class. – For instance, you will use the URI of the following form to invoke the getTotalDepartments() method defined in the DepartmentService class: /departments/count, where departments is the @Path annotation set on the class. <pre>import javax.ws.rs.GET; import javax.ws.rs.Path; import javax.ws.rs.Produces; @Path("departments") public class DepartmentService</pre>

```

    {
        @GET
        @Path("count")
        @Produces("text/plain")
        public Integer getTotalDepartments()
        {
            return findTotalRecordCount();
        }
        //Rest of the code goes here
    }

```

+ Specifying variables in the URI path template

- It is very common that a client wants to retrieve data for a specific object by passing the desired parameter to the server.
- JAX-RS allows you to do this via the URI path variables as discussed here.
- The URI path template allows you to define variables that appear as placeholders in the URI.
- These variables would be replaced at runtime with the values set by the client.
- The following example illustrates the use of the path variable to request for a specific department resource.
- The URI path template looks like /departments/{id}.
- At runtime, the client can pass an appropriate value for the id parameter to get the desired resource from the server.
- For instance, the URI path of the /departments/10 format returns the IT department details to the caller.
- The following code snippet illustrates how you can pass the department ID as a path variable for deleting a specific department record.
- The path URI looks like /departments/10.

```

import javax.ws.rs.Path;
import javax.ws.rs.DELETE;
@Path("departments")
public class DepartmentService
{
    @DELETE
    @Path("{id}")
    public void removeDepartment(@PathParam("id") short id)
    {
        removeDepartmentEntity(id);
    }
    //Other methods removed for brevity
}

```

- In the preceding code snippet, the @PathParam annotation is used for copying the value of the path variable to the method parameter.

22. Write a note on the design principle for building RESTful web services.

ANS 22: DESIGN PRINCIPLE FOR BUILDING RESTful web services

- REST (Representational State Transfer) is an architectural style, and an approach to communications that is often used in the development of Web services and nowadays with Mobile world and Internet of Things.
- Main GOAL:
 - The primary design principle when crafting your API should be to maximize developer productivity and success.

- Principle number 1 - CRUD Actions
 - Nouns are good
 - Verbs are bad
 - Keep the base URL simple and intuitive.
 - Use HTTP verbs

HTTP Verb	Path- GOOD	Path-BAD	HTTP Status	Response
GET	/photos	/getAllPhotos	200	JSON with a list of all photos
GET	/photos/:id	/getPhoto/:id	200	JSON with a photo
POST	/photos	/createPhoto	201	JSON with a new photo
PATCH/PUT	/photos/:id	/updatePhoto	200	JSON with updated photo
DELETE	/photos	/deleteAllPhotos	200	JSON with deleted photos
DELETE	/photos/:id	/deletePhoto	200	JSON with deleted photo

- There are just 2 entry points to perform CRUD Actions

RESOURCE	POST	GET	PUT	DELETE
/photos	Create a new photo	Return all photos	Update all photos	Delete all photos
/photos/1234	405 Method not allowed	Return photo	Update photo or return 400 not found	Delete photo or return 400 not found

- Principle number 2 – Associations
 - Keeps the CRUD actions just one level deep:

RESOURCE	POST	GET	PUT	DELETE
/users/:id/photos	Create a new photo for this user	Return all photos of this user	Update all photos of this user	Delete all photos

/users/:id/photos/1234	405 Method not allowed	Return this photo of this user	Update this photo of this user or return 400 not found	Delete this photo of this user or return 400 not found
------------------------	------------------------	--------------------------------	--	--

- Principle number 3 - Eager or Lazy Loading
 - In order to facilitate the developers work, always provides Eager and Lazy ways to get resources that have associations.
 - Lazy Loading – get user photos:
 - STEP 1:

RESOURCE	POST	GET	PUT	DELETE
/users	Create a new photo	Return all photos	Update all photos	Delete all photos
/users/1234	405 Method not allowed	Return photo	Update photo or return 400 not found	Delete photo or return 400 not found

■ STEP 2:

RESOURCE	POST	GET	PUT	DELETE
/users/1234/photos	Create a new photo for this user	Return all photos of this user	Update all photos of this user	Delete all photos
/users/1234/photos/1234	405 Method not allowed	Return this photo of this user	Update this photo of this user or return 400 not found	Delete this photo of this user or return 400 not found

- Eager Loading – get user photos

RESOURCE	POST	GET	PUT	DELETE
/users?include=photos	405 Method not allowed	Return all User with his photos	Update all Users and his photos	Delete all Users and their photos
/users/:id/?include=photos	405 Method not allowed	Return this User with his photos	Update this User with his photos	Delete this User and his photos

/users?include=photos&comments	405 Method not allowed	Return this User with his photos and comments	Update this User with his photos and comments	Delete this User with his photos and comments
--------------------------------	------------------------	---	---	---


- Principle number 4 – Filters
 - Sometimes it is not necessary get all information about resources, in fact is necessary just some fields or a specific characteristic and also sorting and paginate the results.
 - Fields:
 - GET /users?fields=name,email,phone
 - GET /photos?fields=title,size,state
 - Sorting:
 - GET /user?sort=age,name
 - fields with - sign will be sorting in descending order
GET /user?sort=age,-name
 - Paginating
 - GET /photos?limit=25&offset=50
- Principle number 5 – Search
 - To have a complete REST API it was necessary implement two kind of searches:
 1. Global Search
 - GET /search?resources=photos&title=fog&size=small
 - GET /search?resources=photos,images&size=small
 2. Scoped Search
 - GET /photos?title=fog&size=small
 - GET /users/1234/photos?title=fog&size=small
- Principle number 6 - Error Handling
 - In order to provide to developers the necessary information when errors happen, returns error messages in the response body
 - HTTP Status Code: 401
{ "code" : 401, "message": "Authentication Required" }
- Principle number 7 – Versioning
 - Versioning the API is a way to keep backward compatibility, to implement it, just provide the API version in the URI
 - GET /v1.0/users
- Principle number 8 – Exceptions
 - Sometimes it is necessary to create APIs that are responsible to perform actions instead of return resources, these were the unique cases where verbs are allowed
 - GET /convert?from=EUR&to=CNY&amount=100
 - GET /calculate?operation=sum&val1=8&val2=4
- Principle number 9 – Authentication

→ When resources could not be public, a authentication mechanism becomes necessary and in this case there is no question that the best approach to implement it is OAuth2.

23. What are different security practices and measures.

ANS 23:

- The WS-Security standard revolves around having the security definition included in the SOAP Header.
- The credentials in the SOAP header is managed in 2 ways.
- First, it defines a special element called UsernameToken.
- This is used to pass the username and password to the web service.
- The other way is to use a Binary Token via the BinarySecurityToken.
- This is used in situations in which encryption techniques such as Kerberos or X.509 is used.

	<ul style="list-style-type: none"> The below diagram shows the flow of how the security model works in WS Security.  <pre> graph LR A[Web Service Client] --> B[Security Token Service] B --> C[Web Service] </pre> <ul style="list-style-type: none"> Below are the steps which take place in the above workflow <ul style="list-style-type: none"> A request can be sent from the Web service client to Security Token Service. This service can be an intermediate web service which is specifically built to supply usernames/passwords or certificates to the actual SOAP web service. The security token is then passed to the Web service client. The Web service client then called the web service, but, this time, ensuring that the security token is embedded in the SOAP message. The Web service then understands the SOAP message with the authentication token and can then contact the Security Token service to see if the security token is authentic or not. The below snippet shows the format of the authentication part which is part of the WSDL document. Now based on the below snippet, the SOAP message will contain 2 additional elements, one being the Username and the other being the Password. <pre> <xs:element name="UsernameToken"> <xs:complexType> <xs:sequence> <xs:element ref="Username"/> <xs:element ref="Password" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element> </pre> When the SOAP Message is actually passed between the clients and the server, the part of the message which contains the user credentials could look like the one shown above. The wsse element name is a special element named defined for SOAP and means that it contains security based information.
24.	Write a note on Annotations for specifying request-response media types.
ANS 24:	<ul style="list-style-type: none"> Java annotations provide the metadata for your Java class, which can be used during compilation, during deployment, or at runtime in order to perform designated tasks. The use of annotations allows us to create RESTful web services easily. Annotations for specifying request-response media types : <ul style="list-style-type: none"> The Content-Type header field in HTTP describes the body's content type present in the request and response messages. The content types are represented using the standard Internet media types. A RESTful web service makes use of this header field to indicate the type of content in the request or response message body.

- JAX-RS allows you to specify which Internet media types of representations a resource can produce or consume by using the `@javax.ws.rs.Produces` and `@javax.ws.rs.Consumes` annotations, respectively.

➤ *@Produces :*

- * The `@javax.ws.rs.Produces` annotation is used for defining the Internet media type(s) that a REST resource class method can return to the client.
- * This can be defined either at the class level (which will get defaulted for all methods) or the method level. The method-level annotations override the class-level annotations.
- * The possible Internet media types that a REST API can produce are as follows:
 - `application/atom+xml`
 - `application/json`
 - `application/octet-stream`
 - `application/svg+xml`
 - `application/xhtml+xml`
 - `application/xml`
 - `text/html`
 - `text/plain`
 - `text/xml`
- * The following example uses the `@Produces` annotation at the class level in order to set the default response media type as JSON for all resource methods in this class. At runtime, the binding provider will convert the Java representation of the return value to the JSON format.

```
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("departments")
@Produces(MediaType.APPLICATION_JSON)
public class DepartmentService{
    //Class implementation goes here...
}
```

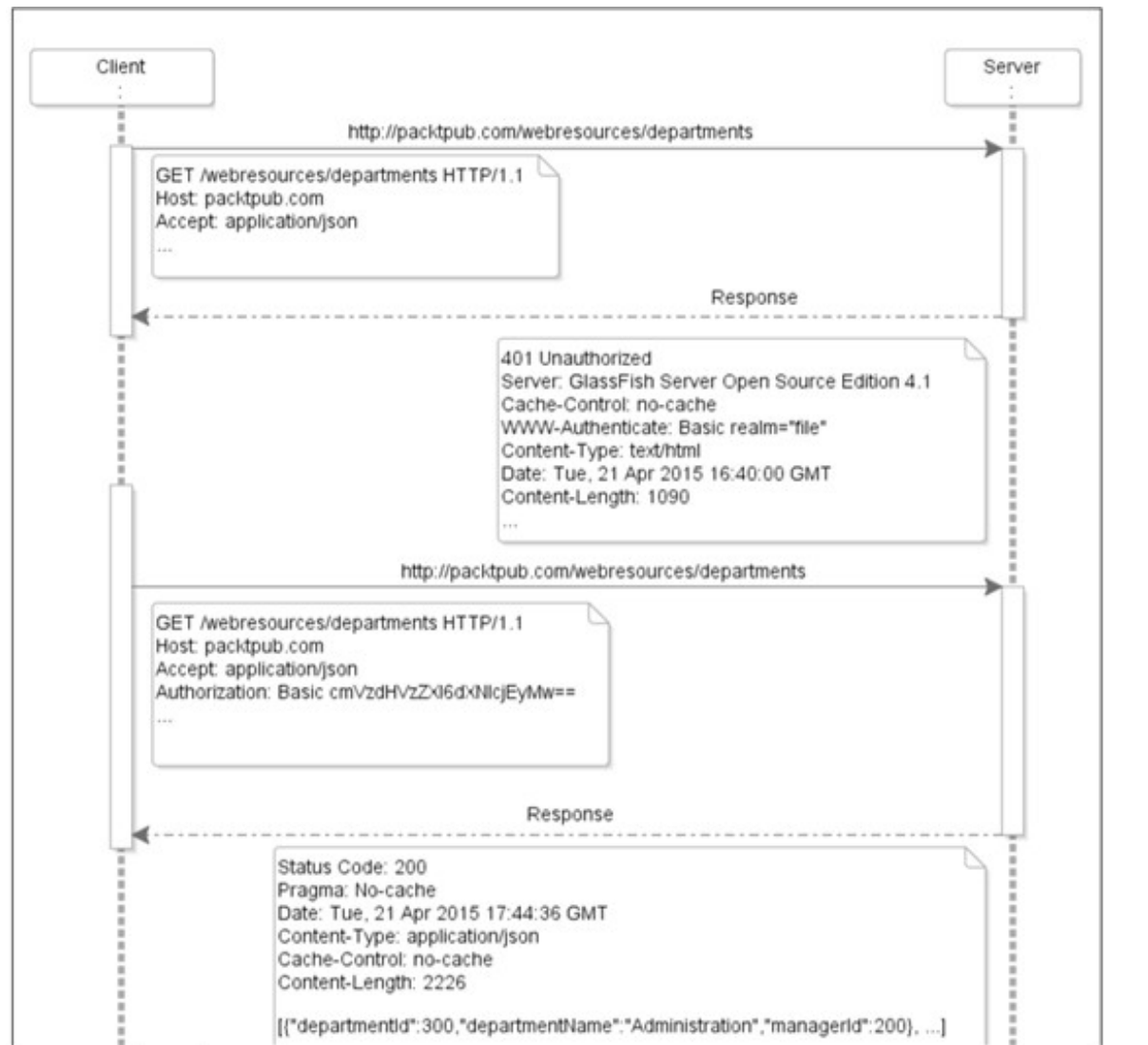
➤ *@Consumes :*

- * The `@javax.ws.rs.Consumes` annotation defines the Internet media type(s) that the resource class methods can accept.
- * You can define the `@Consumes` annotation either at the class level (which will get defaulted for all methods) or the method level.
- * The method-level annotations override the class-level annotations.
- * The possible Internet media types that a REST API can consume are as follows:
 - `application/atom+xml`
 - `application/json`
 - `application/octet-stream`
 - `application/svg+xml`
 - `application/xhtml+xml`
 - `application/xml`
 - `text/html`
 - `text/plain`
 - `text/xml`
 - `multipart/form-data`

	<ul style="list-style-type: none"> * The following example illustrates how you can use the <code>@Consumes</code> attribute to designate a method in a class to consume a payload presented in the JSON media type. * The binding provider will copy the JSON representation of an input message to the <code>Department</code> parameter of the <code>createDepartment()</code> method. <pre>import javax.ws.rs.Consumes; import javax.ws.rs.core.MediaType; import javax.ws.rs.POST; @POST @Consumes(MediaType.APPLICATION_JSON) public void createDepartment(Department entity) { //Method implementation goes here... }</pre>
25.	Write a note on Implementing partial response and Implementing partial update.
ANS 25:	<p><i>Implementing partial response</i></p> <ul style="list-style-type: none"> • Partial response refers to an optimization technique offered by the RESTful web APIs to return only the information (fields) required by the client. • In this mechanism, the client sends the required field names as the query parameters for an API to the server, and the server trims down the default response content by removing the fields that are not required by the client. • In the following example, the <code>select</code> query parameter is used for selecting fields that would be transferred over the wire: <ul style="list-style-type: none"> <code>/employees/1234?select=firstName,lastName,email</code> • The Jersey framework supports the partial response feature via <code>org.glassfish.jersey.message.filtering.SelectableEntityFilteringFeature</code>. • To enable this feature, you just need to register <code>SelectableEntityFilteringFeature</code> in the application. • The client can use the <code>select</code> query parameter to select the fields that would be transferred over the wire, as illustrated in the following example: <ul style="list-style-type: none"> – GET <code>employees/1234?select=email,department.departmentName</code> HTTP/1.1 To learn more about this feature, see the Jersey example available at https://github.com/jersey/jersey/tree/2.18/examples/entity-filteringselectable. <p><i>Implementing partial update</i></p> <ul style="list-style-type: none"> • When a client changes only one part of the resource, you can optimize the entire update process by allowing the client to send only the modified part to the server, thereby saving the bandwidth and server resources. • RFC 5789 proposes a solution for this use case via a new HTTP method called PATCH • However, the HTTP protocol allows both the client and the server to implement any new method. Leveraging this flexibility, many vendors have started supporting the HTTP PATCH method. • The PATCH method takes the following form: <ul style="list-style-type: none"> <code>PATCH /departments/10 HTTP/1.1 [Description of changes]</code> • The <code>[Description of changes]</code> section, in the preceding PATCH method, contains instructions describing how a resource currently residing on the origin server should be modified in order to reflect the changes performed by the client. • RFC 6902 defines a JSON document structure for expressing the sequence of changes performed on the resource by the client.

	<ul style="list-style-type: none"> Note that you can also have the XML structure for describing the changes performed on the XML representation of the resource. The PATCH operations supported by JSON PATCH are add, remove, replace, move, copy, and test. The following example illustrates how you can use JSON PATCH for describing changes performed by a client on a department resource. As the first step, the client retrieves the department resource from the server that looks like the following: <pre> {"departmentId":10, "departmentName":"Administration", "managerId":200, "comments":"Administrative works" } </pre> The client then performs the following modifications on the department resource: <ul style="list-style-type: none"> Modifies the manager by setting managerId to 300 Adds a new locationId=1200 value to the department resource Removes the comments attribute The JSON PATCH request body containing the preceding changes will look like the following: <pre> PATCH /departments/10 HTTP/1.1 [{ "op": "replace", "path": "/managerId", "value": 300 }, { "op": "add", "path": "/locationId", "value": 1200 }, { "op": "remove", "path": "/comments" }] </pre> The server applies the data manipulation instructions present in the incoming JSON PATCH document and modifies the original resource to reflect the changes performed by the client. After applying the modifications, the department resource on the server will look like the following: <pre> {"departmentId":10, "departmentName":"Administration", "managerId":300, locationId=1200} </pre>
26.	What is OAuth? How do you secure RESTful web services with OAuth.
ANS 26:	<ul style="list-style-type: none"> OAuth is an open standard for authorization, used by many enterprises and service providers to protect resources. OAuth solves a different security problem than what HTTP basic authentication has been used for. OAuth protocol allows client applications to access protected resources on behalf of the resource owner (typically, the application user). If we look at the history of this protocol, the OAuth Version 1.0 was published as RFC 5849 in 2010. Later, the next evolution of OAuth, Version 2.0, was published as RFC 6749 in 2012. Note that these two versions are different in their implementations and do not have many things in common. Example of OAuth. <ul style="list-style-type: none"> The typical example used to explain OAuth 1.0 is that of a service provider that stores pictures on the Web (let's call the service StorageInc), and a fictional consumer service that is a picture printing service (let's call the service PrintInc). On its own, PrintInc is a full-blown web service but it does not offer picture storage, its business is only printing pictures. For convenience, PrintInc has created a web service that lets its users download their pictures from StorageInc for printing. This is what happens when a user (the resource owner) decides to use PrintInc (the client application) to print his/her images stored in StorageInc (the service provider): <ul style="list-style-type: none"> ➤ The user creates an account in PrintInc. ➤ Let's call the user Jane, to keep things simple.

	<ul style="list-style-type: none"> ➤ PrintInc asks if Jane wants to use her pictures stored in StorageInc and presents a link to get authorization to download her pictures (the protected resources). ➤ Jane is the resource owner here. ➤ Jane decides to let PrintInc connect to StorageInc on her behalf and clicks on the authorization link. ➤ Both PrintInc and StorageInc have implemented the OAuth protocol, so StorageInc asks Jane if she wants to let PrintInc use her pictures. ➤ If she says yes, then StorageInc asks Jane to provide her username and password. ➤ Note, however, that her credentials are being used at StorageInc's site, and that PrintInc has no knowledge of her credentials. ➤ Once Jane provides her credentials, StorageInc passes PrintInc an authorization token, which is stored as a part of Jane's account on PrintInc. ➤ Now we are back at PrintInc's web application and Jane can now print any of her pictures stored in StorageInc's web service. ➤ Finally, every time Jane wants to print more pictures, all she needs to do is come back to PrintInc's website and download her pictures from StorageInc. ➤ Note that she does not need to provide her username and password again, as she has already authorized these two web services to exchange data on her behalf. <ul style="list-style-type: none"> ○ The preceding example clearly portrays the authorization flow in OAuth 1.0 protocol.
27.	Explain Basic and Digest Authentication for securing RESTful web services.
ANS 27:	<p><i>HTTP basic authentication</i></p> <ul style="list-style-type: none"> ● Basic HTTP authentication works by sending the Base64 encoded username and the password as a pair in the HTTP authorization header. ● The username and password must be sent for every HTTP request made by the client. ● A typical HTTP basic authentication transaction can be depicted with the following sequence diagram. ● In this example, the client is trying to access a protected RESTful web service endpoint (/webresources/departments) to retrieve department details: ● Diagram:



- This diagram represents a whole transaction.
 - A client begins by requesting the URI, /webresources/departments.
 - Because the resource is secured using HTTP basic authentication and the client does not provide the required authorization credentials, the server replies with a 401 HTTP response.
 - The client receives the response, scans through it, and prepares a new request with the necessary data needed to authenticate the user.
 - The new request from the client will contain the authorization header set to a Base64 encoded value of column delimited username and password string, <username>:<password>.
 - a. This time, the server will verify the credentials and replies with the requested resource.
- As we have seen, client requests can be generated from any application that can create HTTP connections, including web browsers.
- Web browsers typically cache the credentials so that users do not have to type in their username and password for every secured resource request.
- This is viewed as a deficiency of the protocol; as unauthorized access can take place with cached credentials, and there is no way for a web service to differentiate authorized requests from unauthorized ones.

	<ul style="list-style-type: none"> ● Furthermore, using basic authentication is not enough for security because usernames and passwords are only encoded using Base64 encoding, which can be easily deciphered. ● However, the intent of Base64 is not to secure the name-value pair, but to uniformly encode characters when transferred over HTTP. Because of these reasons, it is not recommended to use basic authentication over HTTP for any application accessed over the Internet. In general, we solve this potential security hole by using HTTPS (Transport Layer security) instead of HTTP. <p><i>HTTP digest authentication</i></p> <ul style="list-style-type: none"> ● <i>The HTTP digest authentication authenticates a user based on a username and a password.</i> ● <i>However, unlike with basic authentication, the password is not transmitted in clear text between the client and the server.</i> ● <i>Instead, the client sends a one way cryptographic hash of the username, password, and a few other security related fields using the MD5 message-digest hash algorithm.</i> ● <i>When the server receives the request, it regenerates the hashed value for all the fields as done by client and compares it with the one present in the request.</i> ● <i>If the hashes match, the request is treated as authenticated and valid.</i> ● <i>If the client application uses the Jersey framework implementation, then the API to invoke RESTful web services secured via the HTTP digest authentication may look like as shown in the following code snippet:</i> <pre>//Rest of the imports are removed for brevity import org.glassfish.jersey.client.authentication. HTTP_AUTHENTICATION_DIGEST_USERNAME; import org.glassfish.jersey.client.authentication. HTTP_AUTHENTICATION_DIGEST_PASSWORD; //Client code goes here final String RESOURCE_URI = "http://localhost:8080/hrapp/departments"; Client client = javax.ws.rs.client.ClientBuilder.newClient(); //Provide the username and password, and invoke method Response response = client.target(RESOURCE_URI).request() .property(HTTP_AUTHENTICATION_DIGEST_USERNAME, "<Username>") .property(HTTP_AUTHENTICATION_DIGEST_PASSWORD, "<Password>") .get();</pre>
28.	Write a note on Implementing search and sort operations.
ANS 28:	<p><i>Implementing search and sort operations</i></p> <ul style="list-style-type: none"> ● Allowing a client to perform the search and sort operations on a resource collection is very essential to improve the market adoption of APIs. ● As there is no existing standard for passing sort criteria or search conditions, various API vendors follow different patterns. ● A very common approach is to pass the search and sort criteria as the query parameters to the server. ● The following example illustrates how you can pass the search criteria as the query parameters to the server: <ul style="list-style-type: none"> ○ <code>/employees?departmentName=hr&salary>500000</code> ○ The query parameters present in the preceding resource request URI can be used by the RESTful API implementation to find out the employee resources belonging to the HR department whose annual salary is greater than 500000.

	<ul style="list-style-type: none">○ Similarly, to read the collection of resources in a sorted order, you can pass the sort criteria as the query parameter to the API.○ The following example uses the sort keyword as the query parameter to indicate the beginning of fields in the URI for sorting, followed by the asc or desc keyword, indicating the sort order: /employees?sort=firstName:asc,lastName:asc <ul style="list-style-type: none">● If you have complex search conditions that cannot be easily represented as the request parameter in the URI, you can consider moving the search conditions into the request body and use the POST method for issuing the search.● Example: POST employees/searches HTTP/1.1 Host: packtpub.com Accept: application/json Content-Type: application/json { "criteria": [{ "firstName": "A"; "operator": "startswith" }], "sort": [{ "firstName": "asc", "lastName": "asc" }] }
--	--