1. SOAP vs Restful?

REST and SOAP are 2 different approaches to online data transmission. Specifically, both define how to build application programming interfaces (APIs), which allow data to be communicated between web applications. Representational state transfer (REST) is a set of architectural principles. Simple object access protocol (SOAP) is an official protocol maintained by the World Wide Web Consortium (W3C). The main difference is that SOAP is a protocol while REST is not. Typically, an API will adhere to either REST or SOAP, depending on the use case and preferences of the developer.

When a request for data is sent to a REST API, it's usually done through hypertext transfer protocol (commonly referred to as HTTP). Once a request is received, APIs designed for REST (called RESTful APIs or RESTful web services) can return messages in a variety of formats: HTML, XML, plain text, and JSON. JSON (JavaScript object notation) is favored as a message format because it can be read by any programming language (despite the name), is human- and machine-readable, and is lightweight. In this way, RESTful APIs are more flexible and can be easier to set up.

2. Difference between acceptance test and functional test?

The functional test confirms the software performs a function within the boundaries of how you've solved the problem. This is an integral part of developing software, comparable to the testing that is done on mass produced product before it leaves the factory. A functional test verifies that the product works as you (the developer) think it does.

Acceptance tests verify the product solves the problem it was made to solve. This can best be done by the user (customer), for instance performing his/her tasks that the software assists with. If the software passes this real-world test, it's accepted to replace the previous solution. This acceptance test can sometimes only be done properly in production, especially if you have anonymous customers (e.g., a website). Thus, a new feature will only be accepted after days or weeks of use.

3. What is Mocking?

Mocking is primarily used in unit testing. An object under test may have dependencies on other (complex) objects. To isolate the behavior of the object you want to replace the other objects by mocks that simulate the behavior of the real objects. This is useful if the real objects are impractical to incorporate into the unit test. In short, mocking is creating objects that simulate the behavior of real objects.

4. What is a reasonable code coverage % for unit test (and why)?

Code coverage of 70-80% is a reasonable goal for system test of most projects with most coverage metrics. Use a higher goal for projects specifically organized for high testability or that have high failure costs. Minimum code coverage for unit testing can be 10-20% higher than for system testing. Code coverage provides significant benefits to the developer workflow. It is not a perfect measure of test quality, but it does offer a reasonable, objective, industry standard metric with actionable data. It does not require significant human interaction, it applies universally to all products, and there are ample tools available in the industry for most languages.

5. HTTP/POST vs HTTP/PUT?

POST is used to send data to a server to create/update a resource. The data sent to the server with POST is stored in the request body of the HTTP request. POST is one of the most common HTTP methods. POST requests are never cached, POST requests do not remain in the browser history, POST requests cannot be bookmarked, POST requests have no restrictions on data length. PUT is used to send data to a server to create/update a resource. The difference between POST and PUT is that PUT requests are idempotent. That is, calling the same PUT request multiple times will always produce the same result. In contrast, calling a POST request repeatedly have side effects of creating the same resource multiple times.

6. What are the Safe and Unsafe methods of HTTP?

An HTTP method is safe if it doesn't alter the state of the server. In other words, a method is safe if it leads to a read-only operation. Several common HTTP methods are safe: GET, HEAD, or OPTIONS. All safe methods are also idempotent, but not all idempotent methods are safe. For example, PUT and DELETE are both idempotent but unsafe. Even if safe methods have a read-only semantic, servers can alter their state: e.g., they can log or keep statistics.

7. How does HTTP Basic Authentication work?

HTTP basic authentication is a simple challenge and response mechanism with which a server can request authentication information (a user ID and password) from a client. The client passes the authentication information to the server in an Authorization header. The authentication information is in base-64 encoding. If a client makes a request for which the server expects authentication information, the server sends an HTTP response with a 401-status code, a reason phrase indicating an authentication error, and a WWW-Authenticate header. Most web clients handle this response by requesting a user ID and password from the end user.

8. Define RestTemplate in Spring?

RestTemplate is a synchronous client to perform HTTP requests. It uses a simple, template method API over underlying HTTP client libraries such as the JDK HttpURLConnection, Apache HttpComponents, and others. Since Spring 5.0, a new client WebClient is available that can be use do create both synchronous and asynchronous requests. In the future releases, RestTemplate will be deprecated in favour of WebClient.

9. What is idempotent and which HTTP methods are idempotent?

An HTTP method is idempotent if an identical request can be made once or several times in a row with the same effect while leaving the server in the same state. In other words, an idempotent method should not have any side-effects (except for keeping statistics). Implemented correctly, the GET, HEAD, PUT, and DELETE methods are idempotent, but not the POST method. All safe methods are also idempotent. To be idempotent, only the actual back-end state of the server is considered, the status code returned by each request may differ: the first call of a DELETE will likely return a 200, while successive ones will likely return a 404. Another implication of DELETE being idempotent is that developers should not implement RESTful APIs with a delete last entry functionality using the DELETE method.

10. What is DNS Spoofing? How to prevent?

The “spoofing” term in the attack means that the threat actor is using a malicious site that resembles the official website a user knows. Since DNS is a critical part of Internet communication, poisoning entries give an attacker the perfect phishing scenario to collect sensitive data. The threat actor can collect passwords, banking information, credit card numbers, contact information, and geographic data. Any user that accesses the internet from public Wi-Fi is vulnerable to DNS spoofing. To protect from DNS spoofing, internet providers can use DNSSEC (DNS security). When a domain owner sets up DNS entries, DNSSEC adds a cryptographic signature to the entries required by resolvers before they accept DNS lookups as authentic.

11. What is content negotiation?

In HTTP, content negotiation is the mechanism that is used for serving different representations of a resource to the same URI to help the user agent specify which representation is best suited for the user (for example, which document language, which image format, or which content encoding). A specific document is called a resource. When a client wants to obtain a resource, the client requests it via a URL. The server uses this URL to choose one of the variants available–each variant is called a representation–and returns a specific representation to the client. The overall resource, as well as each of the representations, has a specific URL. Content negotiation determines how a specific representation is chosen when the resource is called.

12. What is statelessness in RESTful Web Services?

Statelessness means that every HTTP request happens in complete isolation. When the client makes an HTTP request, it includes all information necessary for the server to fulfill the request. The server never relies on information from previous requests from the client. If any such information is important then the client will send that as part of the current request.

13. What is CSRF attack? How to prevent?

CSRF, (Cross-site request forgery) is a simple yet invasive malicious exploit of a website. It involves a cyberattacker adding a button or link to a suspicious website that makes a request to another site you’re authenticated on. For example, a user is logged into their online banking platform which has poor security, and by clicking a “download” button on an untrusted site, it maliciously initiates a money transfer request on their behalf through their current online banking session. Compromised sites can reveal information or perform actions as an authorized user without your explicit permission. A key design principle that protects you from CSRF attacks is using GET requests for only view or read-only actions. These types of requests should not transform data and must only display recorded data. This limits the number of requests that are vulnerable to CSRF attacks.

14. What are the core components of the HTTP request and HTTP response?

The communication between our client and the API is achieved using HTTP Request which is followed by a Response to the client. Both the Requests and Response follows a certain syntax and structure to ease the communication process. Whenever our client application wants to communicate to the server, it sends out a message to the server using HTTP Protocols, which is also termed as the HTTP Request. Based on that message, the server performs certain operations as demanded by the message and then replies to the client through a message, also known as HTTP Response. There are 5 major components for HTTP Request. Verb − Indicate HTTP methods such as GET, POST, DELETE, PUT etc. URI − Uniform Resource Identifier (URI) to identify the resource on server. HTTP Version − Indicate HTTP version, for example HTTP v1.1. Request Header − Contains metadata for the HTTP Request message as key-value pairs. For example, client (or browser) type, format supported by client, format of message body, cache settings etc. Request Body − Message content or Resource representation HTTP Response broadly has 3 main components; Status Line, Headers, Body (Optional).