SOA (Service Oriented Architecture) Principles

A service-oriented architecture (SOA) is an architectural pattern in computer software design in which application components provide services to other components via a communications protocol, typically over a network. The principles of service-orientation are independent of any product, vendor or technology.

SOA just makes it easier for software components over various networks to work with each other.

Web services which are built as per the SOA architecture tend to make web service more independent. The web services themselves can exchange data with each other and because of the underlying principles on which they are created, they don't need any sort of human interaction and also don't need any code modifications. It ensures that the web services on a network can interact with each other seamlessly.

SOA is based on some key principles which are mentioned below

1. **Standardized Service Contract** - Services adhere to a service description. A service must have some sort of description which describes what the service is about. This makes it easier for client applications to understand what the service does.

2. **Loose Coupling** – Less dependency on each other. This is one of the main characteristics of web services which just states that there should be as less dependency as possible between the web services and the client invoking the web service. So if the service functionality changes at any point in time, it should not break the client application or stop it from working.

3. **Service Abstraction** - Services hide the logic they encapsulate from the outside world. The service should not expose how it executes its functionality; it should just tell the client application on what it does and not on how it does it.

4. **Service Reusability** - Logic is divided into services with the intent of maximizing reuse. In any development company re-usability is a big topic because obviously one wouldn't want to spend time and effort building the same code again and again across multiple applications which require them. Hence, once the code for a web service is written it should have the ability work with various application types.

5. **Service Autonomy** - Services should have control over the logic they encapsulate. The service knows everything on what functionality it offers and hence should also have complete control over the code it contains.

6. **Service Statelessness** - Ideally, services should be stateless. This means that services should not withhold information from one state to the other. This would need to be done from either the client application. An example can be an order placed on a shopping site. Now you can have a web service which gives you the price of a particular item. But if the items are added to a shopping cart and the web page navigates to the page where you do the payment, the responsibility of the price of the item to be transferred to the payment page should not be done by the web service. Instead, it needs to be done by the web application.

7. **Service Discoverability** - Services can be discovered (usually in a service registry). We have already seen this in the concept of the UDDI, which performs a registry which can hold information about the web service.

8. **Service Composability** - Services break big problems into little problems. One should never embed all functionality of an application into one single service but instead, break the service down into modules each with a separate business functionality.

9. **Service Interoperability** - Services should use standards that allow diverse subscribers to use the service. In web services, standards as XML and communication over HTTP is used to ensure it conforms to this principle.

Prev (/api-vs-web-service-difference.html)

Report a Bug

Next (/json-tutorial-example.html)

YOU MIGHT LIKE:

WEB SERVICE

(/top-6-api-testing-tool.html) (/top-6-api-testing-tool.html)

20 Best API Testing Tools in 2021: REST & SOAP Web Services

(/top-6-api-testing-tool.html)

WEB SERVICE

(/wsdl-web-services-description-language.html) (/wsdl-web-services-description-language.html)

WSDL Tutorial: Web Services Description Language with Example

(/wsdl-web-services-description-language.html)

WEB SERVICE

(/web-service-architecture.html) (/web-service-architecture.html)

What are Web Services? Architecture, Types, Example

(/web-service-architecture.html)

WEB SERVICE

(/security-web-services.html) (/security-web-services.html)

Web Service(WS) Security Tutorial with SOAP Example

(/security-web-services.html)

WEB SERVICE

(/api-vs-web-service-difference.html) (/api-vs-web-service-difference.html)

API vs Web Service: What's the Difference?

(/api-vs-web-service-difference.html)

WEB SERVICE

(/soap-simple-object-access-protocol.html) (/soap-simple-object-access-protocol.html)

SOAP Web Services Tutorial: What is SOAP Protocol? EXAMPLE

(/soap-simple-object-access-protocol.html)

Web Services Tutorial

Security in Web Services (/security-web-services.html)

SOA Principles (/soa-principles.html)

(https://www.facebook.com/guru99com/)
(https://twitter.com/guru99com)
(https://www.linkedin.com/company/guru99/)
(https://www.youtube.com/channel/UC19i1XD6k88KqHlET8atqFQ)
(https://forms.aweber.com/form/46/724807646.htm)

About

About Us (/about-us.html)
Advertise with Us (/advertise-us.html)
Write For Us (/become-an-instructor.html)
Contact Us (/contact-us.html)

Career Suggestion

SAP Career Suggestion Tool (/best-sap-module.html)
Software Testing as a Career (/software-testing-career-complete-guide.html)

Interesting

eBook (/ebook-pdf.html)
Blog (/blog/)
Quiz (/tests.html)
SAP eBook (/sap-ebook-pdf.html)

Execute online

Execute Java Online (/try-java-editor.html)
Execute Javascript (/execute-javascript-online.html)
Execute HTML (/execute-html-online.html)
Execute Python (/execute-python-online.html)