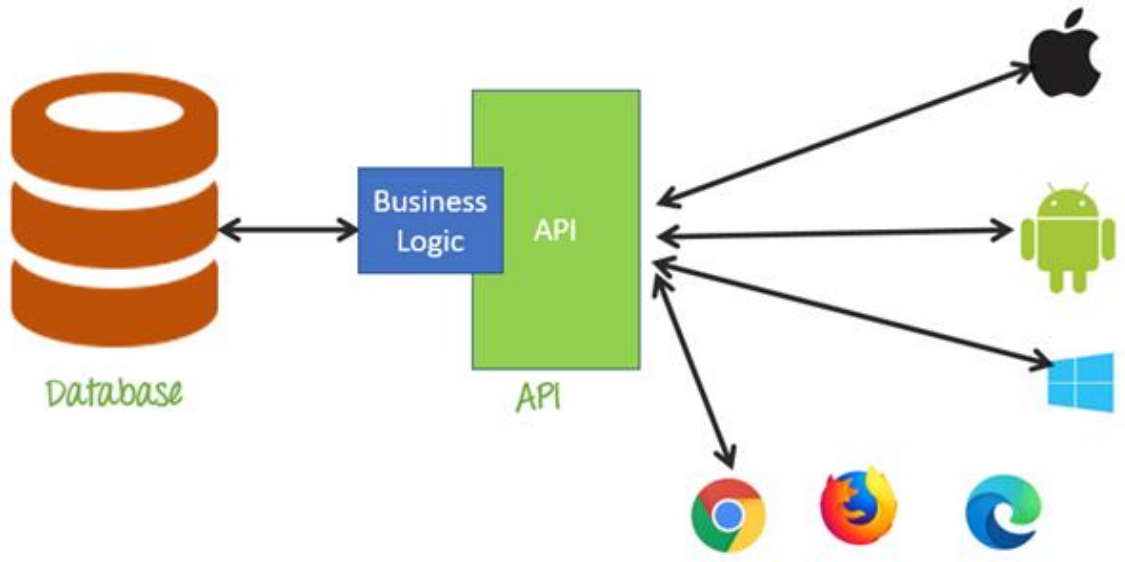
Sachin Sirohi

API E-BOOK

Chapter-1

What is an API?

Application Programming Interface (API) is a software interface that allows two applications to interact with each other without any user intervention. API is a collection of software functions and procedures. In simple terms, API means a software code that can be accessed or executed. API is defined as a code that helps two different software's to communicate and exchange data with each other.
It offers products or services to communicate with other products and services without having to know how they're implemented.



Application Programming Interface

In this API tutorial, you will learn:

How does API work?

To understand the functionality of the API, let see the following example:

API Example :

After understanding the concept, let us take some more technical examples.

For example, you go to the movie site, you enter your movie, name, and credit card information, and behold, you print out tickets.

They are collaborating with other applications. This integration is called “seamless,” as you never have a clue when a software role is passed from one application to another.

Why would we need an API?

Here, are some reason for using API:

- Application Programming Interface acronym API helps two different software’s to communicate and exchange data with each other.
- It helps you to embed content from any site or application more efficiently.
- APIs can access app components. The delivery of services and information is more flexible.
- Content generated can be published automatically.
- It allows the user or a company to customize the content and services which they use the most.
- Software needs to change over time, and APIs help to anticipate changes.

Features of API

Here are some important features of API:

- It offers a valuable service (data, function, audience,.).
- It helps you to plan a business model.
- Simple, flexible, quickly adopted.
- Managed and measured.
- Offers great developer support.

Types of API

There are mainly four main types of APIs:

- **Open APIs:** These types of APIs are publicly available to use like OAuth APIs from Google. It has also not given any restriction to use them. So, they are also known as Public APIs.
- **Partner APIs:** Specific rights or licenses to access this type of API because they are not available to the public.
- **Internal APIs:** Internal or private. These APIs are developed by companies to use in their internal systems. It helps you to enhance the productivity of your teams.
- **Composite APIs:** This type of API combines different data and service APIs.

Communication level of APIs:

Here, are some communication level of APIS:

High-Level APIs:

High-level APIs are those that we can generally use in REST form, where programmers have a high level of abstraction. These API's mostly concerned about performing a limited functionality.

Low-Level APIs:

This kind of APIs has a lower level of abstraction, which means they are more detailed. It allows the programmer to manipulate functions within an application module or hardware at a granular level.

What is Web APIs?

A Web API is an application programming interface which is use either for web server or a web browser.

Two types of Web APIs are
1) Server-side API
2) Client-side API

1.Server-side:

Server-side web API is a programmatic interface that consist of one or more publicly exposed endpoints to a defined request–response message system. It is typically expressed in JSON or XML

2.Client-side:

A client-side web API is a programmatic interface helps to extend functionality within a web browser or other HTTP client.

Examples of web API:

- Google Maps API's allow developers to embed Google Maps on webpages by using a JavaScript or Flash interface.
- YouTube API allows developers to integrate YouTube videos and functionality into websites or applications.
- Twitter offers two APIs. The REST API helps developers to access Twitter data, and the search API provides methods for developers to interact with Twitter Search.
- Amazon's API gives developers access to Amazon's product selection.

API Testing tools

Here are some popular API tools:

1) Postman

Postman is a plugin in Google Chrome, and it can be used for testing API services. It is a powerful HTTP client to check web services. For manual or exploratory testing, Postman is a good choice for testing API.

Features:

- With Postman, almost all modern web API data can be extracted
- Helps you to write Boolean tests within Postman Interface
- You can create a collection of REST calls and save each call as part of a collection for execution in future
- For transmitting and receiving REST information, Postman is more reliable.

Download link: https://www.postman.com/

Types of API protocols

As the use of web APIs has increased, certain protocols have been developed to provide users with a set of defined rules that specifies the accepted data types and commands. In effect, these API protocols facilitate standardized information exchange:

- **SOAP** (Simple Object Access Protocol) is an API protocol built with XML, enabling users to send and receive data through SMTP and HTTP. With SOAP APIs, it is easier to share information between apps or software components that are running in different environments or written in different languages.
- **XML-RPC** is a protocol that relies on a specific format of XML to transfer data, whereas SOAP uses a proprietary XML format. XML-RPC is older than SOAP, but much simpler, and relatively lightweight in that it uses minimum bandwidth.
- **JSON-RPC** is a protocol similar to XML-RPC, as they are both remote procedure calls (RPCs), but this one uses JSON instead of XML format to transfer data. Both protocols are simple. While calls may contain multiple parameters, they only expect one result.
- **REST** (Representational State Transfer) is a set of web API architecture principles, which means there are no official standards (unlike those with a protocol). To be a REST API (also known as a RESTful API), the interface must adhere to certain architectural constraints. It's possible to build RESTful APIs with SOAP protocols, but the two standards are usually viewed as competing specifications.

Advantages of APIs –

- **Efficiency:** API produces efficient, quicker and more reliable results than the outputs produced by human beings in an organization.
- **Flexible delivery of services:** API provides fast and flexible delivery of services according to developers requirements.
- **Integration:** The best feature of API is that it allows movement of data between various sites and thus enhances integrated user experience.
- **Automation:** As API makes use of robotic computers rather than humans, it produces better and automated results.
- **New functionality**: While using API the developers find new tools and functionality for API exchanges.

Disadvantages of APIs –

- **Cost:** Developing and implementing API is costly at times and requires high maintenance and support from developers.
- **Security issues:** Using API adds another layer of surface which is then prone to attacks, and hence the security risk problem is common in API’s.

Chapter-2

What is SOAP?

SOAP is a protocol which was designed before REST and came into the picture. The main idea behind designing SOAP was to ensure that programs built on different platforms and programming languages could exchange data in an easy manner. SOAP stands for Simple Object Access Protocol.

What is REST?

REST was designed specifically for working with components such as media components, files, or even objects on a particular hardware device. Any web service that is defined on the principles of REST can be called a RestFul web service. A Restful service would use the normal HTTP verbs of GET, POST, PUT and DELETE for working with the required components. REST stands for Representational State Transfer.

KEY DIFFERENCE

- SOAP stands for Simple Object Access Protocol whereas REST stands for Representational State Transfer.
- SOAP is a protocol whereas REST is an architectural pattern.
- SOAP uses service interfaces to expose its functionality to client applications while REST uses Uniform Service locators to access to the components on the hardware device.
- SOAP needs more bandwidth for its usage whereas REST doesn't need much bandwidth.
- Comparing SOAP vs REST API, SOAP only works with XML formats whereas REST work with plain text, XML, HTML and JSON.
- SOAP cannot make use of REST whereas REST can make use of SOAP.

Difference Between SOAP and REST

Each technique has its own advantages and disadvantages. Hence, it's always good to understand in which situations each design should be used. This REST and SOAP API difference tutorial will go into some of the key difference between REST and SOAP API as well as what challenges you might encounter while using them.

Below is the main difference between SOAP and REST API

SOAP	REST
• SOAP stands for Simple Object Access Protocol	• REST stands for Representational State Transfer
• SOAP is a protocol. SOAP was designed with a specification. It includes a WSDL file which has the required information on what the web service does in addition to the location of the web service.	• REST is an Architectural style in which a web service can only be treated as a RESTful service if it follows the constraints of being 1. Client Server 2. Stateless 3. Cacheable 4. Layered System 5. Uniform Interface
• SOAP cannot make use of REST since SOAP is a protocol and REST is an architectural pattern.	• REST can make use of SOAP as the underlying protocol for web services, because in the end it is just an architectural pattern.
• SOAP uses service interfaces to expose its functionality to client applications. In SOAP, the WSDL file provides the client with the necessary information which can be used to understand what services the web service can offer.	• REST use Uniform Service locators to access to the components on the hardware device. For example, if there is an object which represents the data of an employee hosted on a URL as http://demo.guru99 , the below are some of URI that can exist to access them. http://demo.guru99.com/Employee http://demo.guru99.com/Employee/1
• SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot. <?xml version="1.0"?> <SOAP-ENV:Envelope xmlns:SOAP-ENV ="http://www.w3.org/2001/12/soap-envelope" SOAP-ENV:encodingStyle =" http://www.w3.org/2001/12/soap-encoding"> <soap:Body>	• REST does not need much bandwidth when requests are sent to the server. REST messages mostly just consist of JSON messages. Below is an example of a JSON message passed to a web server. You can see that the size of the message is comparatively smaller to SOAP. {"city":"Mumbai","state":"Maharastra"}

<Demo.guru99WebService xmlns="http://tempuri.org/"> <EmployeeID>int</EmployeeID> </Demo.guru99WebService> </soap:Body> </SOAP-ENV:Envelope>	
• SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format.	• REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON.

When to use REST?

One of the most highly debatable topics is when REST should be used or when to use SOAP while designing web services. Below are some of the key factors that determine when REST and SOAP API technology should be used for web services **REST services should be used in the following instances**

- **Limited resources and bandwidth** – Since SOAP messages are heavier in content and consume a far greater bandwidth, REST should be used in instances where network bandwidth is a constraint.
- **Statelessness** – If there is no need to maintain a state of information from one request to another then REST should be used. If you need a proper information flow wherein some information from one request needs to flow into another then SOAP is more suited for that purpose. We can take the example of any online purchasing site. These sites normally need the user first to add items which need to be purchased to a cart. All of the cart items are then transferred to the payment page in order to complete the purchase. This is an example of an application which needs the state feature. The state of the cart items needs to be transferred to the payment page for further processing.
- **Caching** – If there is a need to cache a lot of requests then REST is the perfect solution. At times, clients could request for the same resource multiple times. This can increase the number of requests which are sent to the server. By implementing a cache, the most frequent queries results can be stored in an intermediate location. So whenever the client requests for a resource, it will first check the cache. If the resources exist then, it will not proceed to the server. So caching can help in minimizing the amount of trips which are made to the web server.
- **Ease of coding** – Coding REST Services and subsequent implementation is far easier than SOAP. So if a quick win solution is required for web services, then REST is the way to go.

Next in this SOAP and REST difference tutorial, we will learn when to use SOAP API.

When to use SOAP?

SOAP should be used in the following instances

1. **Asynchronous processing and subsequent invocation** – if there is a requirement that the client needs a guaranteed level of reliability and security then the new SOAP standard of SOAP 1.2 provides a lot of additional features, especially when it comes to security.
2. **A Formal means of communication** – if both the client and server have an agreement on the exchange format then SOAP 1.2 gives the rigid specifications for this type of interaction. An example is an online purchasing site in which users add items to a cart before the payment is made. Let's assume we have a web service that does the final payment. There can be a firm agreement that the web service will only accept the cart item name, unit price, and quantity. If such a scenario exists then, it's always better to use the SOAP protocol.
3. **Stateful operations –** if the application has a requirement that state needs to be maintained from one request to another, then the SOAP 1.2 standard provides the WS* structure to support such requirements.

Next in this REST vs SOAP API difference, we will learn about challenges with SOAP API.

Challenges in SOAP API

API is known as the **Application Programming Interface** and is offered by both the client and the server. In the client world, this is offered by the browser whereas in the server world it's what is provided by the web service which can either be SOAP or REST.

Challenges with the SOAP API

1. WSDL file – One of the key challenges of the SOAP API is the WSDL document itself. The WSDL document is what tells the client of all the operations that can be performed by the web service. The WSDL document will contain all information such as the data types being used in the SOAP messages and what all operations are available via the web service. The below code snippet is just part of a sample WSDL file.

```
<?xml version="1.0"?>
<definitions name="Tutorial"
        targetNamespace=http://demo.guru99.com/Tutorial.wsdl
        xmlns:tns=http://demo.guru99.com/Tutorial.wsdl
        xmlns:xsd1=http://demo.guru99.com/Tutorial.xsd
        xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
        xmlns="http://schemas.xmlsoap.org/wsdl/">

        <types>
                <schema targetNamespace=http://Demo.guru99.com/Tutorial.xsd
                xmlns="http://www.w3.org/2000/10/XMLSchema">

        <element name="TutorialNameRequest">
                        <complexType>
                                <all>
                                        <element name="TutorialName"
type="string"/>
                                </all>
                        </complexType>
                </element>
        <element name="TutorialIDRequest">
                <complexType>
                        <all>
                                <element name="TutorialID" type="number"/>
                        </all>
                </complexType>
        </element>
        </schema>
</types>
```

As per the above WSDL file, we have an element called “TutorialName” which is of the type String which is part of the element TutorialNameRequest.

Now, suppose if the WSDL file were to change as per the business requirements and the TutorialName has to become TutorialDescription. This would mean that all the clients who are currently connecting to this web service would then need to make this corresponding change in their code to accommodate the change in the WSDL file.

This shows the biggest challenge of the WSDL file which is the tight contract between the client and the server and that one change could cause a large impact, on the whole, client applications.

2. Document size – The other key challenge is the size of the SOAP messages which get transferred from the client to the server. Because of the large messages, using SOAP in places where bandwidth is a constraint can be a big issue.

Next in this RESTful vs SOAP difference, we will learn about challenges with REST API.

Challenges in REST API

1. **Lack of Security** – REST does not impose any sort of security like SOAP. This is why REST is very appropriate for public available URL's, but when it comes down to confidential data being passed between the client and the server, REST is the worst mechanism to be used for web services.

Lack of state – Most web applications require a stateful mechanism. For example, if you had a purchasing site which had the mechanism of having a shopping cart, it is required to know the number of items in the shopping cart before the actual purchase is made. Unfortunately, the burden of maintaining this state lies with the client, which just makes the client application heavier and difficult to maintain

Chapter-3

REST API (Introduction)

REpresentational **S**tate **T**ransfer (REST) is an architectural style that defines a set of constraints to be used for creating web services. **REST API** is a way of accessing web services in a simple and flexible way without having any processing.

REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST uses less bandwidth, simple and flexible making it more suitable for internet usage. It's used to fetch or give some information from a web service. All communication done via REST API uses only HTTP request.

Working

A request is sent from client to server in the form of web URL as HTTP GET or POST or PUT or DELETE request. After that, a response comes back from server in the form of a resource which can be anything like HTML, XML, Image or JSON. But now JSON is the most popular format being used in Web Services.

In **HTTP** there are five methods which are commonly used in a REST based Architecture i.e., POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations respectively. There are other methods which are less frequently used like OPTIONS and HEAD.

1. **GET:** The HTTP GET method is used to **read** (or retrieve) a representation of a resource. In the safe path, GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).

2. **POST:** The POST verb is most-often utilized to **create** new resources. In particular, it's used to create subordinate resources. That is, subordinate to some other (e.g. parent) resource. On successful creation, return HTTP status 201, returning a Location header with a link to the newly-created resource with the 201 HTTP status.
   NOTE: POST is neither safe nor idempotent.

3. **PUT:** It is used for **updating** the capabilities. However, PUT can also be used to **create** a resource in the case where the resource ID is chosen by the client instead of by the server. In other words, if the PUT is to a URI that contains the value of a non-existent resource ID. On successful update, return 200 (or 204 if not returning any content in the body) from a PUT. If using PUT for create, return HTTP status 201 on successful creation. PUT is not safe operation but it's idempotent.

4. **PATCH:** It is used for **modify** capabilities. The PATCH request only needs to contain the changes to the resource, not the complete resource. This resembles PUT, but the body contains a set of instructions describing how a resource currently residing on the server should be modified to produce a new version. This means that the PATCH body should not just be a modified part of the resource, but in some kind of patch language like JSON Patch or XML Patch. PATCH is neither safe nor idempotent.

5. **DELETE:** It is used to **delete** a resource identified by a URI. On successful deletion, return HTTP status 200 (OK) along with a response body.

Idempotence: An idempotent HTTP method is a HTTP method that can be called many times without different outcomes. It would not matter if the method is called only once, or ten times over. The result should be the same. Again, this only applies to the result, not the resource itself. Example,

REST API Architectural Constraints

REST stands for REpresentational State Transfer and API stands for Application Program Interface. REST is a software architectural style that defines the set of rules to be used for creating web services. Web services which follow the REST architectural style are known as RESTful web services. It allows requesting systems to access and manipulate web resources by using a uniform and predefined set of rules. Interaction in REST based systems happen through Internet's Hypertext Transfer Protocol (HTTP).

A Restful system consists of a:

- client who requests for the resources.
- server who has the resources.

It is important to create REST API according to industry standards which results in ease of development and increase client adoption.

Architectural Constraints of RESTful API: There are six architectural constraints which makes any web service are listed below:

- Uniform Interface
- Stateless

- Cacheable
- Client-Server
- Layered System
- Code on Demand

The only optional constraint of REST architecture is code on demand. If a service violates any other constraint, it cannot strictly be referred to as RESTful.

Uniform Interface: It is a key constraint that differentiate between a REST API and Non-REST API. It suggests that there should be an uniform way of interacting with a given server irrespective of device or type of application (website, mobile app).

There are four guidelines principle of Uniform Interface are:

- **Resource-Based:** Individual resources are identified in requests. For example: API/users.
- **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.
- **Self-descriptive Messages:** Each message includes enough information to describe how to process the message so that server can easily analyses the request.
- **Hypermedia as the Engine of Application State (HATEOAS):** It need to include links for each response so that client can discover other resources easily.

Stateless: It means that the necessary state to handle the request is contained within the request itself and server would not store anything related to the session. In REST, the client must include all information for the server to fulfill the request whether as a part of query params, headers or URI. Statelessness enables greater availability since the server does not have to maintain, update or communicate that session state. There is a drawback when the client need to send too much data to the server so it reduces the scope of network optimization and requires more bandwidth.

Cacheable: Every response should include whether the response is cacheable or not and for how much duration responses can be cached at the client side. 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. 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.

Client-Server: REST application should have a client-server architecture. A Client is someone who is requesting resources and are not concerned with data storage, which remains internal to each server, and server is someone who holds the resources and are not concerned with the user interface or user state. They can evolve independently. Client doesn't need to know anything about business logic and server doesn't need to know anything about frontend UI.

Layered system: An application architecture needs to be composed of multiple layers. Each layer doesn't know any thing about any layer other than that of immediate layer and there can be lot of intermediate servers between client and the end server. Intermediary servers may improve system availability by enabling load-balancing and by providing shared caches.
Code on demand: It is an optional feature. According to this, servers can also provide executable code to the client. The examples of code on demand may include the compiled components such as Java applets and client-side scripts such as JavaScript.
Rules of REST API: There are certain rules which should be kept in mind while creating REST API endpoints.

- REST is based on the resource or noun instead of action or verb based. It means that a URI of a REST API should always end with a noun. Example: /api/users is a good example, but /api?type=users is a bad example of creating a REST API.
- HTTP verbs are used to identify the action. Some of the HTTP verbs are – GET, PUT, POST, DELETE, UPDATE, PATCH.
- A web application should be organized into resources like users and then uses HTTP verbs like – GET, PUT, POST, DELETE to modify those resources. And as a developer it should be clear that what needs to be done just by looking at the endpoint and HTTP method used.

URI	HTTP verb	Description
api/users	GET	Get all users
api/users/new	GET	Show form for adding new user
api/users	POST	Add a user
api/users/1	PUT	Update a user with id = 1
api/users/1/edit	GET	Show edit form for user with id = 1
api/users/1	DELETE	Delete a user with id = 1
api/users/1	GET	Get a user with id = 1

- Always use plurals in URL to keep an API URI consistent throughout the application.
- Send a proper HTTP code to indicate a success or error status.

Note : You can easily use GET and POST but in order to use PUT and DELETE you will need to install method override. You can do this by following below code :

```
npm install method-override --save
```

This simply require this package in your code by writing :

```
var methodOverride = require("method-override");
```

Now you can easily use PUT and DELETE routes :

```
app.use(methodOverride("_method"));
```

HTTP verbs: Some of the common HTTP methods/verbs are described below:

- **GET:** Retrieves one or more resources identified by the request URI and it can cache the information receive.
- **POST:** Create a resource from the submission of a request and response is not cacheable in this case. This method is unsafe if no security is applied to the endpoint as it would allow anyone to create a random resource by submission.
- **PUT:** Update an existing resource on the server specified by the request URI.
- **DELETE:** Delete an existing resource on the server specified by the request URI. It always return an appropriate HTTP status for every request.
- GET, PUT, DELETE methods are also known as Idempotent methods. Applying an operation once or applying it multiple times has the same effect. Example: Delete any resource from the server and it succeeds with 200 OK and then try again to delete that resource than it will display an error message 410 GONE.

Difference between REST API and SOAP API

There is no direct comparison between SOAP and REST APIs. But there are some points to be listed below which makes you choose better between these two web services. Here are:

- SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol and REST stands for **RE**presentational **S**tate **T**ransfer.
- Since SOAP is a protocol, it follows a strict standard to allow communication between the client and the server whereas REST is an architectural style that doesn't follow any strict standard but follows

six constraints defined by Roy Fielding in 2000. Those constraints are – Uniform Interface, Client-Server, Stateless, Cacheable, Layered System, Code on Demand.

- SOAP uses only XML for exchanging information in its message format whereas REST is not restricted to XML and its the choice of implementer which Media-Type to use like XML, JSON, Plain-text. Moreover, REST can use SOAP protocol but SOAP cannot use REST.
- On behalf of services interfaces to business logic, SOAP uses @WebService whereas REST instead of using interfaces uses URI like @Path.
- SOAP is difficult to implement and it requires more bandwidth whereas REST is easy to implement and requires less bandwidth such as smartphones.
- Benefits of SOAP over REST as SOAP has ACID compliance transaction. Some of the applications require transaction ability which is accepted by SOAP whereas REST lacks in it.
- On the basis of Security, SOAP has SSL(**S**ecure **S**ocket **L**ayer) and WS-security whereas REST has SSL and HTTPS. In the case of Bank Account Password, Card Number, etc. SOAP is preferred over REST. The security issue is all about your application requirement, you have to build security on your own. It's about what type of protocol you use.

Basics of SOAP – Simple Object Access Protocol

Introduction:

Simple Object Access Protocol(SOAP) is a network protocol for exchanging structured data between nodes. It uses XML format to transfer messages. It works on top of application layer protocols like HTML and SMTP for notations and transmission. SOAP allows processes to communicate throughout platforms, languages and operating systems, since protocols like HTTP are already installed on all platforms.

SOAP was designed by Bob Atkinson, Don Box, Dave Winer, and Mohsen Al-Ghosein at Microsoft in 1998. SOAP was maintained by the XML Protocol Working Group of the World Wide Web Consortium until 2009.

Message Format:

SOAP message transmits some basic information as given below

- Information about message structure and instructions on processing it.
- Encoding instructions for application defined data types.
- Information about Remote Procedure Calls and their responses.

The message in XML format contains three parts

1. **Envelope:**
   It specifies that the XML message is a SOAP message. A SOAP message can be defined as an XML document containing header and body encapsulated in the envelope. The fault is within the body of the message.
2. **Header:**
   This part is not mandatory. But when it is present it can provide crucial information about the applications.
3. **Body:**
   It contains the actual message that is being transmitted. Fault is contained within the body tags.
4. **Fault:**
   This section contains the status of the application and also contains errors in the application. This section is also optional. It should not appear more than once in a SOAP message.

Sample Message:

```
Content-Type: application/soap+xml

<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">

    <env:Header>

        <m:GetLastTradePrice xmlns:m="Some-URI" />

    </env:Header>

    <env:Body>

        <symbol xmlns:p="Some-URI" >DIS</symbol>

    </env:Body>

</env:Envelope>
```

Source: https://tools.ietf.org/html/rfc4227

Advantages of SOAP

1. SOAP is a light weight data interchange protocol because it is based on XML.
2. SOAP was designed to be OS and Platform independent.
3. It is built on top of HTTP which is installed in most systems.
4. It is suggested by W3 consortium which is like a governing body for the Web.

5. SOAP is mainly used for Web Services and Application Programming

   Interfaces (APIs).