SDJ INTERNATIONAL COLLEGE

Bachelor of Computer Applications (BCA) Programme

Seminar Report

BCA Sem VI
AY 2020-21

API(application Programming Interface)

by

Exam No.	Name of Student
3271	Mansi C. Satani

Seminar Guide by :
Prof. Eeva Kapopara

SDJ INTERNATIONAL COLLEGE

C E R T I F I C A T E

This is to certify that Ms. **Satani Mansi Chimanbhai** examination number **3271** has satisfactorily completed her seminar work entitled **API(Application Programming Interface)** as partial fulfillment of requirements for BCA Sem VI, during the academic year 2020-21.

Date: 17/06/2021

Place: Surat

Dr.Aditi Bhatt
(I/C Principal)
SDJ International College,
Surat

Acknowledgement

The success and final outcome of this seminar required a lot of guidance and assistance from many people and I am extremely fortunate to have got this all along the completion of my seminar work. Whatever I have done is only due to such guidance and assistance and I would not forget to thank them.

I owe our profound gratitude to our Director Mr. Deepak Vaidya, I/c Principal Dr. Aditi Bhatt, Head of Department Mr.Vaibhav Desai and Seminar guide prof. Eeva Kapopara and all other Assistant professors of SDJ International College, who took keen interest on my Seminar work and guided me all along, till the completion of my seminar work by providing all the necessary information for presenting a good Concept. I am extremely grateful to them for providing such a nice support and guidance though they had busy schedule managing the college affairs.

I am thankful and fortunate enough to get support and guidance from all Teaching staffs of Bachelor of Computer Application Department which helped me in successfully completing my seminar work. Also, I would like to extend my sincere regards to all the non-teaching staff of Bachelor of Computer Application Department for their timely support.

Mansi C. Satani

3271

I N D E X

1. Overview of APIs

An API is anything that acts an interface to the outside world or to an external program. API management is critical to the effective usage of APIs.

With the advent of internet communication technologies and tools, every device is now capable of communicating with another devicea. Communication is possible because all the devices speak in one language or have translating tools or functions that help them talk to each other. Every programmer in today's world works with APIs either as a developer of the API or as the consumer of the API.

If you ever read tech magazines or blogs, you've probably seen the abbreviation API. It sounds solid, but what does it mean and why should you bother?

Let's start with a simple example: human communication. We can express our thoughts, needs, and ideas through language (written and spoken), gestures, or facial expressions. Interaction with computers, apps, and websites require user interface components – a screen with a menu and graphical elements, a keyboard, and a mouse.

Software or its elements don't need a graphical user interface to communicate with each other. Software products exchange data and functionalities via machine-readable interfaces

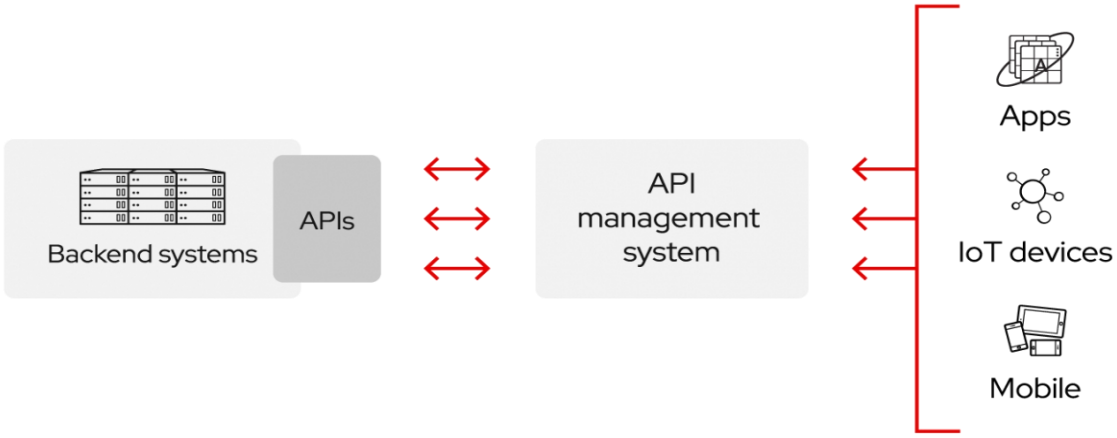
The Red Hat specialists note that APIs are sometimes considered contracts, where documentation is an agreement between the parties: *"If party first sends a remote request structured a particular way, this is how the second party's software will respond."* The API documentation is a manual for developers that includes all necessary information on how to work with the API and use the services it provides. We will talk more about the documentation in one of the next sections.

APIs serve numerous purposes. Generally, they can simplify and speed up software development. Developers can add functionality (i.e., recommended engine, accommodation booking, image recognition, payment processing) from other providers to existing solutions or build new applications using services by third-party providers. In all these cases, specialists don't have to deal with source code, trying to understand how the other solution works. They simply connect their software to another one. In other words, APIs serve as an abstraction layer between two systems, hiding the complexity and working details of the latter.

2. Introduction of APIs

2.1 What is API?

An **API** is a set of programming code that enables data transmission between one software product and another. It also contains the terms of this data exchange.



Application programming interfaces consist of two components:

- Technical specification describing the data exchange options between solutions with the specification done in the form of a request for processing and data delivery protocols
- Software interface written to the specification that represents it

The software that needs to access information (i.e., X hotel room rates for certain dates) or functionality (i.e., a route from point A to point B on a map based on a user's location) from another software, calls its API while specifying the requirements of how data/functionality must be provided. The other software returns data/functionality requested by the former application.

Each API contains and is implemented by **function calls** – language statements that request software to perform particular actions and services. Function calls are phrases composed of verbs and nouns, for example:

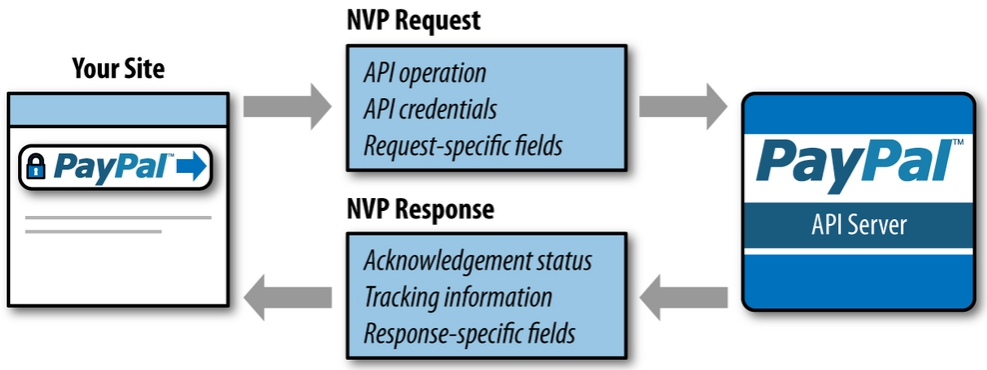
- Start or finish a session
- Get amenities for a single room type
- Restore or retrieve objects from a server.

2.2 What Is an Example of an API?

1.Pay with PayPal

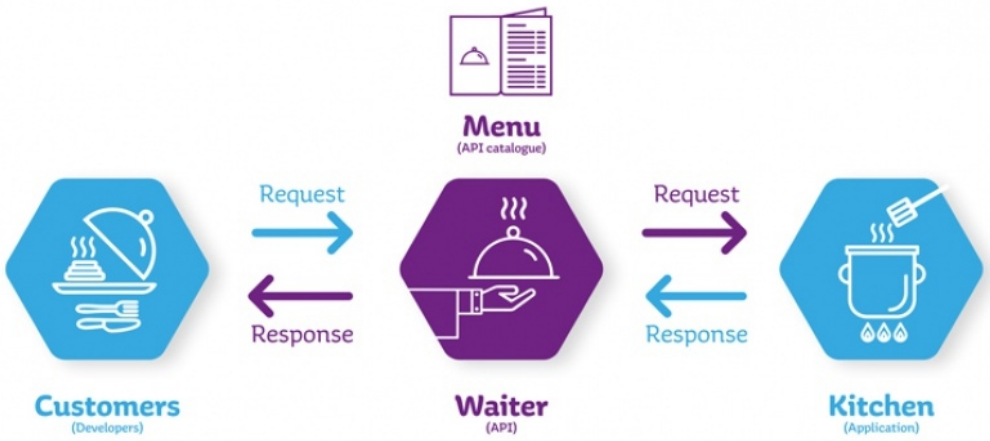
Ever used PayPal to pay for something, directly within an E-Commerce store? Yep, that's also an API at work. Just like with logging-in using a social media service, the "Pay with PayPal" functionality is built with APIs to ensure that the end application can only do what it needs to, without being exposed to sensitive data or gaining access to unintended permissions.

In terms of the inner-workings of this handy function, it's very similar to the log-in process described above. When the user clicks the "Pay with PayPal" button, the application sends an "order" request to the PayPal API, specifying the amount owed and other important details. Then, a pop-up authenticates the user and confirms their purchase. Finally, if everything goes to plan, the API sends confirmation of payment back to the application.



2.Restaurant management

Imagine you're sitting at a table in a restaurant with a menu of choices to order from. What is missing is the critical link to communicate your order to the kitchen and deliver your food back to your table. That's where the waiter or API comes in. The waiter is the messenger – or API – that takes your request or order and tells the kitchen – the system – what to do. Then the waiter delivers the response back to you; in this case, it is the food.



2.3 Why should I use an API?

The need to efficiently share vast amounts of data across various departments and with citizens is an issue that faces most government officials today.

A key tool to tackling this challenge is the Application Programming Interface (API), which at its most basic acts as a door or window into a software program, allowing other programs to interact with it without the need for a developer to share its entire code.

"For example, an API would allow a mobile app, set top box or other connected device in a home to communicate with a service," said Greg Brail, chief architect with Apigee. "The company exposes an API that tells a programmer how they will interact with the service. The API could be open to customers or just people inside their own company."

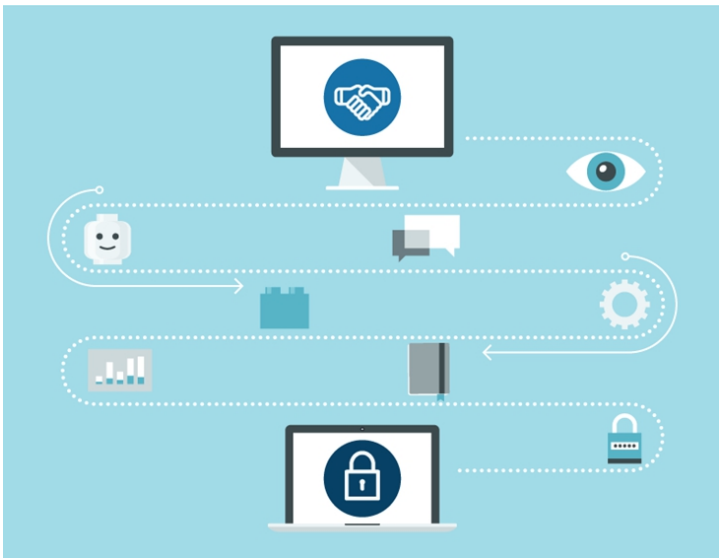
APIs have been used for years in the private sector, but governments at all levels are now playing catch-up to implement the technology across a number of platforms.

"The [API] would have to have some type of credential that would give users permission to access certain types of information," he said. "You can also have an audit trail in the API that would make it possible to see who accessed the data and when."

While government officials have security concerns with the use of APIs, as the technology continues to evolve, the need for municipalities to allow residents to interact with local officials continues to grow. To do so, APIs will play a significant role.

"If we can allow people to build new apps to report things like potholes from their smart phone, we can easily get photos that are geo coded," explained Headd, who became technical evangelist for Accela Inc. earlier this year. "The mobile device knows where you are and should allow for governments to retrieve better information and perhaps better prioritize where the severe problems are."

As the public sector works to catch up to its private sector compatriots in the use of APIs, some government officials have reached out to high profile companies including Amazon, Federal Express and Walmart to discuss each company's transformation into the information age.



2.4 What is API key

An API key or application programming interface key is a code that gets passed in by computer applications. The program or application then calls the API or application programming interface to identify its user, developer or calling program to a website.

Application programming keys are normally used to assist in tracking and controlling how the interface is being utilized. Often, it does this to prevent abuse or malicious use of the API in question.

An API key can act as a secret authentication token as well as a unique identifier. Typically, the key will come with a set of access rights for the API that it is associated with.

- When and Why to Use API Keys

  API keys are used with projects, while authentication is designated for the users. Cloud Endpoints will, in many cases, handle both the authentication procedures as well as the API keys. The differentiating factor between the two is:

  - Authentication tokens are used to identify the users, i.e., the person who is using that particular website or application.
  - API keys are used to identifying the project making the call. This can either be the website or the application that is making the call to the application programming interface.

- Application Programming Interface Keys Guarantee Project Authorization

deciding on the most appropriate scheme, you will need first to understand what authentication and API keys can provide. The keys can provide:

- Project authorization—To help check whether the application making the call has access to call it. It also checks whether the API in this project is enabled.
- Project identification—Identify the project or the application making the call to the API.

You should note that the API keys are not as secure as the tokens used for authentication purposes. However, they do assist in identifying the project or the application that is behind the call.

The keys get generated on the project that is behind the call. This means that you can easily restrict their usage to environments such as an iOS or Android application. You could also use an IP range to restrict usage.

The ability to identify the project making the call means that the API keys can be used to associate use information with a given project. The keys also make it possible for the ESP (Extensible Service Proxy) to reject calls coming from projects that do not have access or which have not been enabled in that particular API.

- User Authentication

Authentication schemes are designed to serve two main purposes:

1. User authorization—check whether the user making the call has permission to make this kind of request.
2. User authentication—verify that the person making the call is the person he or she is claiming to be.

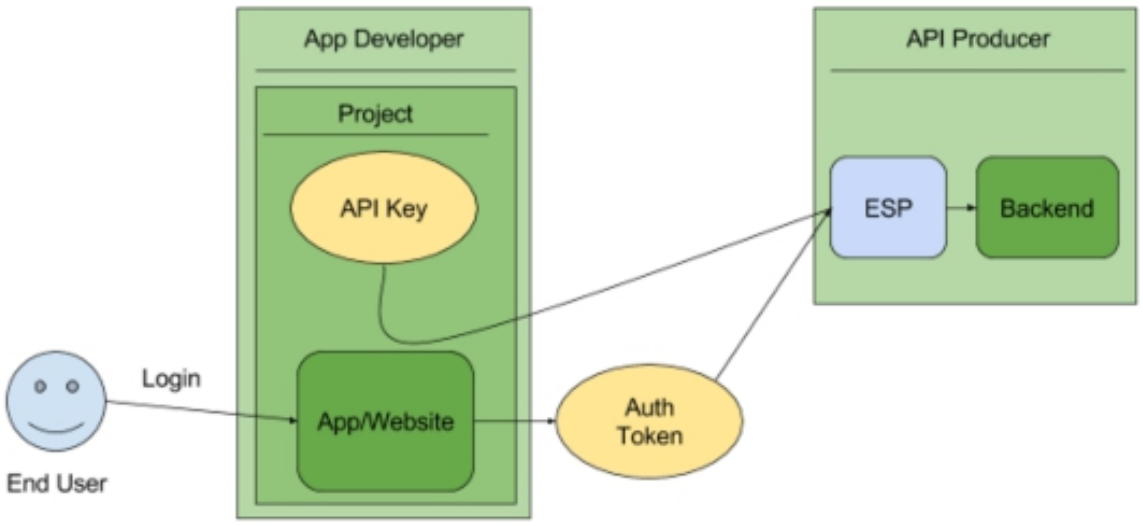
The purpose of an authentication scheme is to make it possible to identify the identity of the caller. An endpoint can also check with the authentication token to confirm that permission has been granted for it to make a call to the API. Based on the information available on the authentication token, the API server gets to make the final decision on whether to authorize that particular request.

- To get an API Key:

1.Go to the APIs & Services > Credentials page.

2.On the Credentials page, click Create credentials > API key.
The API key created dialog displays your newly created API key.

3.Click Close.
The new API key is listed on the Credentials page under API keys.
(Remember to restrict the API key before using it in production.)



2.5 Synchronous and Asynchronous API calls

- Synchronous API calls

  If an API call is synchronous, it means that code execution will block (or wait) for the API call to return before continuing. This means that until a response is returned by the API, your application will not execute any further, which could be perceived by the user as latency or performance lag in your app. Making an API call synchronously can be beneficial, however, if there if code in your app that will only execute properly once the API response is received.

- Asynchronous API calls

  Asynchronous calls do not block (or wait) for the API call to return from the server. Execution continues on in your program, and when the call returns from the server, a “callback” function is executed.An API may be asynchronous where data or service availability and connectivity are low or over-saturated with demand.

3.Structure of APIs

3.1 How API works

How are APIs used in the real world? Here's a very common scenario of the API economy at work: booking a flight.

When you search for flights online, you have a menu of options to choose from. You choose a departure city and date, a return city and date, cabin class, and other variables like your meal, your seat, or baggage requests.

To book your flight, you need to interact with the airline's website to access the airline's database to see if any seats are available on those dates, and what the cost might be based on the date, flight time, route popularity, etc.

You need access to that information from the airline's database, whether you're interacting with it from the website or an online travel service that aggregates information from multiple airlines. Alternatively, you might be accessing the information from a mobile phone. In any case, you need to get the information, and so the application must interact with the airline's API, giving it access to the airline's data.

The API is the interface that, like your helpful waiter, runs and delivers the data from the application you're using to the airline's systems over the Internet. It also then takes the airline's response to your request and delivers right back to the travel application you're using. Moreover, through each step of the process, it facilitates the interaction between the application and the airline's systems – from seat selection to payment and booking.

APIs do the same for all interactions between applications, data, and devices. They allow the transmission of data from system to system, creating connectivity. APIs provide a standard way of accessing any application data, or device, whether it's accessing cloud applications like Salesforce, or shopping from your mobile phone.

3.2 APIs by use cases

APIs can be classified according to the systems for which they are designed.

1. Database APIs.

Database APIs enable communication between an application and a database management system. Developers work with databases by writing queries to access data, change tables, etc. The Drupal 7 Database API, for example, allows users to write unified queries for different databases, both proprietary and open source (Oracle, MongoDB, PostgreSQL, MySQL, CouchDB, and MSSQL).

Another example is ORDS database API, which is embedded into Oracle REST Data Services.

2. Operating systems APIs.

This group of APIs defines how applications use the resources and services of operating systems. Every OS has its set of APIs, for instance, Windows API or Linux API (kernel–user space API and kernel internal API).

Apple provides API reference for macOS and iOS in its developer documentation. APIs for building applications for Apple's macOS desktop operating system are included in the Cocoa set of developer tools. Those building apps for the iOS mobile operating system use Cocoa Touch – a modified version of Cocoa.

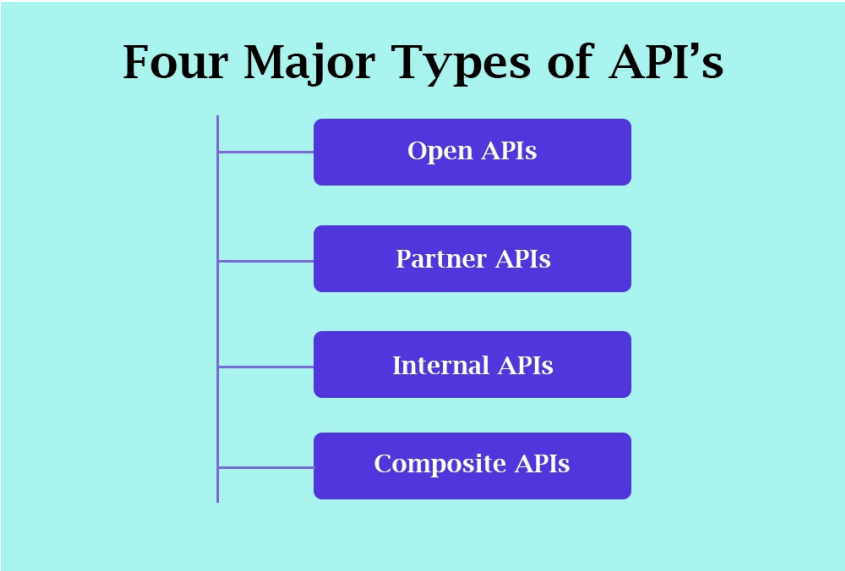
3. Remote APIs.

Remote APIs define standards of interaction for applications running on different machines. In other words, one software product accesses resources located outside the device that requests them, which explains the name. Since two remotely located applications are connected over a communications network, particularly the internet, most remote APIs are written based on web standards. Java Database Connectivity API and Java Remote Method Invocation API are two examples of remote application programming interfaces.

4. Web APIs.

This API class is the most common. Web APIs provide machine-readable data and functionality transfer between web-based systems which represent client-server architecture. These APIs mainly deliver requests from web applications and responses from servers using Hypertext Transfer Protocol (HTTP).

3.3. Local APIs



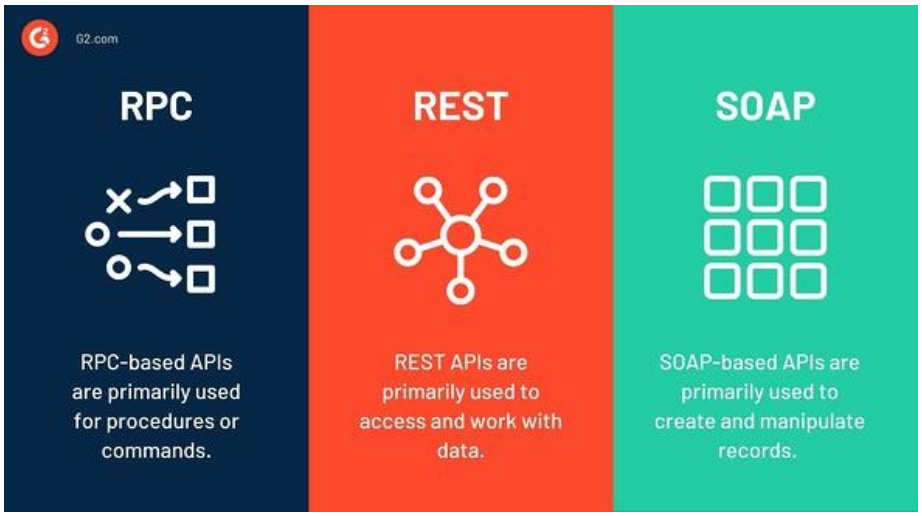
1.Open APIs, aka **Public APIs,** are publicly available to developers and other users with minimal restriction. They may require registration, use of an API Key or Oath, or maybe completely open. They focus on external users, to access data or services.

2.Partner APIs are APIs exposed by/to the strategic business partners. They are not available publicly and need specific entitlement to access them. Like open APIs, partner APIs are the tip of the iceberg because they are the most visible ones and are used to communicate beyond the boundaries of the company. They are usually exposed to a public API developer portal that developers can access in self-service mode. While open APIs are completely open, there is an on-boarding process with a specific validation workflow to get access to partner APIs.

3.Internal APIs, aka **private APIs,** are hidden from external users and only exposed by internal systems. Internal APIs are not meant for consumption outside of the company but rather for use across different internal development teams for better productivity and reuse of services. A good governance process comprises exposing them to an internal API developer portal that connects to the internal IAM systems to authenticate and allow users to access the right set of APIs.

4.Composite APIs combine multiple data or service APIs. They are built using the API orchestration capabilities of an API creation tool. They allow developers to access several endpoints in one call. Composite APIs are useful, for example, in a micro-services architecture pattern where you need information from several services to perform a single task.

3.4 Web service APIs



Apart from the main web APIs, there are also web service APIs:

- SOAP
- REST
- RPC :-
    - XML-RPC
    - JSON-RPC

A web service is a system or software that uses an address, i.e., URL on the World Wide Web, to provide access to its services.

The following are the most common types of web service APIs:

1.SOAP (Simple Object Access Protocol): This is a protocol that uses XML as a format to transfer data. Its main function is to define the structure of the messages and methods of communication. It also uses WSDL, or Web Services Definition Language, in a machine-readable document to publish a definition of its interface.

2.REST (Representational State Transfer): REST is not a protocol like the other web services, instead, it is a set of architectural principles. The REST service needs to have certain characteristics, including simple interfaces, which are resources identified easily within the request and manipulation of resources using the interface.

3.XML-RPC: This is a protocol that uses a specific XML format to transfer data compared to SOAP that uses a proprietary XML format. It is also older than SOAP. XML-RPC uses minimum bandwidth and is much simpler than SOAP.

4.JSON-RPC: This protocol is similar to XML-RPC but instead of using XML format to transfer data it uses JSON.

1.SOAP (Simple Object Access Protocol)

- **What is SOAP?**

SOAP or **Simple Objects Access Protocol** is a web communication protocol designed for Microsoft back in 1998. Today, it's mostly used to expose web services and transmit data over HTTP/HTTPS. But it's not limited to them. SOAP, unlike the REST pattern, supports the XML data format only and strongly follows preset standards such as messaging structure, a set of encoding rules, and a convention for providing procedure requests and responses.

The built-in functionality to create web-based services allows SOAP to handle communications and make responses language- and platform-independent.

While most web data exchange happens over REST exchange, SOAP isn't disappearing anytime soon, because it's highly standardized, allows for automation in certain cases, and it's more secure. Let's have a look at the main SOAP features.

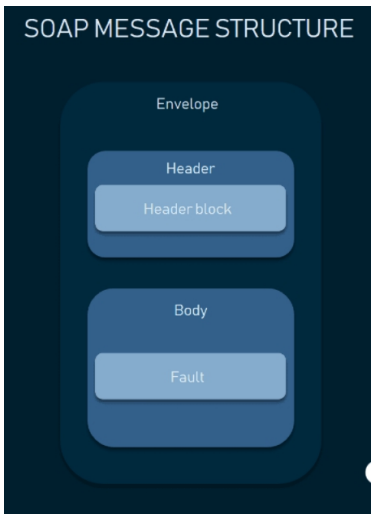
- **SOAP works with XML only**

Web-transmitted data is usually structured in some way. The two most popular data formats are XML and JSON.

XML (or Extensible Markup Language) is a text format that establishes a set of rules to structure messages as both human- and machine-readable records. But XML is verbose as it aims at creating a web document with all its formality. **JSON**, on the other hand, has a loose structure that focuses on the data itself. Have a look at the image below to get the idea.

- **SOAP message structure**

XML isn't the only reason SOAP is considered verbose and heavy compared to REST. It's also the way SOAP messages are composed. Standard SOAP API requests and responses appear as an enveloped message that consists of four elements with specific functions for each one.



- **Envelope** is the core and essential element of every message, which begins and concludes messages with its tags, enveloping it, hence the name.
- **Header** (optional) determines the specifics, extra requirements for the message, e.g. authentication.
- **Body** includes the request or response.
- **Fault** (optional) shows all data about any errors that could emerge throughout the API request and response.

- **Web Service Description Language (WSDL) document**

One of the major features of SOAP APIs is that they almost always use a WSDL document.

Simply put, a WSDL document is an XML description of a web service. It acts as a guideline of how to communicate with a web service, defining the endpoints and describing all processes that could be performed by the exposed applications. These may include data types being used inside the SOAP messages and all actions available via the web service. Thus, a WSDL file serves as a signed contract between a client and a server.

- **SOAP stateful and stateless messaging**

The beginning of the 21st century is remembered for the internet boom. Thousands of internet-driven companies were emerging and millions of users were accessing the web every day. Now imagine that a single server starts receiving thousands of requests from users (clients) simultaneously. And if this resource does something more complex than show walls of text, things can get slow. For instance, if users check the upcoming flights schedule and must drill down to each flight detail, the server must be aware of what's happening with the client, right?

It appears that you can handle this situation in two ways: using stateful and stateless operations. And SOAP supports both.

Stateful means that the server keeps the information that it receives from a client across multiple requests. For instance, first it memorizes the flight dates that you're looking for and then provides information on the pricing after the second request. This allows you to chain messages together, making the server aware of the previous requests. Stateful messaging may be crucial in operations involving multiple parties and complex transactions, e.g. bank operations or flight booking. But still, it's really heavy to a server.

Stateless communication means that each message contains enough information about the state of the client so that a server doesn't have to be bothered with it. Once the server returns requested data, it forgets about the client. Each request is isolated from the previous one. Stateless operations helped reduce server load and increase the speed of communication.

- **Transfer protocols: HTTP, TCP, SMTP, FTP, and more**

In layman terms, a transfer protocol is a set of rules and commands used to transfer data over the internet. There are low-level protocols like IPv4, which simply delivers data packets from one point to another. There are higher transfer layers, like TCP, which ensures that data is indeed delivered. And, finally, there are application-level protocols that are used by web browsers to communicate with web servers, but don't take charge of the connection itself.

SOAP supports a variety of transfer protocols, both high- and low-level ones. For instance, SOAP allows for messaging via TCP (Transaction Control Protocol), a low-level data exchange method that works between ports via an IP network. You can go for the SMTP (Simple Mail Transfer Protocol) option, which is a communication protocol for electronic mail transmission, FTP (File Transfer Protocol), and any other transfer method that supports text data exchange.

Does it make any sense to send data using other protocols than HTTP/HTTPS? In most cases, it doesn't. SOAP was primarily designed to work with HTTP. But there may be scenarios, such as security constraints, server requirements, solution architectures, or simply speed that will benefit from this SOAP versatility.

- **SOAP Use Cases**

Considering these differences, it gets obvious why web messaging is mostly done with REST. To wrap things up, let's define the cases when SOAP is still the major technology.

Highly standardized operations: billing, navigation, facilities. All use cases where you have to eliminate any kind of misinterpretation are a good fit for SOAP communication. Usually, these systems have strict contracts with clearly defined logic that can be described with a WSDL document.

Bank transactions and payment systems. When you need your transactions to always be reliable and non-reachable by third parties, SOAP has multiple benefits to consider. First, it's the level of security with ACID compliance and WS-Security protocols. Additionally, this set of use cases usually requires stateful messaging, i.e. using chained transactions that aren't isolated one from another. Since payment systems may have multiple parties involved in a single operation, SOAP allows for better coordination of their behavior.

Flight booking systems. Since flight booking usually involves multiple parties, some providers from this industry still rely on SOAP to handle stateful and chained messaging.

Non-HTTP messaging and legacy environments. If the server requirements and existing systems leverage communication protocols besides HTTP, SOAP is the first option to look at.

2.REST (Representational State Transfer)

A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for

An API is a set of definitions and protocols for building and integrating application software. It's sometimes referred to as a contract between an information provider and an information user—establishing the content required from the consumer (the call) and the content required by the producer (the response). For example, the API design for a weather service could specify that the user supply a zip code and that the producer reply with a 2-part answer, the first being the high temperature, and the second being the low.

In other words, if you want to interact with a computer or system to retrieve information or perform a function, an API helps you communicate what you want to that system so it can understand and fulfill the request.

You can think of an API as a mediator between the users or clients and the resources or web services they want to get. It's also a way for an organization to share resources and information while maintaining security, control, and authentication—determining who gets access to what.

Another advantage of an API is that you don't have to know the specifics of caching—how your resource is retrieved or where it comes from.

REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways.

When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular programming language to use because, despite its name, it's language-agnostic, as well as readable by both humans and machines.

Something else to keep in mind: Headers and parameters are also important in the HTTP methods of a RESTful API HTTP request, as they contain important identifier information as to the request's metadata, authorization, uniform resource identifier (URI), caching, cookies, and more. There are request headers and response headers, each with their own HTTP connection information and status codes.

In order for an API to be considered RESTful, it has to conform to these criteria:

- A client-server architecture made up of clients, servers, and resources, with requests managed through HTTP.
- Stateless client-server communication, meaning no client information is stored between get requests and each request is separate and unconnected.
- Cache-able data that streamlines client-server interactions.
- A uniform interface between components so that information is transferred in a standard form. This requires that:

  - resources requested are identifiable and separate from the representations sent to the client.
  - resources can be manipulated by the client via the representation they receive because the representation contains enough information to do so.
  - self-descriptive messages returned to the client have enough information to describe how the client should process it.
  - hypertext/hypermedia is available, meaning that after accessing a resource the client should be able to use hyperlinks to find all other currently available actions they can take.
- A layered system that organizes each type of server (those responsible for security, load-balancing, etc.) involved the retrieval of requested information into hierarchies, invisible to the client.
- Code-on-demand (optional): the ability to send executable code from the server to the client when requested, extending client functionality.

Though the REST API has these criteria to conform to, it is still considered easier to use than a prescribed protocol like SOAP (Simple Object Access Protocol), which has specific requirements like XML messaging, and built-in security and transaction compliance that make it slower and heavier.

The Anatomy Of A Request

It's important to know that a request is made up of four things:

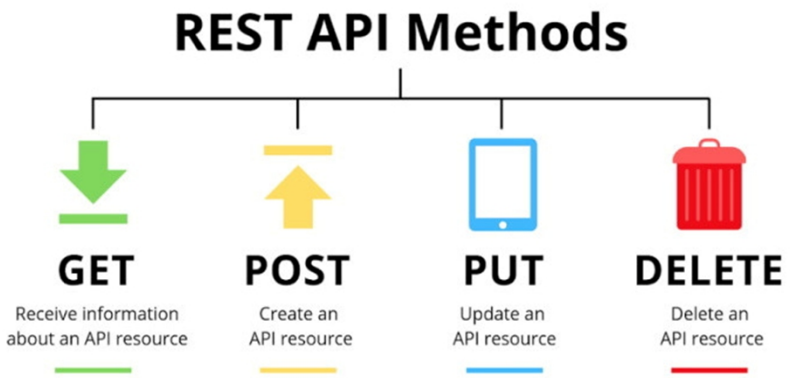
1. The endpoint
2. The method
3. The headers
4. The data (or body)

Methods for REST APIs

The method is the type of request you send to the server. You can choose from these five types below:

- GET
- POST
- PUT
- DELETE

These methods provide meaning for the request you're making. They are used to perform four possible actions: Create, Read, Update and Delete (CRUD).



Method Name	Request Meaning
`GET`	This request is used to get a resource from a server. If you perform a `GET` request, the server looks for the data you requested and sends it back to you. In other words, a `GET` request performs a `READ` operation. This is the default request method.
`POST`	This request is used to create a new resource on a server. If you perform a `POST` request, the server creates a new entry in the database and tells you whether the creation is successful. In other words, a `POST` request performs an `CREATE` operation.
`PUT` and `PATCH`	These two requests are used to update a resource on a server. If you perform a `PUT` or `PATCH` request, the server updates an entry in the database and tells you whether the update is successful. In other words, a `PUT` or `PATCH` request performs an `UPDATE` operation.
`DELETE`	This request is used to delete a resource from a server. If you perform a `DELETE` request, the server deletes an entry in the database and tells you whether the deletion is successful. In other words, a `DELETE` request performs a `DELETE` operation.

- **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 tutorial will go into some of the key differences between these techniques as well as what challenges you might encounter while using them.

Below are the main differences between SOAP and REST

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 ➢ Client Server ➢ Stateless ➢ Cache-able ➢ Layered System ➢ 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

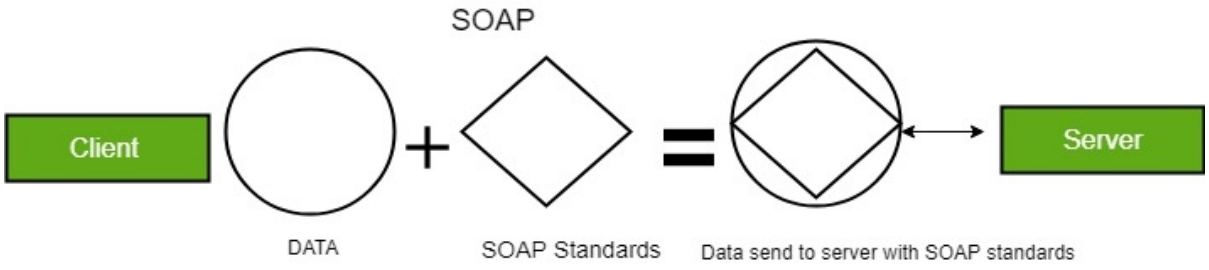
- SOAP requires more bandwidth for its usage. Since SOAP Messages contain a lot of information inside of it, the amount of data transfer using SOAP is generally a lot.

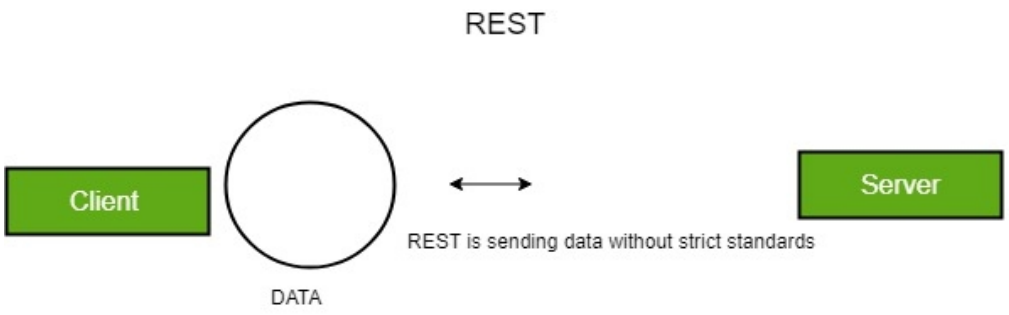
```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV
="http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle
=" http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
<Demo.guru99WebService
xmlns="http://tempuri.org/">
<EmployeeID>int</EmployeeID>
</Demo.guru99WebService>
</soap:Body>
</SOAP-ENV:Envelope>
```

- 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"}
```

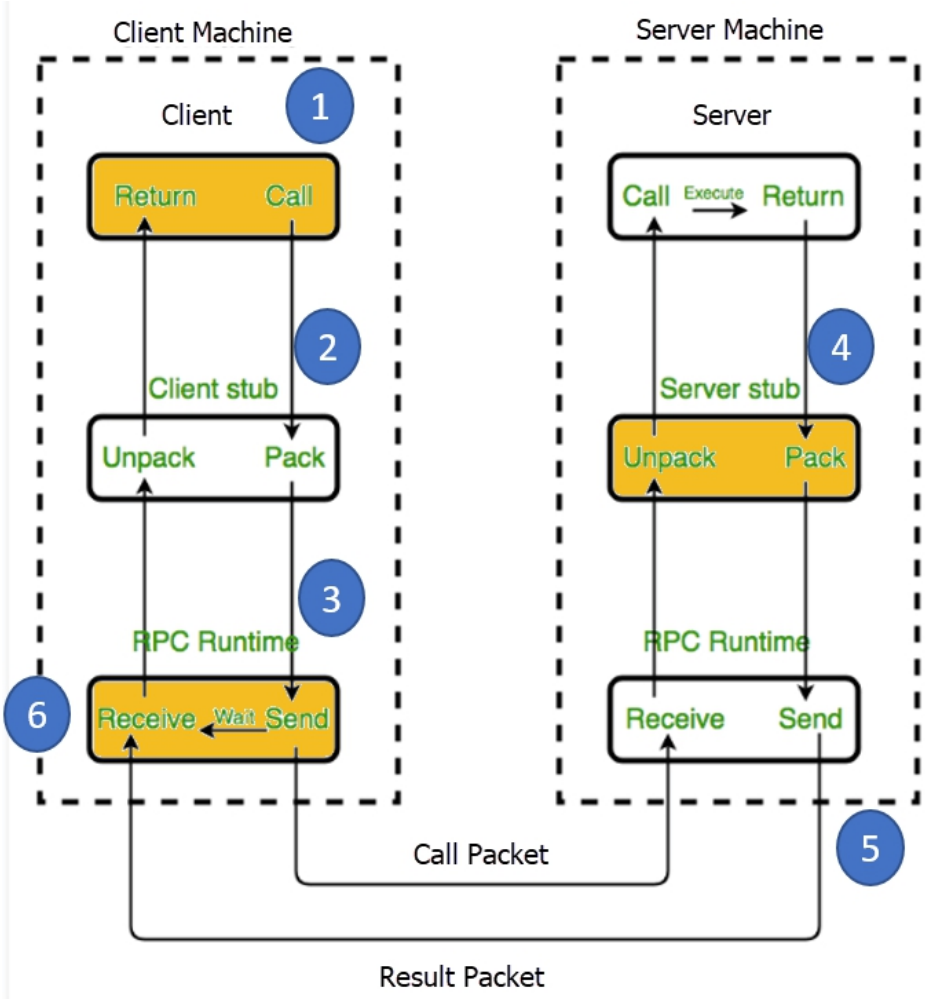
- SOAP can only work with XML format. As seen from SOAP messages, all data passed is in XML format.
- REST permits different data format such as Plain text, HTML, XML, JSON, etc. But the most preferred format for transferring data is JSON.





3.How RPC works

A client invokes a remote procedure, serializes the parameters and additional information into a message, and sends the message to a server. On receiving the message, the server deserializes its content, executes the requested operation, and sends a result back to the client. The server stub and client stub take care of the serialization and deserialization of the parameters.



- **RPC use cases**

The RPC pattern started being used around the 80s, but this doesn't automatically make it obsolete. Big companies like Google, Facebook (Apache Thrift), and Twitch (Twirp) are using RPC high-performance variates internally to perform extremely high-performance, low-overhead messaging. Their massive microservices systems require internal communication to be clear while arranged in short messages.

➢ XML-RPC

RPC stands for Remote Procedure Call. As its name indicates, it is a mechanism to call a procedure or a function available on a remote computer. RPC is a much older technology than the Web. Effectively, RPC gives developers a mechanism for defining interfaces that can be called over a network. These interfaces can be as simple as a single function call or as complex as a large API.

● What is XML-RPC ?

- ➢ XML-RPC is among the simplest and most foolproof web service approaches that makes it easy for computers to call procedures on other computers.
- ➢ XML-RPC permits programs to make function or procedure calls across a network.
- ➢ XML-RPC uses the HTTP protocol to pass information from a client computer to a server computer.
- ➢ XML-RPC uses a small XML vocabulary to describe the nature of requests and responses.
- ➢ XML-RPC client specifies a procedure name and parameters in the XML request, and the server returns either a fault or a response in the XML response.
- ➢ XML-RPC parameters are a simple list of types and content - structs and arrays are the most complex types available.
- ➢ XML-RPC has no notion of objects and no mechanism for including information that uses other XML vocabulary.
- ➢ With XML-RPC and web services, however, the Web becomes a collection of procedural connections where computers exchange information along tightly bound paths.
- ➢ XML-RPC emerged in early 1998; it was published by UserLand Software and initially implemented in their Frontier product.

● Why XML-RPC ?

If you need to integrate multiple computing environments, but don't need to share complex data structures directly, you will find that XML-RPC lets you establish communications quickly and easily.

Even if you work within a single environment, you may find that the RPC approach makes it easy to connect programs that have different data models or processing expectations and that it can provide easy access to reusable logic.

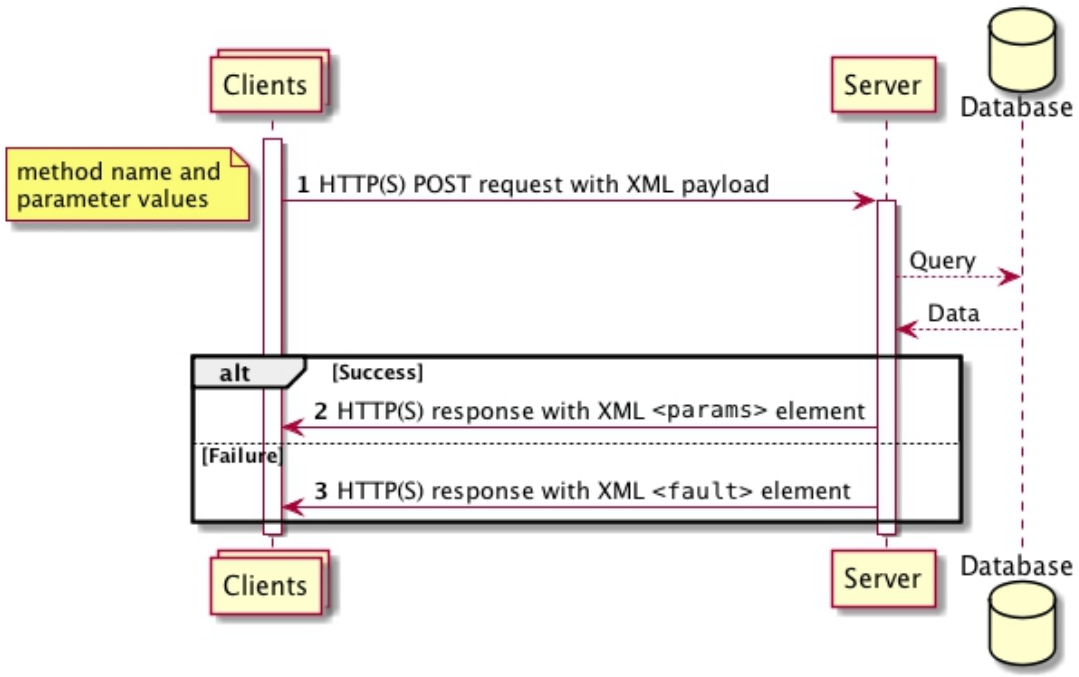
- ➢ XML-RPC is an excellent tool for establishing a wide variety of connections between computers.
- ➢ XML-RPC offers integrators an opportunity to use a standard vocabulary and approach for exchanging information.
- ➢ XML-RPC's most obvious field of application is connecting different kinds of environments, allowing Java to talk with Perl, Python, ASP, and so on.

- **Basic Data Types in XML-RPC**

Type	Value	Examples
int or i4	32-bit integers between -2,147,483,648 and 2,147,483,647.	<int>27</int> <i4>27</i4>
double	64-bit floating-point numbers	<double>27.31415</double> <double>-1.1465</double>
Boolean	true (1) or false (0)	<boolean>1</boolean> <boolean>0</boolean>
string	ASCII text, though many implementations support Unicode	<string>Hello</string> <string>bonkers! @</string>
dateTime.iso8601	Dates in ISO8601 format: CCYYMMDDTHH:MM:SS	<dateTime.iso8601> 20021125T02:20:04 </dateTime.iso8601> <dateTime.iso8601> 20020104T17:27:30 </dateTime.iso8601>
base64	Binary information encoded as Base 64, as defined in RFC 2045	<base64>SGVsbG8sIFdvcmxkIQ==</base64>

These basic types are always enclosed in *value* elements. Strings (and only strings) may be enclosed in a *value* element but omit the *string* element. These basic types may be combined into two more complex types, arrays, and structs. Arrays represent sequential information, while structs represent name-value pairs, much like hashtables, associative arrays, or properties.

- **XML -RPC Request**



XML-RPC requests are a combination of XML content and HTTP headers. The XML content uses the data typing structure to pass parameters and contains additional information identifying which procedure is being called, while the HTTP headers provide a wrapper for passing the request over the Web.

Each request contains a single XML document, whose root element is a *methodCall* element. Each *methodCall* element contains a *methodName* element and a *params* element. The *methodName* element identifies the name of the procedure to be called, while the *params* element contains a list of parameters and their values. Each *params* element includes a list of param elements which in turn contain *value* elements.

For example, to pass a request to a method called *circle-area*, which takes a *Double* parameter (for the radius), the XML-RPC request would look like:

➢ JSON-RPC

JSON-RPC is a remote procedure call protocol encoded in JSON. It is similar to the XML-RPC protocol, defining only a few data types and commands. JSON-RPC allows for notifications (data sent to the server that does not require a response) and for multiple calls to be sent to the server which may be answered asynchronously.

● Usage

JSON-RPC works by sending a request to a server implementing this protocol. The client in that case is typically software intending to call a single method of a remote system. Multiple input parameters can be passed to the remote method as an array or object, whereas the method itself can return multiple output data as well. (This depends on the implemented version.)

All transfer types are single objects, serialized using JSON.[1] A request is a call to a specific method provided by a remote system. It can contain three members:

- `method` - A String with the name of the method to be invoked. Method names that begin with "rpc." are reserved for rpc-internal methods.
- `params` - An Object or Array of values to be passed as parameters to the defined method. This member may be omitted.
- `id` - A string or non-fractional number used to match the response with the request that it is replying to.[2] This member may be omitted if no response should be returned.[3]

The receiver of the request must reply with a valid response to all received requests. A response can contain the members mentioned below.

- `result` - The data returned by the invoked method. This element is formatted as a JSON-stat object. If an error occurred while invoking the method, this member must not exist.[4]
- `error` - An error object if there was an error invoking the method, otherwise this member must not exist.[5] The object must contain members *code* (integer) and *message* (string).[6] An optional *data* member can contain further server-specific data. There are pre-defined error codes which follow those defined for XML-RPC.

- `id` - The id of the request it is responding to.

Request and response:

```
--> {"jsonrpc": "2.0", "method": "subtract", "params": {"minuend": 42, "subtrahend": 23}, "id": 3}<-- {"jsonrpc": "2.0", "result": 19, "id": 3}
```

Notification (no response):

```
--> {"jsonrpc": "2.0", "method": "update", "params": [1,2,3,4,5]}
```

3.5 Advantages / Disadvantages of web-service APIs.

- **Advantages of SOAP API :-**

- **Language- and platform-agnostic.** The built-in functionality to create web-based services allows SOAP to handle communications and make responses language- and platform-independent.

- **Bound to a variety of transport protocols.** SOAP is flexible in terms of transfer protocols to accommodate for multiple scenarios.

- **Built-in error handling.** SOAP API specification allows for returning the Retry XML message with error code and its explanation.

- **A number of security extensions.** Integrated with the WS-Security protocols, SOAP meets an enterprise-grade transaction quality. It provides privacy and integrity inside the transactions while allowing for encryption on the message level.

- **Advantages of REST API :-**

- **Decoupled client and server.** Decoupling the client and the server as much as possible, REST allows for a better abstraction than RPC. A system with abstraction levels is able to encapsulate its details to better identify and sustain its properties. This makes a REST API flexible enough to evolve over time while remaining a stable system.

- **Discoverability.** Communication between the client and server describes everything so that no external documentation is required to understand how to interact with the REST API.

- **Cache-friendly.** Reusing a lot of HTTP tools, REST is the only style that allows caching data on the HTTP level. In contrast, caching implementation on any other API will require configuring an additional cache module.

- **Multiple formats support.** The ability to support multiple formats for storing and exchanging data is one of the reasons REST is currently a prevailing choice for building public APIs.

- **Advantages of RPC :-**

- **Straightforward and simple interaction.** RPC uses GET to fetch information and POST for everything else. The mechanics of the interaction between a server and a client come down to calling an endpoint and getting a response.

- **Easy-to-add functions.** If we get a new requirement for our API, we can easily add another endpoint executing this requirement: 1) Write a new function and throw it behind an endpoint and 2) now a client can hit this endpoint and get the info meeting the set requirement.

- **High performance.** Lightweight payloads go easy on the network providing high performance, which is important for shared servers and for parallel computations executing on networks of workstations.

- **Disadvantages of SOAP API :-**

These days, many developers shudder at the idea of having to integrate a SOAP API for several reasons.

- **XML only.** SOAP messages contain a lot of metadata and only support verbose XML structures for requests and responses.
- **Heavyweight.** Due to the large size of XML-files, SOAP services require a large bandwidth.
- **Narrowly specialized knowledge.** Building SOAP API servers requires a deep understanding of all protocols involved and their highly restricted rules.
- **Tedious message updating.** Requiring additional effort to add or remove the message properties, rigid SOAP schema slows down adoption.

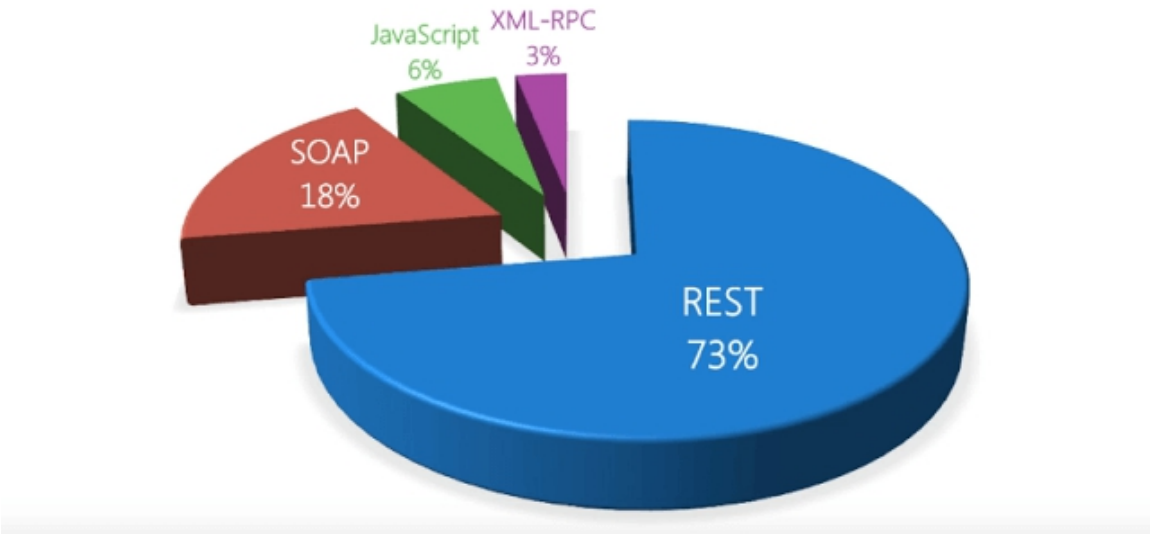
- **Disadvantages of REST API :-**

- **No single REST structure.** There's no exact right way to build a REST API. How to model resources and which resources to model will depend on each scenario. This makes REST simple in theory, but difficult in practice.
- **Big payloads.** REST returns a lot of rich metadata so that the client can understand everything necessary about the state of the application just from its responses. And this chattiness is no big deal for a big network pipe with lots of bandwidth capacity. But that's not always the case. This was the key driving factor for Facebook coming up with the description of GraphQL style in 2012.
- **Over- and under-fetching problems.** Containing either too much data or not enough of it, REST responses often create the need for another request.

- **Disadvantages of RPC :-**
  - **Tight coupling to the underlying system.** An API's abstraction level contributes to its reusability. The tighter it is to the underlying system, the less reusable it will be for other systems. RPC's tight coupling to the underlying system doesn't allow for an abstraction layer between the functions in the system and the external API. This raises security issues as it's quite easy to leak implementation details about the underlying system into the API. An RPC's tight coupling makes scalability requirements and loosely coupled teams hard to achieve. **Low discoverability.** In RPC there's no way to introspect the API or send a request and start understanding what function to call based on its requests.
  - **Function explosion.** It's so easy to create new functions. So, instead of editing the existing ones, we create new ones ending up with a huge list of overlapping functions that are hard to understand.

.

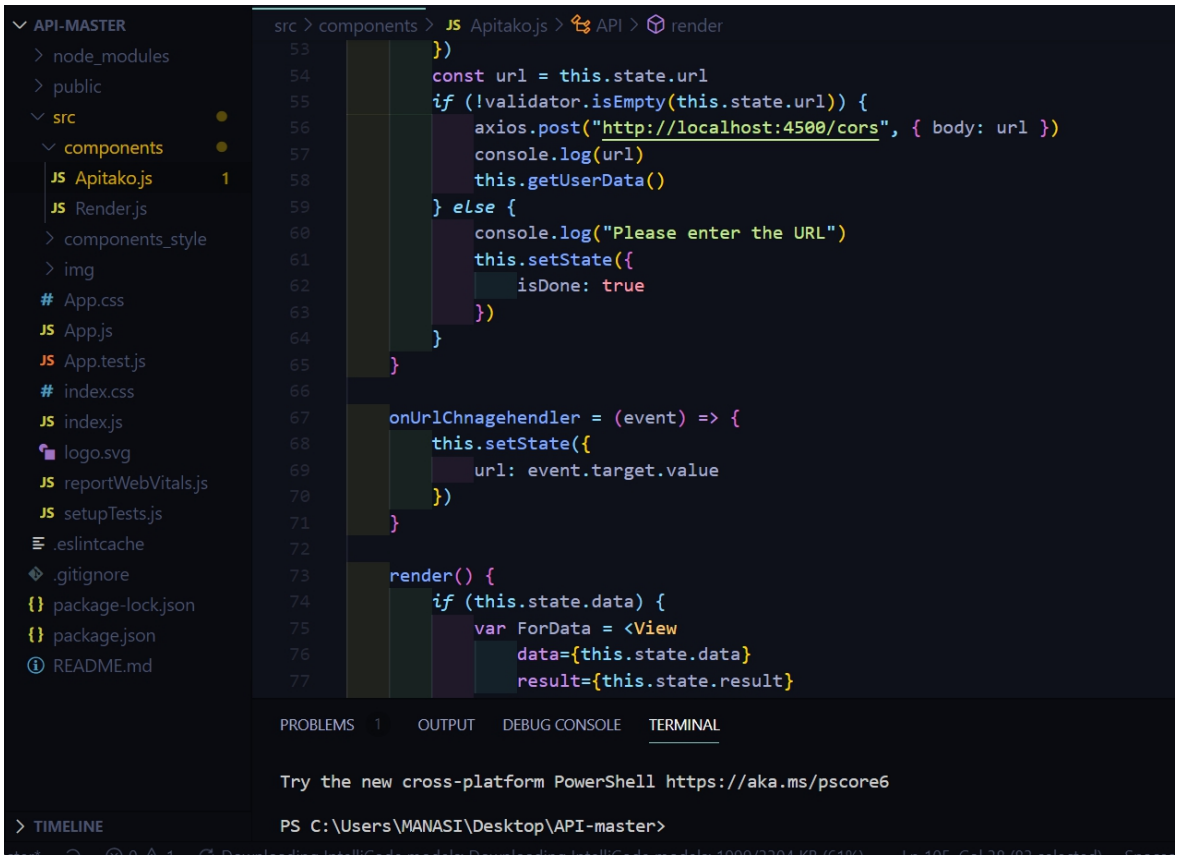
3.6 Enterprise analysis



Here we can see that the REST API has been used 73% while SOAP and RPC (XML , JSON) are used less in comparison of the REST.
So the reason behind using REST more than any other is that it supports both the formats as we saw that in above information. It supports XML as well as the JSON format and as we see in the chart JS has been used more than XML and that maybe the reason of REST been used more than the SOAP.

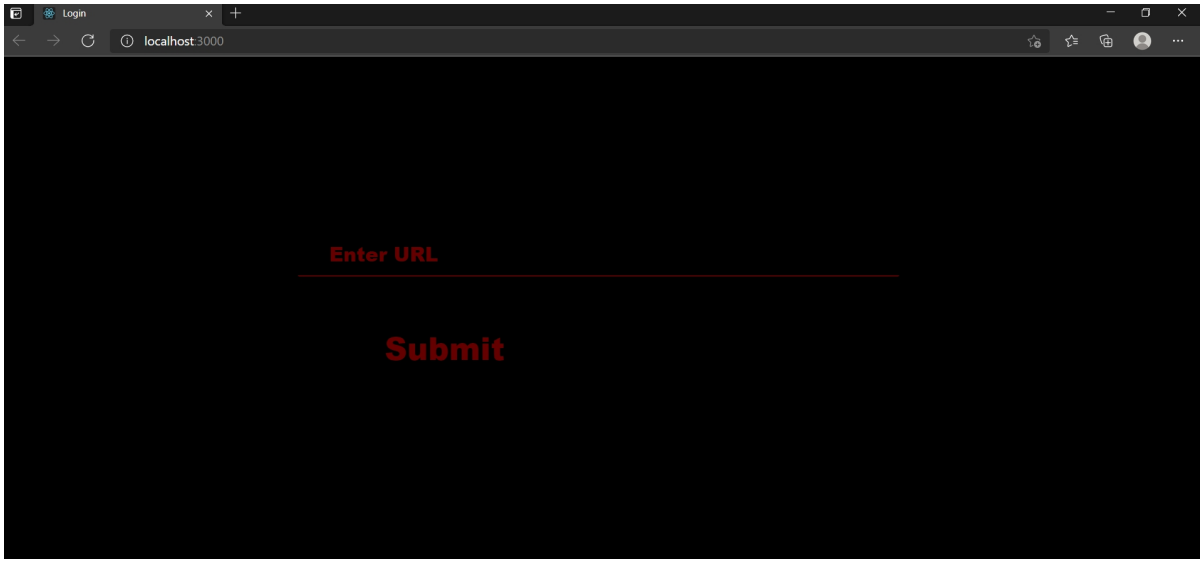
4. Working Model

(4.5.1)



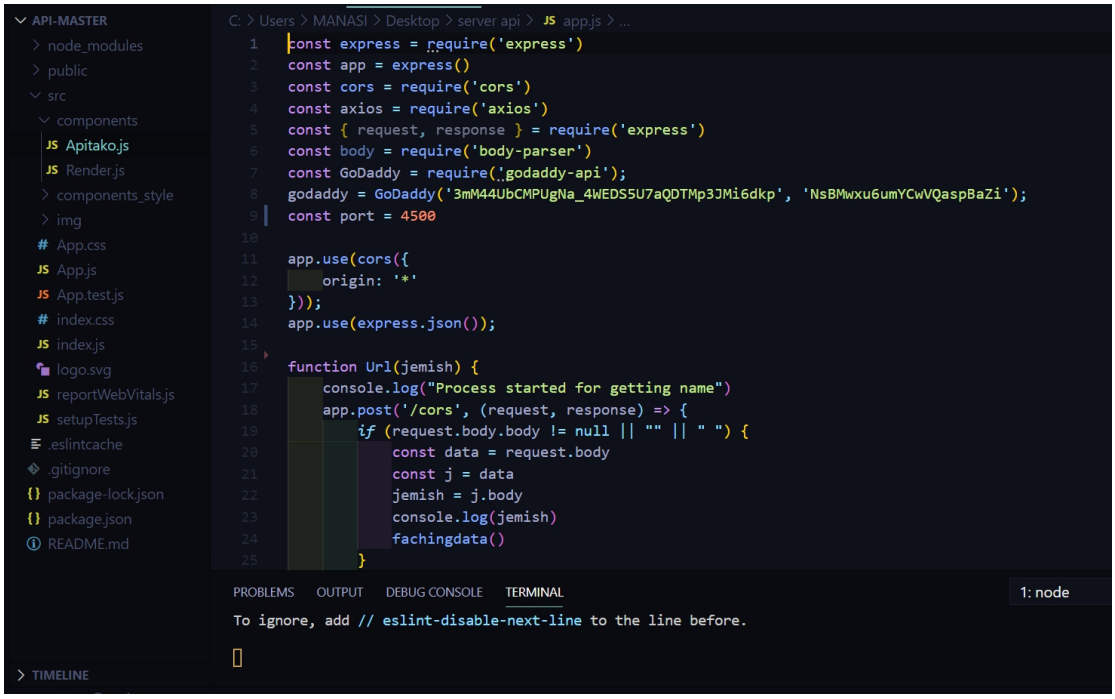
(4.5.2)

Here is the screen-shot(3.5.1) and (3.5.2) of the screen which has code for the loading screen which asks the user for add a website name and then search it by clicking the submit button so that it will look for that website and data of that particular website we can see that loading screen in the following screen-shots(4.5.3) and (4.5.4)



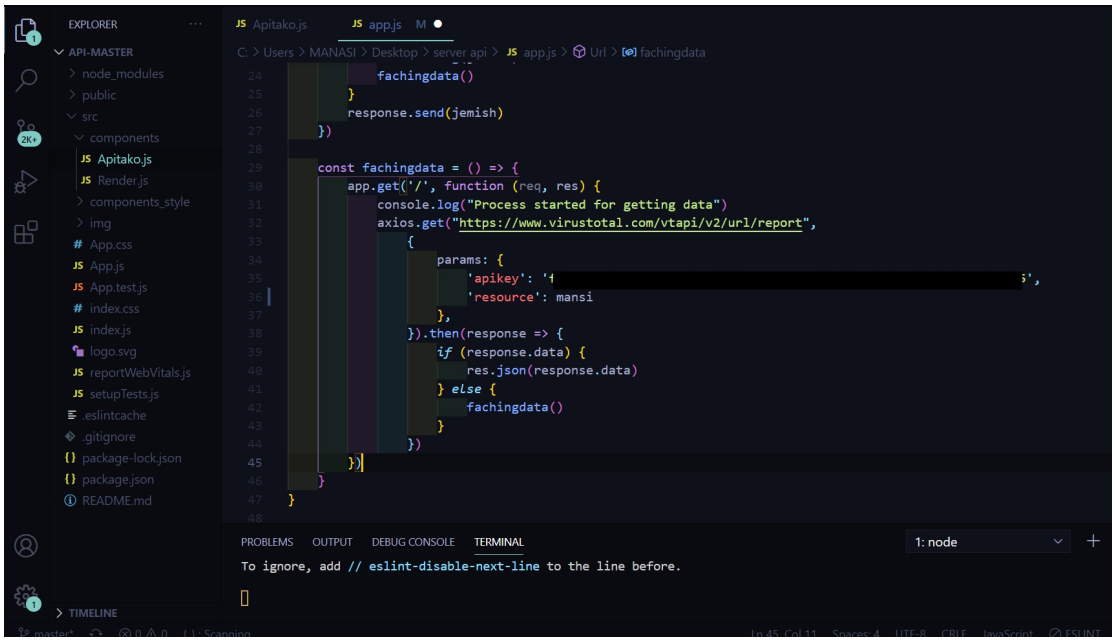
(4.5.3)

(4.5.4)



(4.4.6)

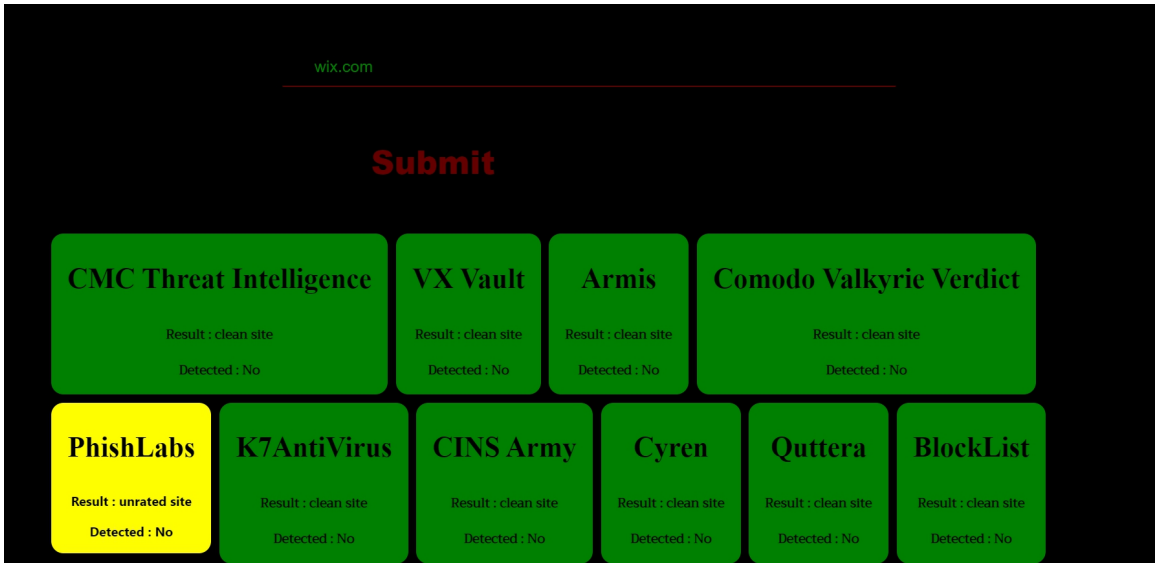
Here is the code of the file which contains the how the app will able to take the data from the API of the virus-total and fetch the data from that API and gives the result but 1st we have to add the API key and then only we can access the API of virus-total so here in the image (4.4.7) we can see the API key and resource of that key that which user wants to get the data.



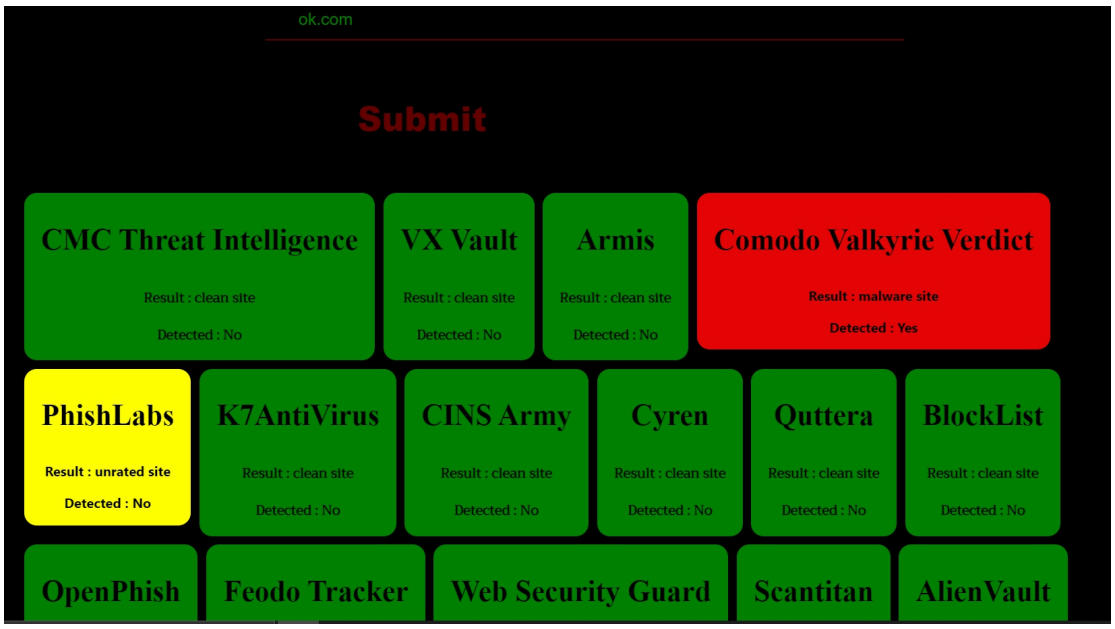
(4.4.7)

Now by these sort of code we can get the result as we see in the given screenshot (4.4.8),(4.4.9),(4.4.10),(4.4.11) that there has been lots of green , red and yellow boxes which contains the virus detecting application names and under that app names there is shown that particular website for which we had search is clean or not .

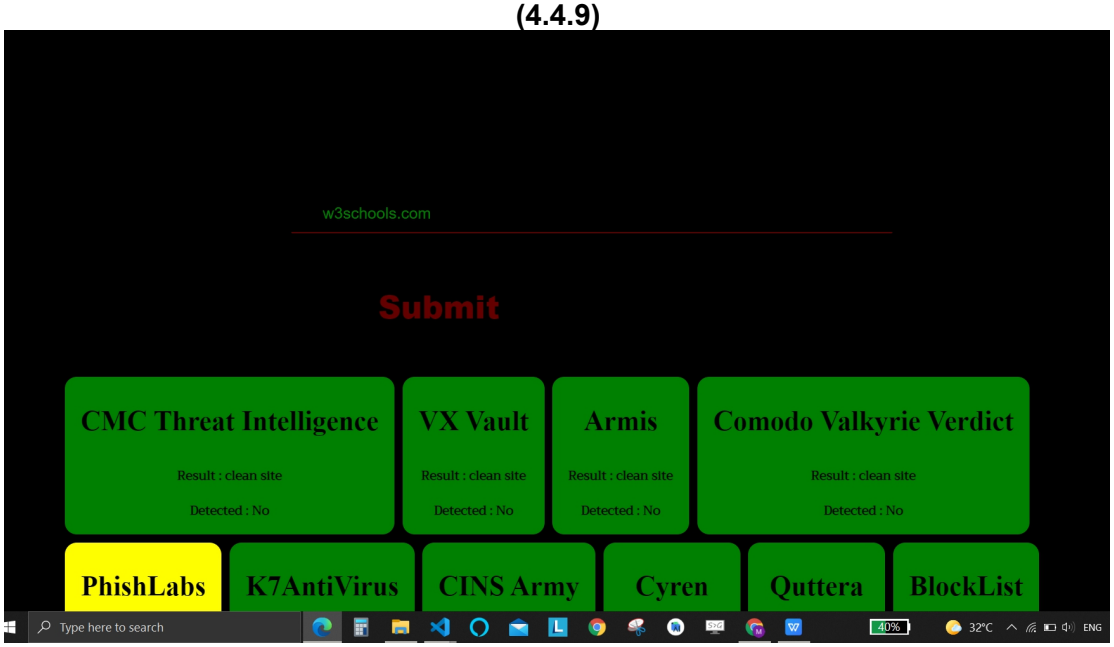
Now there are three types of the boxes :- the red green one shows that site is clean there is no virus detected in that website, the red one shows that their might be some issues with that site and the last yellow one shows that particular application's API doesn't have any site which has that name as we searched.



(4.4.8)



(4.4.9)



(4.4.10)

(4.4.11)

6.Conclusion

Normally, when you are visiting a website in your browser, your browser is asking the website server for access to the request website. If URL matches, then the request is accepted and the website loads in a new page. You can notice this if you visit a website on a day you have particularly slow internet (and a very image/media heavy website) and looking at the lower left of your browser where all steps are quickly written out. API works in similar fashion, but instead of asking to load the website, it request the data that is within the website.json() allows you to receive the data contained within the website. But be aware that it will only return the specific endpoint you asked for and if it doesn't not match, you will receive one of several errors.

6.References

- https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/
- https://www.mulesoft.com/resources/api/what-is-an-api
- https://rapidapi.com/blog/types-of-apis/
- https://en.wikipedia.org/wiki/API
- https://whatis.techtarget.com/definition/synchronous-asynchronous-API
- https://www.mydbsync.com/blogs/soap-vs-rest-api-a-comparative-analysis/
- https://www.programmableweb.com/wp-content/protocols34.png
- https://csharpcorner.azureedge.net/article/analysis-of-rest-based-networking/Images/image001.png
- https://rapidapi.com/blog/api-glossary/api-key/
- https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/