API Testing using Postman Tutorial

Learn Automation Testing from Scratch

API Testing using Postman – Session 1

Trainer – Haradhan Pal

Agenda

- What is Client, Server and Host?
- Client-Server Model & Architecture
- Web Services & it’s Components
- HTTP vs HTTPS
- What is URI and URL?
- What is API Testing?
- API Testing Process
- How to approach API testing?
- Benefits of API Testing
- Why is API Testing important?
- Types of API Testing
- API Testing Best Practices
- API Testing Tools
- Types of Bugs that API testing detects
- Challenges of API Testing

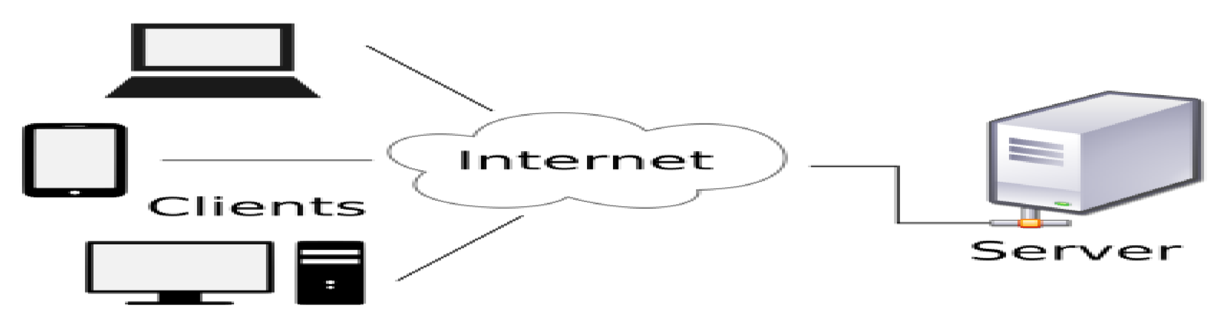
What is Client, Server and Host?

What is a Client?

A client is a computer hardware device or software that accesses a service made available by a server. The server is often (but not always) located on a separate physical computer.

What is a Server?

A server is a physical computer dedicated to run services to serve the needs of other computers. Depending on the service that is running, it could be a file server, database server, home media server, print server, or web server.



What is a Host?

A host is a computer, connected to other computers for which it provides data or services over a network. In theory, every computer connected to a network acts as a host to other peers on the network. In essence, a host reflects the logical relationship of two or more computers on a network.

Let's say, user want to download an image from another computer from his network. That computer is "hosting" the image and therefore, it is the host computer. On the other hand, if that same computer downloads an image from user computer, then user computer becomes the host computer.

User computer can be a host to other computers. Likewise, user router can be a host to other routers. But a host must have an assigned IP address. Therefore, modems, hubs, and switches are not considered hosts because they do not have assigned IP addresses.

Client-Server Model

What is a hostname and host ID?

The hostname is the name of the computer.

The host ID is the physical address (the MAC address of the Network Interface Controller).

What is the Client-Server Model?

The client-server model, or client-server architecture, is a distributed application framework dividing tasks between servers and clients, which either reside in the same system or communicate through a computer network or the Internet. The client relies on sending a request to another program in order to access a service made available by a server. The server runs one or more programs that share resources with and distribute work among clients.

The client server relationship communicates in a request–response messaging pattern and must adhere to a common communications protocol, which formally defines the rules, language, and dialog patterns to be used. Client-server communication typically adheres to the TCP/IP protocol suite.

TCP protocol maintains a connection until the client and server have completed the message exchange. TCP protocol determines the best way to distribute application data into packets that networks can deliver, transfers packets to and receives packets from the network, and manages flow control and retransmission of dropped or garbled packets. IP is a connectionless protocol in which each packet traveling through the Internet is an independent unit of data unrelated to any other data units.

Client requests are organized and prioritized in a scheduling system, which helps servers cope in the instance of receiving requests from many distinct clients in a short space of time. The client-server approach enables any general-purpose computer to expand its capabilities by utilizing the shared resources of other hosts. Popular client-server applications include email, the World Wide Web, and network printing.

Client Server Architecture

There are four main categories of client-server computing:

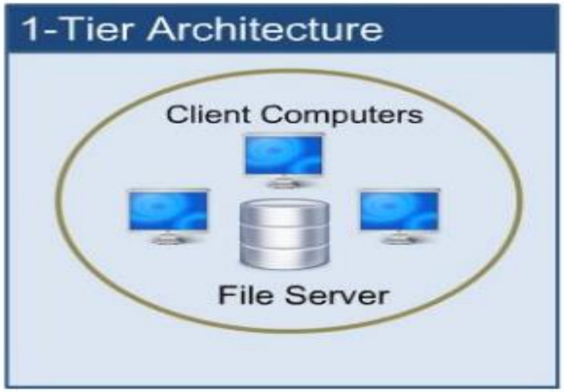
One-Tier architecture: where Data and Application resides in one machine. Presentation, Business, Data Access layers within a single software package. The data is usually stored in the local system or a shared drive. Application such as MS Office come under one-tier application.

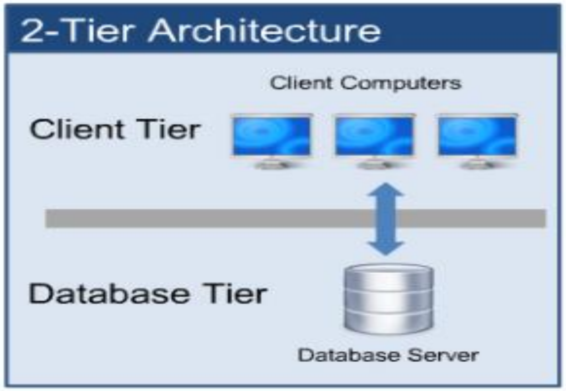
Two-Tier architecture: where Client resides at one system and database server is at another system. User can have multiple clients. For instance, online ticket reservations software use the two-tier architecture. The client-server GUI is written in high-level languages such as C++ and Java.

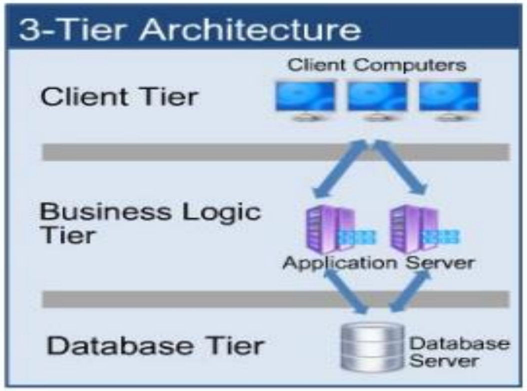
Three-Tier architecture: In 3 tier architecture, user need Internet. This is applicable mainly for web applications. We have client (Browser), DB server (where we store our data). In between client and server, there is business logic layer (which is called as middle layer).

So, we have 3 layers here:

1. Client layer/Client Server/Presentation Layer/UI (Front End)
2. Application layer/Application server/Web server (Business Logic)
3. Database Layer/Database server (Back End)







N-Tier architecture: divides an application into logical layers, which separate responsibilities and manage dependencies, and physical tiers, which run on separate machines, improve scalability, and add latency from the additional network communication. N-Tier architecture can be closed-layer, in which a layer can only communicate with the next layer down, or open-layer, in which a layer can communicate with any layers below it.

Web Services & it’s Components

Web Services: A Web services are any bit of services that makes it accessible over the Internet and normalizes its correspondence through XML encoding. A customer conjures web services by sending a solicitation (for the most part as an XML message), and the services send back an XML response. Web services summon communication over a network, with HTTP as the most widely recognized methods for the network between the two frameworks. Web services are equivalent to SOA (Services Oriented Architecture) and fundamentally depend on measures, for example, XML-RPC and SOAP (Simple Object Access Protocol).

Components: All the standard web services work using the following components.

SOAP (Simple Object Access Protocol)

UDDI (Universal Description, Discovery and Integration)

WSDL (Web Services Description Language)

Web APIs: API stands for Application Programming Interface. It is a collection of communication conventions and subroutines used by various programs to communicate between them. A developer can utilize different API apparatuses to make its program simpler and less complex. Likewise, an API encourages the developers with a proficient method to build up their product programs. Thus, in simple terms, an API determines how programming segments ought to associate with one another. It is a set of protocols and schedules, and its reactions are returned as JSON or XML in data. APIs can utilize any kind of communication convention and are not restricted similarly as a web service is.

KEY DIFFERENCE between API and Web Service:

Web service is a collection of open-source protocols and standards used for exchanging data between systems or applications whereas API is a software interface that allows two applications to interact with each other without any user involvement.

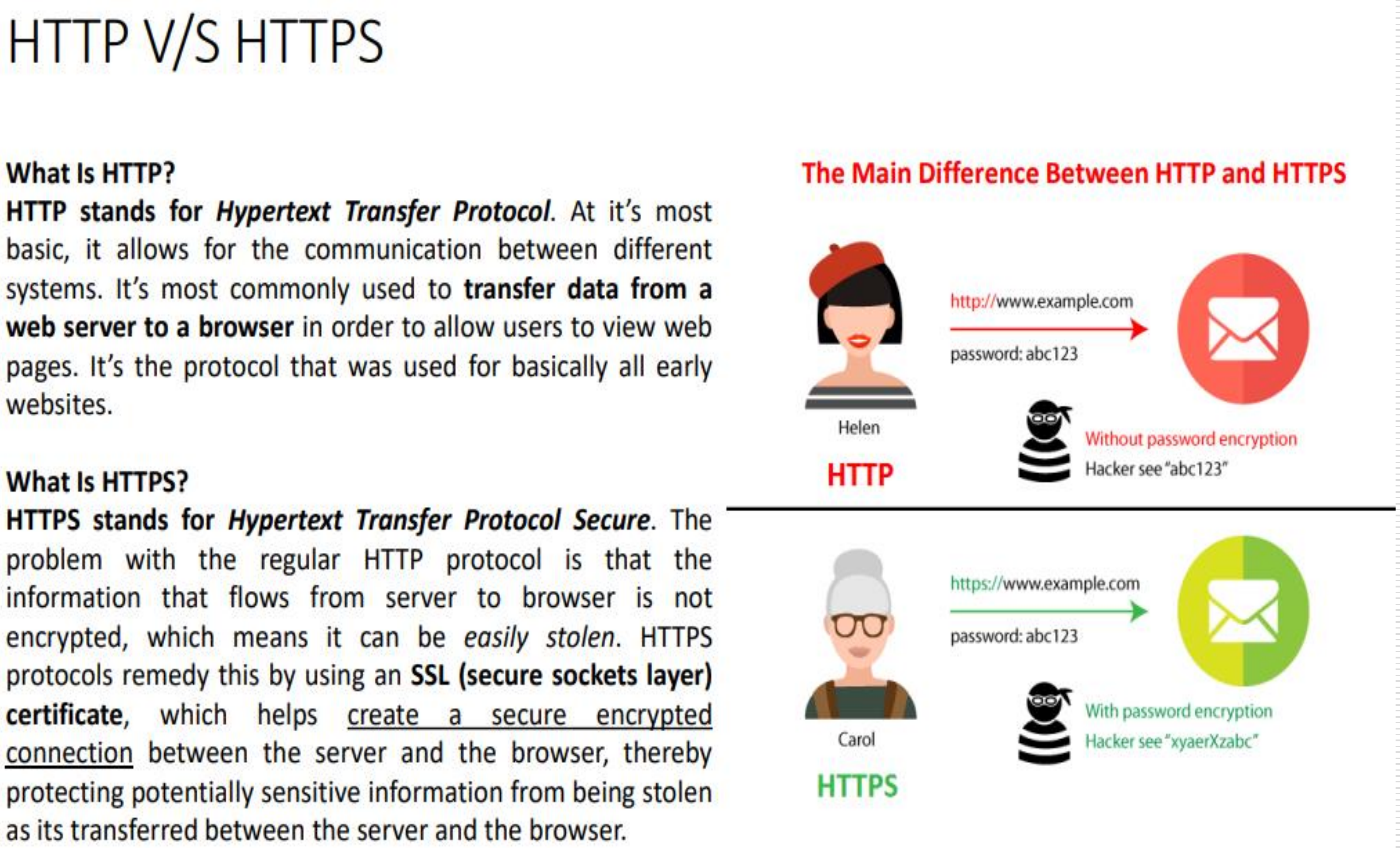
Web service is used for REST, SOAP and XML-RPC for communication while API is used for any style of communication.

Web service supports only HTTP protocol whereas API supports HTTP/HTTPS protocol.

Web service supports XML while API supports XML and JSON.

All Web services are APIs but all APIs are not web services.

HTTP vs HTTPS



HTTP V/S HTTPS

What Is HTTP?

HTTP stands for *Hypertext Transfer Protocol*. At it's most basic, it allows for the communication between different systems. It's most commonly used to **transfer data from a web server to a browser** in order to allow users to view web pages. It's the protocol that was used for basically all early websites.

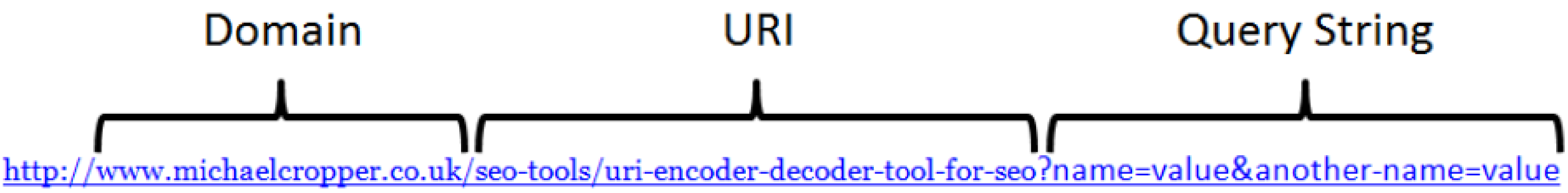
What Is HTTPS?

HTTPS stands for *Hypertext Transfer Protocol Secure*. The problem with the regular HTTP protocol is that the information that flows from server to browser is not encrypted, which means it can be *easily stolen*. HTTPS protocols remedy this by using an **SSL (secure sockets layer) certificate**, which helps create a secure encrypted connection between the server and the browser, thereby protecting potentially sensitive information from being stolen as its transferred between the server and the browser.

The Main Difference Between HTTP and HTTPS

What is URI and URL?

- Uniform Resource Identifier (URI) is a sequence of characters that distinguishes one resource from another.
- There are two types of URIs: URNs and URLs.
- Uniform Resource Name (URN) is a persistent and location-independent identifier
- Uniform Resource Locator (URL) is a specific type of identifier that not only identifies the resource but tells you how to access it or where it's located.
- The key difference between URIs and URLs is that URIs are identifiers, whereas URLs are locators. In other words, a URI simply identifies the resource. It does not describe or imply how to locate the resource. A URL does.
- The most common analogy used to understand the difference between URIs and URLs is comparing a person's name vs. their address. A person's name is like a URI because it identifies the person without providing any information on how to locate them. An address, however, identifies the person as resident of that address and provides their physical location. That's why it's like a URL.

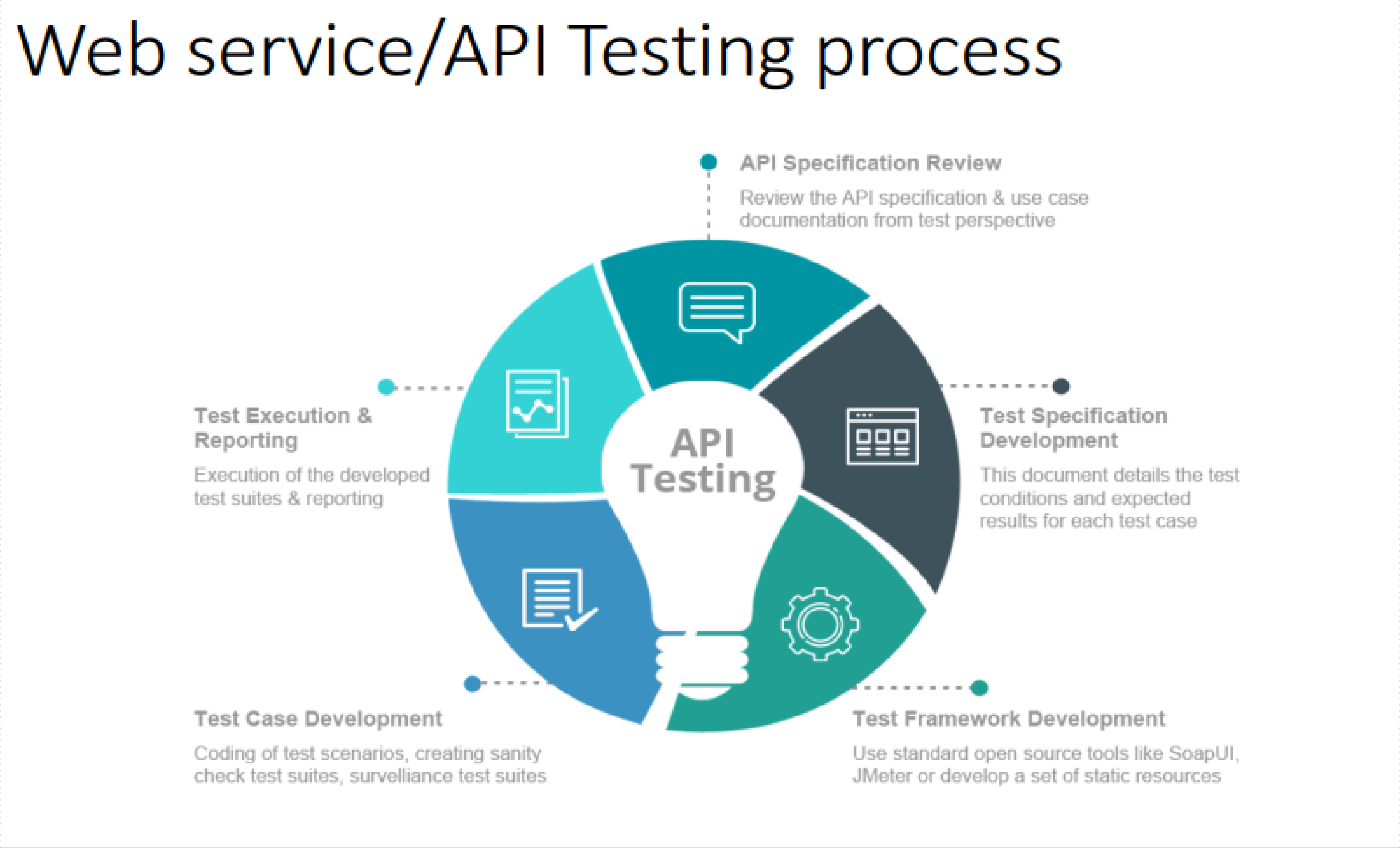

Domain
URI
Query String
http://www.michaelcropper.co.uk/seo-tools/uri-encoder-decoder-tool-for-seo?name=value&another-name=value

Domain:	The physical server where your website is hosted
URI:	The identifier which maps to files on your server
Query String:	Part of a GET request to easily pass in values to customise the output

What is API Testing?

- In software application development, API is the middle layer between the presentation (UI) and the database layer. APIs enable communication and data exchange from one software system to another.
- API testing is a software testing practice that tests the APIs directly — from their functionality, reliability, performance, to security. Part of integration testing, API testing effectively validates the logic of the build architecture within a short amount of time.
- Applications frequently have three layers: a data layer, a service layer -- the API layer -- and a presentation layer -- the user interface (UI) layer. The business logic of the application guides to how users can interact with the services, functions and data held within the app is in the API layer. API testing focuses on analyzing the business logic as well as the security of the application and data responses. An API test is generally performed by making requests to one or more API endpoints and comparing the response with expected results.
- API testing is a type of software testing that analyzes an application program interface (API) to verify it fulfills its expected functionality, security, performance and reliability. The tests are performed either directly on the API or as part of integration testing. An API is middleware code that enables two software programs to communicate with each other. The code also specifies the way an application requests services from the operating system (OS) or other applications.
- API testing is frequently automated and used by DevOps, quality assurance (QA) and development teams for continuous testing practices.

API Testing Process



Web service/API Testing process

API Specification Review

Review the API specification & use case documentation from test perspective

Test Specification Development

This document details the test conditions and expected results for each test case

Test Framework Development

Use standard open source tools like SoapUI, JMeter or develop a set of static resources

Test Case Development

Coding of test scenarios, creating sanity check test suites, survelliance test suites

Test Execution & Reporting

Execution of the developed test suites & reporting

How to approach API testing?

- An API testing process should begin with a clearly defined scope of the program as well as a full understanding of how the API is supposed to work. Some questions that testers should consider include:
  - What endpoints are available for testing?
  - What response codes are expected for successful requests?
  - What response codes are expected for unsuccessful requests?
  - Which error message is expected to appear in the body of an unsuccessful request?
- Once factors such as these are understood, testers can begin applying various testing techniques. Test cases should also be written for the API. These test cases define the conditions or variables under which testers can determine whether a specific system performs correctly and responds appropriately. Once the test cases have been specified, testers can perform them and compare the expected results to the actual results. The test should analyze responses that include:
  - reply time,
  - data quality,
  - confirmation of authorization,
  - HTTP status code and
  - error codes.
- API testing can analyze multiple endpoints, such as web services, databases or web user interfaces. Testers should watch for failures or unexpected inputs. Response time should be within an acceptable agreed-upon limit, and the API should be secured against potential attacks.
- Tests should also be constructed to ensure users can't affect the application in unexpected ways, that the API can handle the expected user load and that the API can work across multiple browsers and devices.
- The test should also analyze the results of nonfunctional tests as well, including performance and security.

Benefits of API Testing

Earlier Testing:

With API testing, once the logic is designed, tests can be built to validate the correctness in responses and data. We don't have to wait for various teams to finish their work or for full applications to be built - test cases are isolated and ready to built immediately.

Easier Test Maintenance:

UIs are constantly changing and moving around based on how they are accessed - browsers, devices, screen orientation, etc. This creates a nightmare scenario where tests are being constantly rewritten to keep up with the actual code in production. API changes are much more controlled and infrequent - often API definitions files like OpenAPI Spec can help make refactoring tests only a seconds of work.

Faster Time To Resolution:

When API tests fail, we know exactly where our system broke and where the defect can be found. This helps reduce time triaging bugs between builds, integrations, and even different team-members. The small, isolated footprint of an API test is perfect for faster MTTR stats, a valuable KPI for DevOps teams.

Speed and Coverage of Testing:

300 UI tests may take 30 hours to run. 300 API tests could be run in 3 minutes. That means user will find more bugs in less time, while also being about to fix them immediately.

Language-independent:

Data is exchanged via XML and JSON formats, so any language can be used for test automation. XML and JSON are typically structured data, making the verification fast and stable. There are also built-in libraries to support comparing data using these data formats.

GUI-independent:

API testing can be performed in the app prior to GUI testing. Early testing means early feedback and better team productivity. The app's core functionalities can be tested to expose small errors and to evaluate the build's strengths.

Improved test coverage:

Most API/web services have specifications, allowing you to create automated tests with high coverage — including functional testing and non-functional testing.

Why is API Testing important?

- User interface tests are often inefficient for validating API service functionality and often do not cover all the necessary aspects of back-end testing. This can result in bugs left within the server or unit levels -- a costly mistake that can greatly delay the product release and often requires large amounts of code to be rewritten.
- API testing allows developers to start testing early in the development cycle before the UI is ready. Any request that doesn't produce the appropriate value at the server layer will not display it on the UI layer. This enables developers to kill at least half of the existing bugs before they become more serious problems. It also enables testers to make requests that might not be possible through the UI -- a necessity for exposing security flaws.
- Many companies are using microservices for their software applications because they allow software to be deployed more efficiently. If one area of the app is being updated, the other areas can continue functioning without interruption. Each application section has a separate data store and different commands for interacting with that data store. Most microservices use APIs; therefore, as more business adopt the use of microservices, API testing will become increasingly necessary to ensure all parts are working correctly.
- API testing is also integral to Agile software development, in which instant feedback is necessary to the process flow. In Agile environments, unit tests and API tests are preferred over graphical user interface (GUI) tests because they are easy to maintain and more efficient. GUI tests often require intense reworking if they want to keep pace with the frequent changes in an Agile environment.
- Overall, incorporating API tests into the test-driven development process can benefit engineering and development teams across the entire development lifecycle. These benefits are then passed along to customers in the form of improved services and better-quality products.

Types of API Testing

API testing typically involves the following practices:

- Unit testing - Testing the functionality of individual operations.
- Validation Testing - It occurs among the final steps and plays an essential role in the development process. It verifies the aspects of product, behavior, and efficiency. In other words, validation testing can be seen as an assurance of the correct development.
- UI testing - It is defined as a test of the user interface for the API and other integral parts. UI testing focuses more on the interface which ties into the API rather than the API testing itself. Although UI testing is not a specific test of API in terms of codebase, this technique still provides an overview of the health, usability, and efficiency of the app’s front and back ends.
- Functional testing - Testing the functionality of broader scenarios, often using unit tests as building blocks for end-to-end tests. Includes test case definition, execution, validation, and regression testing.
- Load testing - Validating functionality and performance under load, often by reusing functional test cases.
- Runtime error detection - Monitoring an application the execution of automated or manual tests to expose problems such as race conditions, exceptions, and resource leaks.
- Security testing - Includes penetration testing and fuzz testing as well as validating authentication, encryption, and access control.
- Web UI testing - Performed as part of end-to-end integration tests that also cover APIs, enables teams to validate GUI items in the context of the larger transaction.
- Interoperability testing - (SOAP only) Checking conformance to Web Services Interoperability profiles.
- Penetration testing - Considered the second test in the auditing process. In this type, users with limited API knowledge will try to assess the threat vector from an outside perspective, which is about functions, resources, processes, or aim to the entire API and its components.
- Fuzz-testing - Massive amounts of purely random data, sometimes referred to as "noise" or "fuzz", is forcibly input into the system in order to attempt a forced crash, overflow, or other negative behavior. This is done to test the API at its absolute limits and serves somewhat as a "worst case scenario".

API Testing Best Practices

- API Test cases should be grouped by test category
- On top of each test, user should include the declarations of the APIs being called.
- Parameters selection should be explicitly mentioned in the test case itself
- Prioritize API function calls so that it will be easy for testers to test
- Each test case should be as self-contained and independent from dependencies as possible
- Avoid "test chaining" in test script development
- Special care must be taken while handling one-time call functions like – Delete, CloseWindow etc.
- Call sequencing should be performed and well planned
- To ensure complete test coverage, create API test cases for all possible input combinations of the API.

API Testing Tools

- When performing an API test, developers can either write their own framework or choose from a variety of ready-to-use API testing tools. Designing an API test framework enables developers to customize the test; they are not limited to the capabilities of a specific tool and its plugins. Testers can add whichever library they consider appropriate for their chosen coding platform, build unique and convenient reporting standards and incorporate complicated logic into the tests. However, testers need sophisticated coding skills if they choose to design their own framework.
- Conversely, API testing tools provide user-friendly interfaces with minimal coding requirements that enable less-experienced developers to feasibly deploy the tests. Unfortunately, the tools are often designed to analyze general API issues and problems more specific to the tester's API can go unnoticed.
- A large variety of API testing tools is available, ranging from paid subscription tools to open-source offerings. Some specific examples of API testing tools include:
- SoapUI: The tool focuses on testing API functionality in SOAP and REST APIs and web services.
- Apache Jmeter: An open-source tool for load and functional API testing.
- Apigee: A cloud API testing tool from Google that focuses on API performance testing.
- REST Assured: An open source, Java-specific language that facilitates and eases the testing of REST APIs.
- Postman: A Google chrome app used for verifying and automating API testing.
- Katalon: An open-source application that helps with UI automated testing.
- Swagger UI: An open-source tool that creates a webpage that documents APIs used.

Types of Bugs that API testing detects

- Fails to handle error conditions gracefully
- Unused flags
- Missing or duplicate functionality
- Reliability Issues. Difficulty in connecting and getting a response from API.
- Security Issues
- Multi-threading issues
- Performance Issues. API response time is very high.
- Improper errors/warning to a caller
- Incorrect handling of valid argument values
- Response Data is not structured correctly (JSON or XML)

Challenges of API Testing

- Main challenges in Web API testing is Parameter Combination, Parameter Selection, and Call Sequencing
- There is no GUI available to test the application which makes difficult to give input values
- Validating and Verifying the output in a different system is little difficult for testers
- Parameters selection and categorization is required to be known to the testers
- Exception handling function needs to be tested
- Coding knowledge is necessary for testers