Chapter 1 Notes Introduction to Internet Programming

1.1 Introduction to Web Development

Web application development is the important aspect in the field of Information Technologies as many of the large scale business runs completely on the web sites now a days, amazon, flipkart, bbc, ICICI, HDFC, government sites etc are the examples. As the number of users grows the challenges in the web application development for the development point of view increases with time. Traditional we development frameworks heavily depends on the 2 tier client server based model. With the advancement in the web development technologies new frameworks getting added every year. The framework like React, Angular, Flask, Django, MS Dot net are the few examples of modern web development framework.

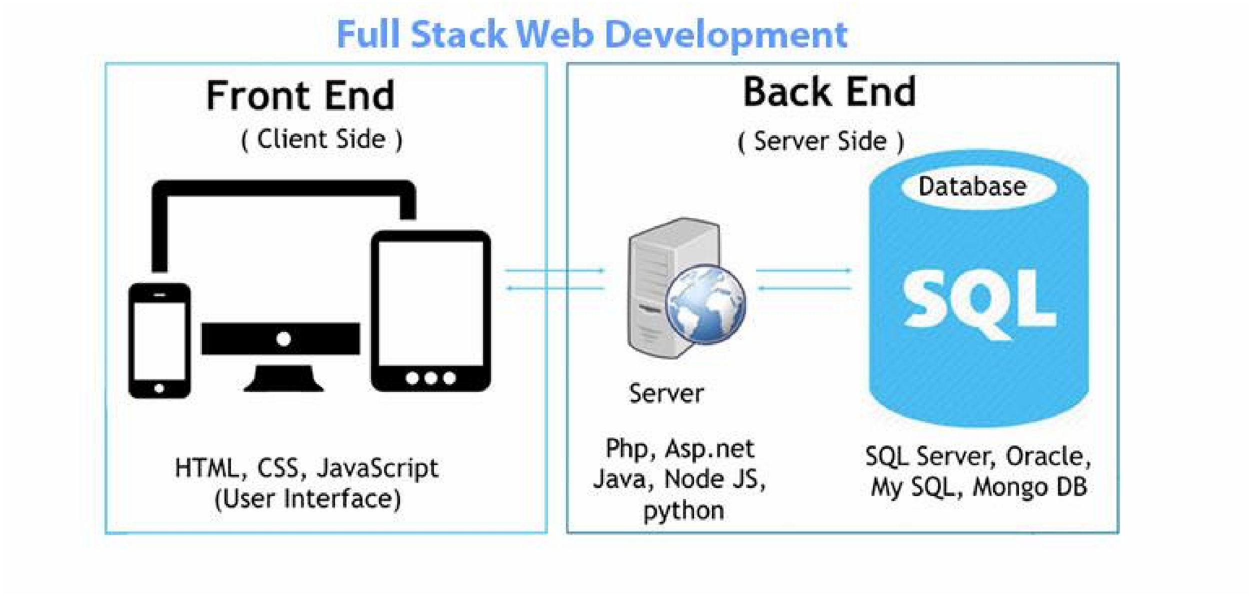


Figure 1 Full Stack Web Development

1.2 Theory behind Web Application Development

The main aspect to understand some basics traditional 2 tier client server, how it works and then 3 –tier followed by n-tier architecture. Some of the concepts like http, https, dns, TLS, web server, XML, Json, Rest API etc should be known before actually starts actual web application development process,

Client Server 2-Tier Approach- 2-tier approach are considered as traditional approach in which server perform all the task and client simply send the request and wait for the response, its servers responsibility to read request, process request and send response back to the client. Here server need to do lot of work and in some cased its time consuming specially for scalable web sites.

The web development using simple HTML 4/5, CSS, Traditional with Apache Web Server and PHP script can be considered as traditional 2 tier web development approach; however the same technologies used in the modern web application framework as well. The PHP is one of the popular server side scripting language still exist with one of the popular server side scripting language.

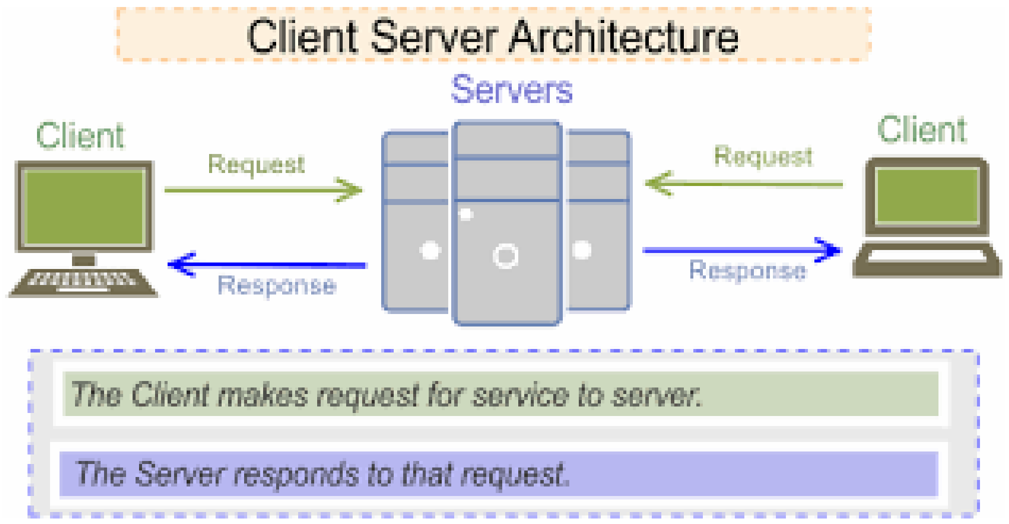

Client Server Architecture
Servers
Client
Request
Response
Client
Request
Response
The Client makes request for service to server.
The Server responds to that request.

Figure 1.2 Typical Client Server Architecture

The 3 tier architecture includes database server as separated from the web server.

1.3 Web Server and Application Server

The web server is the server machine refereeing as physical server machine which act as a server, technically its a special configuration set of machines which is imaging as large set of cluster machine acts as a web server.

The web application server is the software which typically runs on web server machine for example Apache Server is the software acts like a server, Apache server is the software running on the port number 80, however the from the configuration file this port number can be changed.

Note- While performing the practical's several time in the CC lab apache web server not starting (We are using XAMP Server) reason may be port no 80 or 443 block by other programs.

To solve this issue we can change the config. file of apache server and some setting so that we able to start apache web server in CC lab.

The main difference between web server and Application server is Application server process the request in software form and generate the http response and then send the response back to the client. The other examples of web application server are Microsoft IIS, Oracle Glassfish etc.

1.4 HTTP (Hyper Text Transport Protocol)

HTTP protocol is used in the in the internet communication, technically client send the request to the server in the http, means the http packet is send from client to the server, all the details are written in the http header and payload (As discussed during Lab Practical's)

Http protocol is responsible to communication between client and the server,

Http designed to transfer web pages using web browser

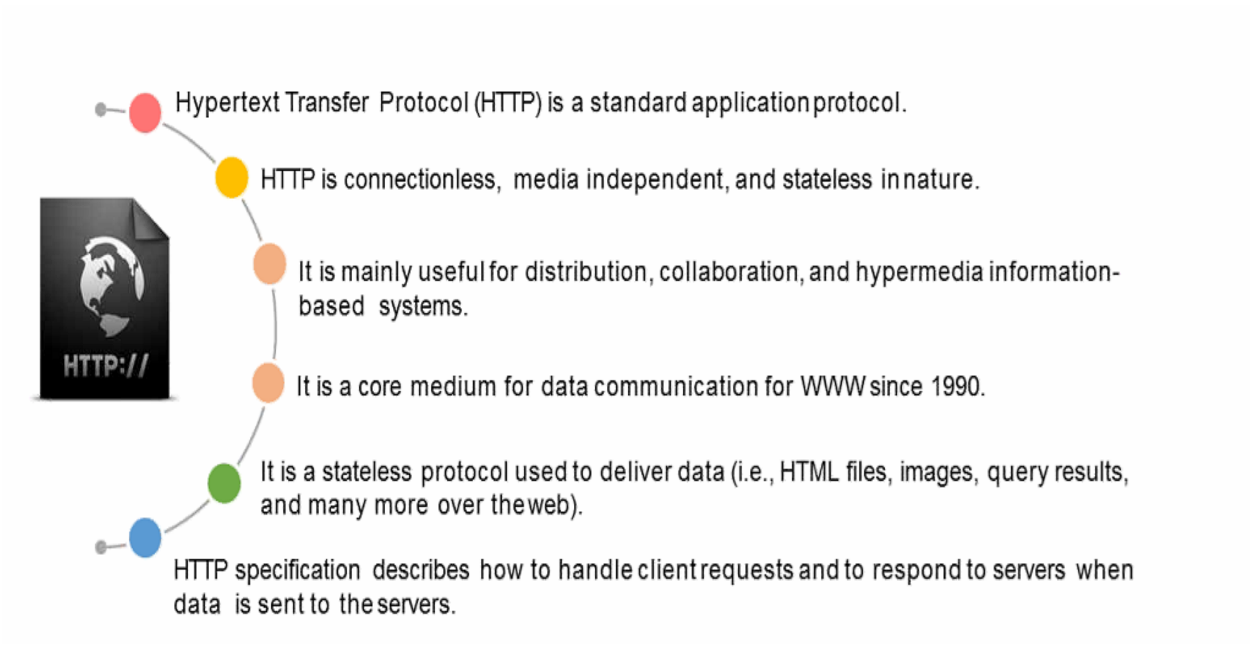

Hypertext Transfer Protocol (HTTP) is a standard application protocol.
HTTP is connectionless, media independent, and stateless in nature.
It is mainly useful for distribution, collaboration, and hypermedia information-based systems.
It is a core medium for data communication for WWW since 1990.
It is a stateless protocol used to deliver data (i.e., HTML files, images, query results, and many more over the web).
HTTP specification describes how to handle client requests and to respond to servers when data is sent to the servers.
HTTP://

Figure 1.3 HTTP Protocol

The Http protocol is very important protocol as all the communication happens through the http only, as a application server point of view generating http response based on the incoming request is the most important task, and since its stateless the maintaining states if required is the challenges which was address in technologies like AJAX or using JSON etc.

Http Architecture

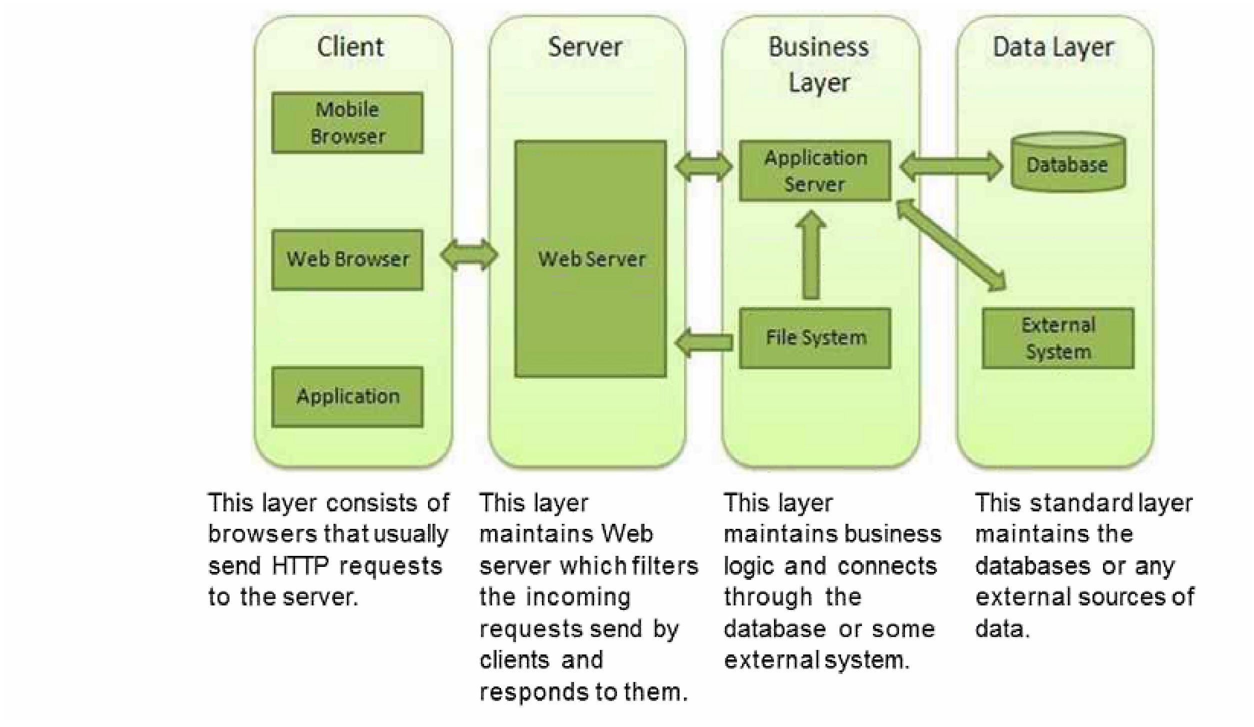

Client
Mobile Browser
Web Browser
Application
Server
Web Server
Business Layer
Application Server
File System
Data Layer
Database
External System
This layer consists of browsers that usually send HTTP requests to the server.
This layer maintains Web server which filters the incoming requests send by clients and responds to them.
This layer maintains business logic and connects through the database or some external system.
This standard layer maintains the databases or any external sources of data.

Figure 1.4 HTTP Architecture

What does a typical HTTP request look like?

An HTTP request is just a series of lines of text that follow the HTTP protocol. A GET request might look like this:

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.63.0 libcurl/7.63.0 OpenSSL/1.1.l zlib/1.2.11
Host: www.example.com
Accept-Language: en
```

This section of text, generated by the user's browser, gets sent across the Internet. The problem is, it's sent just like this, in plaintext that anyone monitoring the connection can read. (Those who are unfamiliar with the HTTP protocol may find this text hard to understand, but anyone with a baseline knowledge of the protocol's commands and syntax can read it easily.)

This is especially an issue when users submit sensitive data via a website or a web application. This could be a password, a credit card number, or any other data entered into a form, and in HTTP all this data is sent in plaintext for anyone to read. (When a user submits a form, the browser translates this into an HTTP POST request instead of an HTTP GET request.)

When an origin server receives an HTTP request, it sends an HTTP response, which is similar:

```
HTTP/1.1 200 OK
Date: Wed, 30 Jan 2019 12:14:39 GMT
Server: Apache
Last-Modified: Mon, 28 Jan 2019 11:17:01 GMT
Accept-Ranges: bytes
Content-Length: 12
Vary: Accept-Encoding
Content-Type: text/plain

Hello World!
```

If a website uses HTTP instead of HTTPS, all requests and responses can be read by anyone who is monitoring the session. Essentially, a malicious actor can just read the text in the request or the response and know exactly what information someone is asking for, sending, or receiving.

What is HTTPS?

The S in HTTPS stands for "secure." HTTPS uses TLS (or SSL) to encrypt HTTP requests and responses, so in the example above, instead of the text, an attacker would see a bunch of seemingly random characters.

HTTPS stands for Hyper Text Transport Protocol SSL (Secure Software layer) used to secure http communication.

In https the exchange of information in the secure way in other words the data communicate using encryption.

SO https communication is used where the data need to secure in the web applications like transactions, banking and where security need to maintain at very high priority.

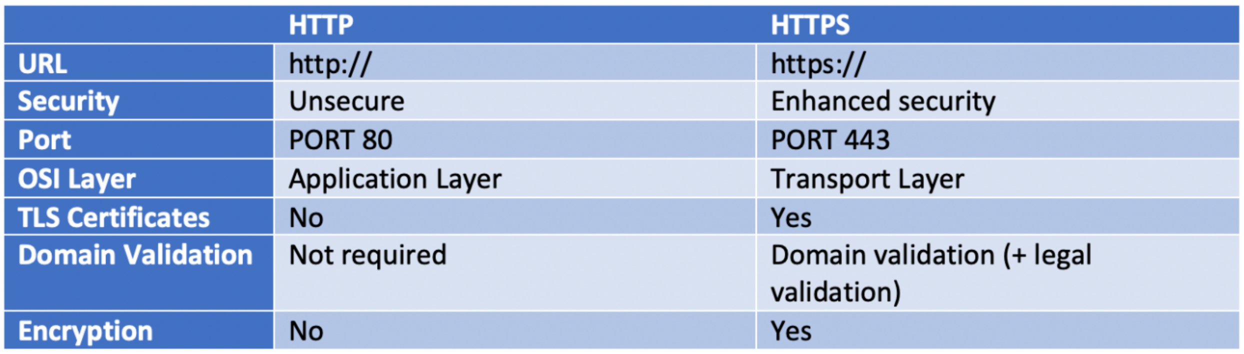
Instead of:

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.63.0 libcurl/7.63.0 OpenSSL/1.1.l zlib/1.2.11
Host: www.example.com
Accept-Language: en
```

The attacker sees something like:

```
t8Fw6T8UV81pQfyhDkhebbz7+oiwldr1j2gHBB3L3RFTRsQCpaSnSBZ78Vme+DpDVJPvZdZUZHpzbbcqmS
W1+3
```

Difference between HTTP and HTTPS-



	HTTP	HTTPS
URL	http://	https://
Security	Unsecure	Enhanced security
Port	PORT 80	PORT 443
OSI Layer	Application Layer	Transport Layer
TLS Certificates	No	Yes
Domain Validation	Not required	Domain validation (+ legal validation)
Encryption	No	Yes

Figure 1.5 Difference Between HTTP vs HTTPS

1.5 DNS (Domain Name System)

In the internet communication end user from client side uses URL (Uniform Resource Locator) address to access any web site, for example if user wants to open google page uses type

http://www.google.com on the web browsers address bar, actually system is not identify google web page with www.google.com , actually internet system uses IP Addresses.

The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources.

Each device connected to the Internet has a unique IP address which other machines use to find the device. DNS servers eliminate the need for humans to memorize IP addresses such as 192.168.1.1 (in IPv4), or more complex newer alphanumeric IP addresses such as 2400:cb00:2048:1::c629:d7a2 (in IPv6).

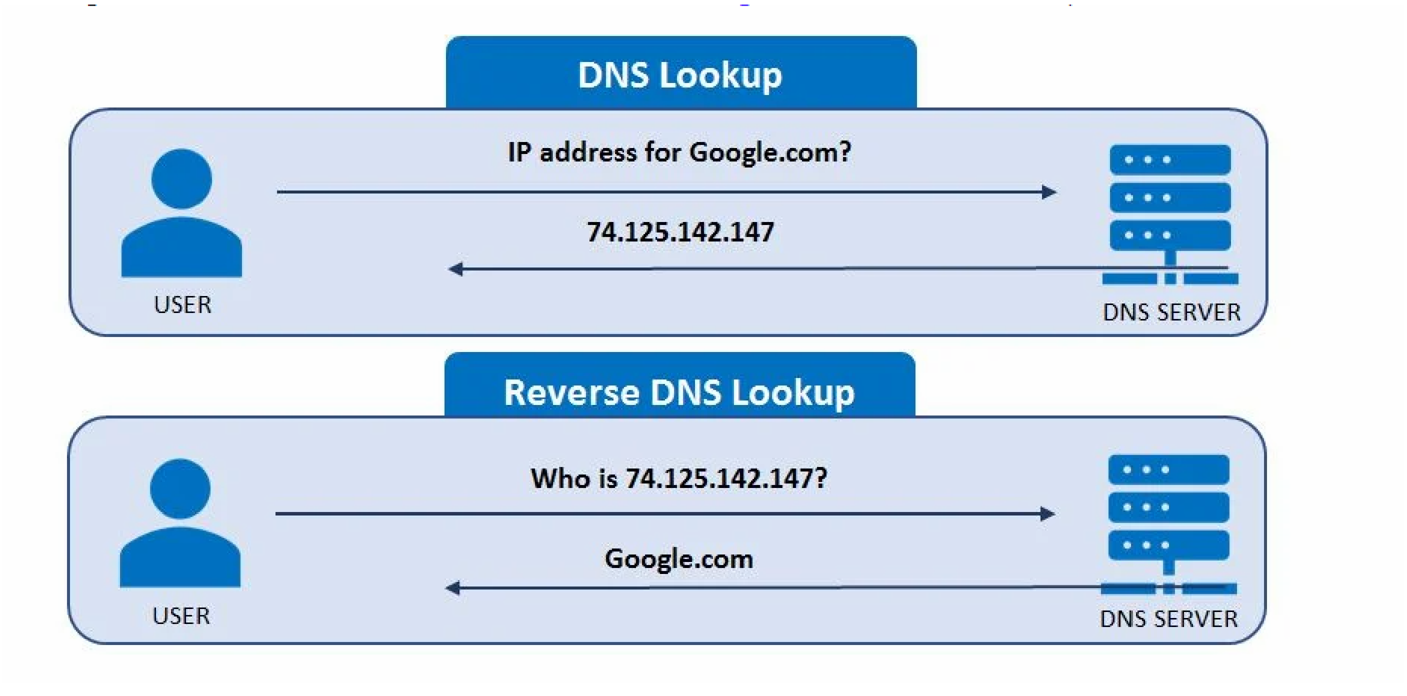


Figure 1.6 DNS Lookup

What are the steps in a DNS lookup?

For most situations, DNS is concerned with a domain name being translated into the appropriate IP address. To learn how this process works, it helps to follow the path of a DNS lookup as it travels from a web browser, through the DNS lookup process, and back again. Let's take a look at the steps.

Note: Often DNS lookup information will be cached either locally inside the querying computer or remotely in the DNS infrastructure. There are typically 8 steps in a DNS lookup. When DNS information is cached, steps are skipped from the DNS lookup process which makes it quicker. The example below outlines all 8 steps when nothing is cached.

The 8 steps in a DNS lookup:

1. A user types ‘example.com’ into a web browser and the query travels into the Internet and is received by a DNS recursive resolver.
2. The resolver then queries a DNS root nameserver (.).
3. The root server then responds to the resolver with the address of a Top Level Domain (TLD) DNS server (such as .com or .net), which stores the information for its domains. When searching for example.com, our request is pointed toward the .com TLD.
4. The resolver then makes a request to the .com TLD.
5. The TLD server then responds with the IP address of the domain’s nameserver, example.com.
6. Lastly, the recursive resolver sends a query to the domain’s nameserver.
7. The IP address for example.com is then returned to the resolver from the nameserver.
8. The DNS resolver then responds to the web browser with the IP address of the domain requested initially.

Once the 8 steps of the DNS lookup have returned the IP address for example.com, the browser is able to make the request for the web page:

9. The browser makes a HTTP request to the IP address.
10. The server at that IP returns the webpage to be rendered in the browser (step 10).

1.6 What is Transport Layer Security (TLS)?

Transport Layer Security, or TLS, is a widely adopted security protocol designed to facilitate privacy and data security for communications over the Internet. A primary use case of TLS is encrypting the communication between web applications and servers, such as web browsers loading a website. TLS can also be used to encrypt other communications such as email, messaging, and voice over IP (VoIP). In this article we will focus on the role of TLS in web application security.

TLS was proposed by the Internet Engineering Task Force (IETF), an international standards organization, and the first version of the protocol was published in 1999. The most recent version is TLS 1.3, which was published in 2018.

What is the difference between TLS and SSL?

TLS evolved from a previous encryption protocol called Secure Sockets Layer (SSL), which was developed by Netscape. TLS version 1.0 actually began development as SSL version 3.1, but the name of the protocol was changed before publication in order to indicate that it was no longer associated with Netscape. Because of this history, the terms TLS and SSL are sometimes used interchangeably.

What is the difference between TLS and HTTPS?

HTTPS is an implementation of TLS encryption on top of the HTTP protocol, which is used by all websites as well as some other web services. Any website that uses HTTPS is therefore employing TLS encryption.

Why should businesses and web applications use the TLS protocol?

TLS encryption can help protect web applications from data breaches and other attacks. Today, TLS-protected HTTPS is a standard practice for websites. The Google Chrome browser gradually cracked down on non-HTTPS sites, and other browsers have followed suit. Everyday Internet users are more wary of websites that do not feature the HTTPS padlock icon.

What does TLS do?

There are three main components to what the TLS protocol accomplishes: Encryption, Authentication, and Integrity.

- Encryption: hides the data being transferred from third parties.
- Authentication: ensures that the parties exchanging information are who they claim to be.
- Integrity: verifies that the data has not been forged or tampered with.

How does TLS work?

For a website or application to use TLS, it must have a TLS certificate installed on its origin server (the certificate is also known as an "SSL certificate" because of the naming confusion described above). A TLS certificate is issued by a certificate authority to the person or business that owns a domain. The certificate contains important information about who owns the domain, along with the server's public key, both of which are important for validating the server's identity.

A TLS connection is initiated using a sequence known as the TLS handshake. When a user navigates to a website that uses TLS, the TLS handshake begins between the user's device (also known as the client device) and the web server.

During the TLS handshake, the user's device and the web server:

- Specify which version of TLS (TLS 1.0, 1.2, 1.3, etc.) they will use
- Decide on which cipher suites (see below) they will use
- Authenticate the identity of the server using the server's TLS certificate
- Generate session keys for encrypting messages between them after the handshake is complete

The TLS handshake establishes a cipher suite for each communication session. The cipher suite is a set of algorithms that specifies details such as which shared encryption keys, or session keys, will be used for that particular session. TLS is able to set the matching session keys over an unencrypted channel thanks to a technology known as public key cryptography.

The handshake also handles authentication, which usually consists of the server proving its identity to the client. This is done using public keys. Public keys are encryption keys that use one-way encryption, meaning that anyone with the public key can unscramble the data encrypted with the server's private key to ensure its authenticity, but only the original sender can encrypt data with the private key. The server's public key is part of its TLS certificate.

Once data is encrypted and authenticated, it is then signed with a message authentication code (MAC). The recipient can then verify the MAC to ensure the integrity of the data. This is kind of like the tamper-proof foil found on a bottle of aspirin; the consumer knows no one has tampered with their medicine because the foil is intact when they purchase it.

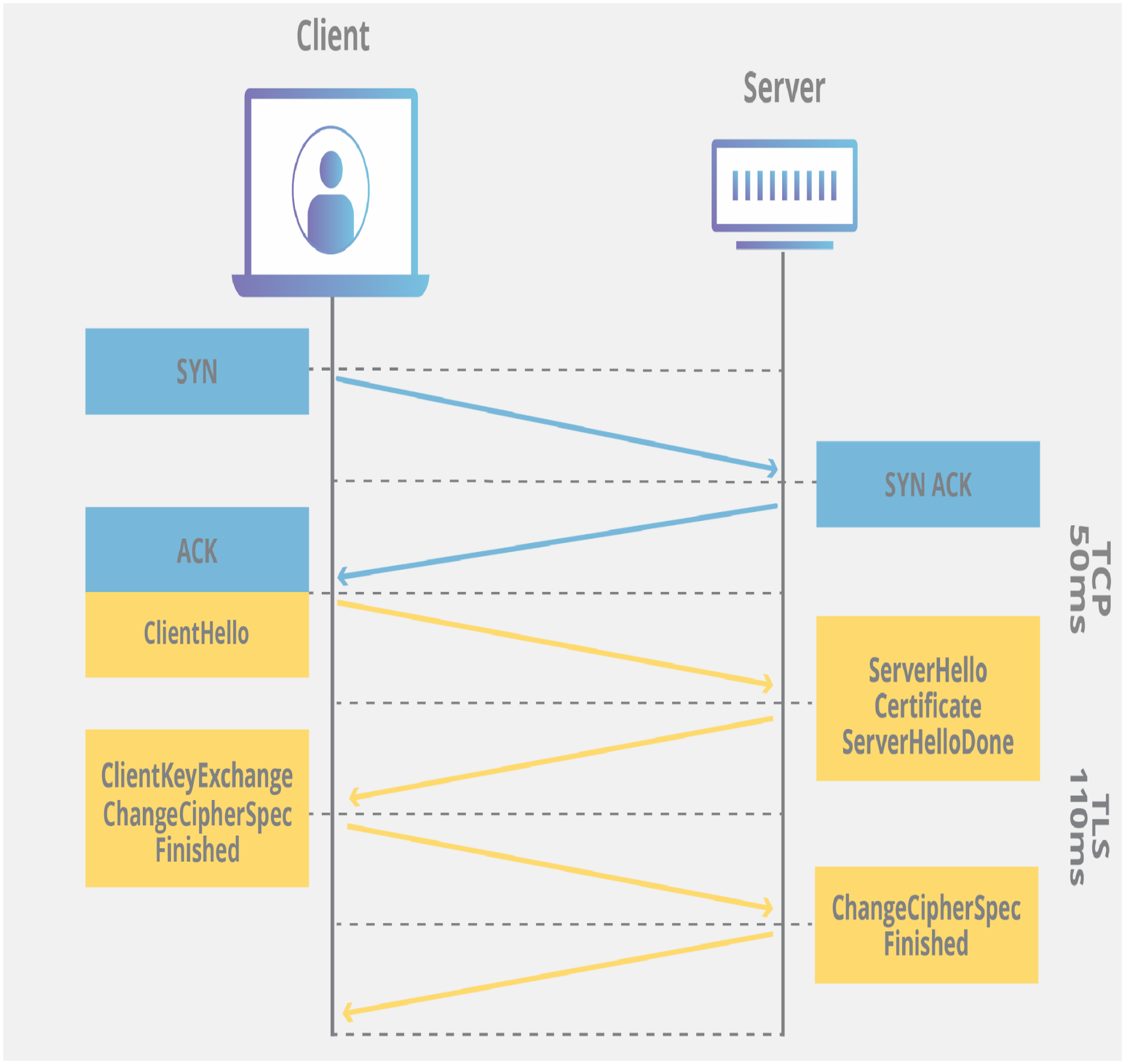


Figure 1.7 TLS

How does TLS affect web application performance?

The latest versions of TLS hardly impact web application performance at all.

Because of the complex process involved in setting up a TLS connection, some load time and computational power must be expended. The client and server must communicate back and forth several times before any data is transmitted, and that eats up precious milliseconds of load times for web applications, as well as some memory for both the client and the server.

However, there are technologies in place that help to mitigate potential latency created by the TLS handshake. One is TLS False Start, which lets the server and client start transmitting data before the TLS handshake is complete. Another technology to speed up TLS is TLS Session Resumption, which allows clients and servers that have previously communicated to use an abbreviated handshake.

These improvements have helped to make TLS a very fast protocol that should not noticeably affect load times. As for the computational costs associated with TLS, they are mostly negligible by today's standards.

TLS 1.3, released in 2018, has made TLS even faster. TLS handshakes in TLS 1.3 only require one round trip (or back-and-forth communication) instead of two, shortening the process by a few milliseconds. When the user has connected to a website before, the TLS handshake has zero round trips, speeding it up still further.

1.7 XML

XML is widely used in the era of web development. It is also used to simplify data storage and data sharing.

XML is a simple text-based language which was designed to store and transport data in plain text format. It stands for Extensible Markup Language. Following are some of the salient features of XML.

- XML is a markup language.
- XML is a tag based language like HTML.
- XML tags are not predefined like HTML.
- You can define your own tags which is why it is called extensible language.
- XML tags are designed to be self-descriptive.
- XML is W3C Recommendation for data storage and data transfer.

The main features or advantages of XML are given below.

1) XML separates data from HTML

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes.

With XML, data can be stored in separate XML files. This way you can focus on using HTML/CSS for display and layout, and be sure that changes in the underlying data will not require any changes to the HTML.

With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

2) XML simplifies data sharing

In the real world, computer systems and databases contain data in incompatible formats.

XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data.

This makes it much easier to create data that can be shared by different applications.

3) XML simplifies data transport

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

XML parser

XML parsing is the process of reading an XML document and providing an interface to the user application for accessing the document.

XML parsers check the well-formedness of the XML document and many can also validate the document with respect to a DTD (Document Type Definition) or XML schema.

Example to parse XML from java Script

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
var parser, xmlDoc;
var text = "<bookstore><book>" +
"<title>Java Programming</title>" +
"<author>Dr. Ashfaq Shaikh </author>" +
"<year>2023</year>" +
"</book></bookstore>";
parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");
document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
</script>
</body>
</html>
```

1.8 JSON

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data-interchange format
- JSON is plain text written in JavaScript object notation
- JSON is used to send data between computers
- JSON is language independent.

JSON Syntax Rules

JSON syntax is derived from JavaScript object notation syntax:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays
- JSON Data - A Name and a Value
- JSON data is written as name/value pairs (aka key/value pairs).

JSON Parse Example

```
<!DOCTYPE html>
<html>
<body>
<h2>Creating an Object from a JSON String</h2>
<p id="demo"></p>
<script>
const txt = '{"name":"John", "age":30, "city":"New York"}'
const obj = JSON.parse(txt);
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;
</script>
</body>
</html>
```

JSON V/S XML

JSON is Like XML Because

- Both JSON and XML are "self describing" (human readable)
- Both JSON and XML are hierarchical (values within values)
- Both JSON and XML can be parsed and used by lots of programming languages
- Both JSON and XML can be fetched with an XMLHttpRequest

JSON is Unlike XML Because

- JSON doesn't use end tag

- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays

The biggest difference is:

XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

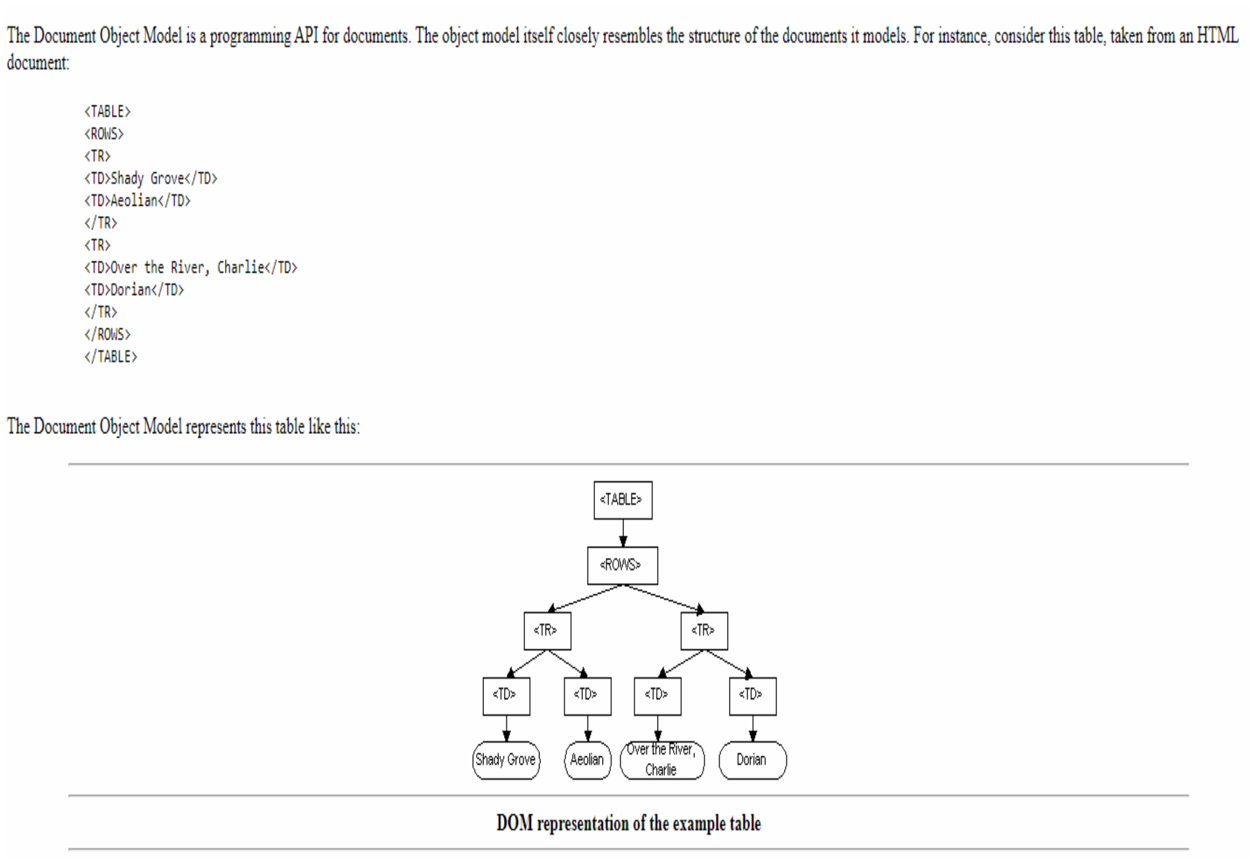
Why JSON is Better Than XML

- XML is much more difficult to parse than JSON.
- JSON is parsed into a ready-to-use JavaScript object.

For AJAX applications, JSON is faster and easier than XML:

1.9 What is the Document Object Model?

- The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.
- With the Document Object Model, programmers can create and build documents, navigate their structure, and add, modify, or delete elements and content.
- As a W3C specification, one important objective for the Document Object Model is to provide a standard programming interface that can be used in a wide variety of environments and applications. The Document Object Model can be used with any programming language



The Document Object Model is a programming API for documents. The object model itself closely resembles the structure of the documents it models. For instance, consider this table, taken from an HTML document:

```
<TABLE>
<ROWS>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</ROWS>
</TABLE>
```

The Document Object Model represents this table like this:

DOM representation of the example table

Figure 1.8 DOM Example

1.10 URL and URI

URL (Uniform Resource Locator):

URL (Uniform Resource Locator) is often defined as a string of characters that is directed to an address. It is a very commonly used way to locate resources on the web. It provides a way to retrieve the presentation of the physical location by describing its network location or primary access mechanism.

The protocol is described within the URL which is employed to retrieve the resource and resource name. The URL contains http/https at the start if the resource may be a web type resource. Similarly, it begins with ftp if the resource may be a file and mailto if the resource is an email address. The syntax of an URL is shown below where the primary part is employed for protocol and the remainder of the part is employed for the resource which consists of a website name or program name.

URI (Uniform Resource Identifier):

Similar to URL, URI (Uniform Resource Identifier) is also a string of characters that identifies a resource on the web either by using location, name or both. It allows uniform identification of the resources. A URI is additionally grouped as a locator, a name or both which suggests it can describe a URL, URN or both. The term identifier within the URI refers to the prominence of the resources, despite the technique used.

The former category in URI is URL, during which a protocol is employed to specify the accessing method of the resource and resource name is additionally laid out in the URL. A URL may be a non-persistent sort of the URI. A URN is required to exist globally unique and features a global scope.

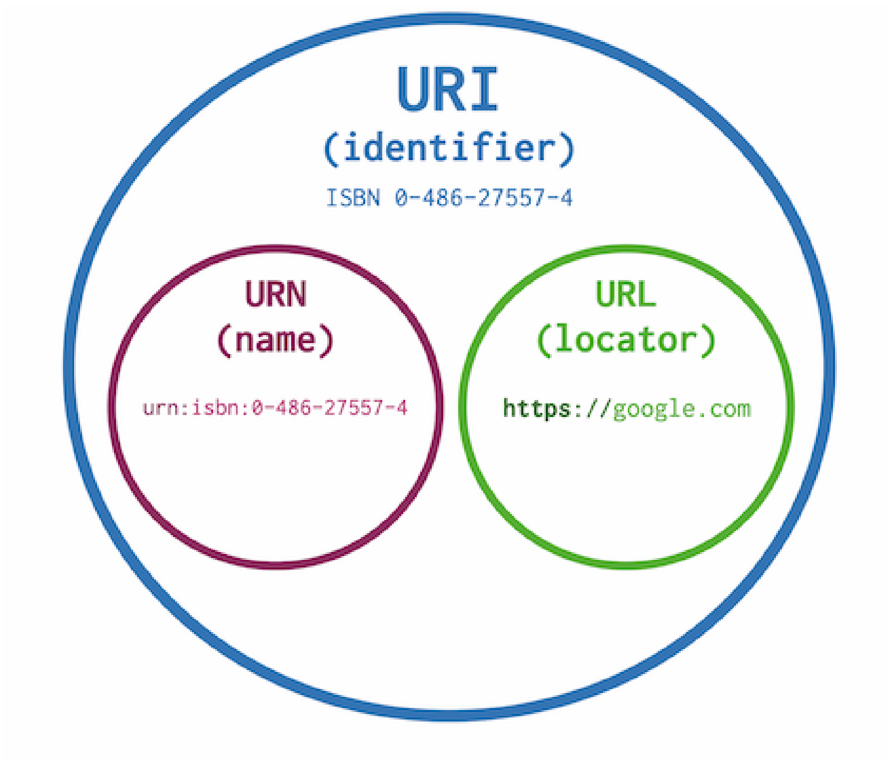


Figure 1.9 URL URI and URN

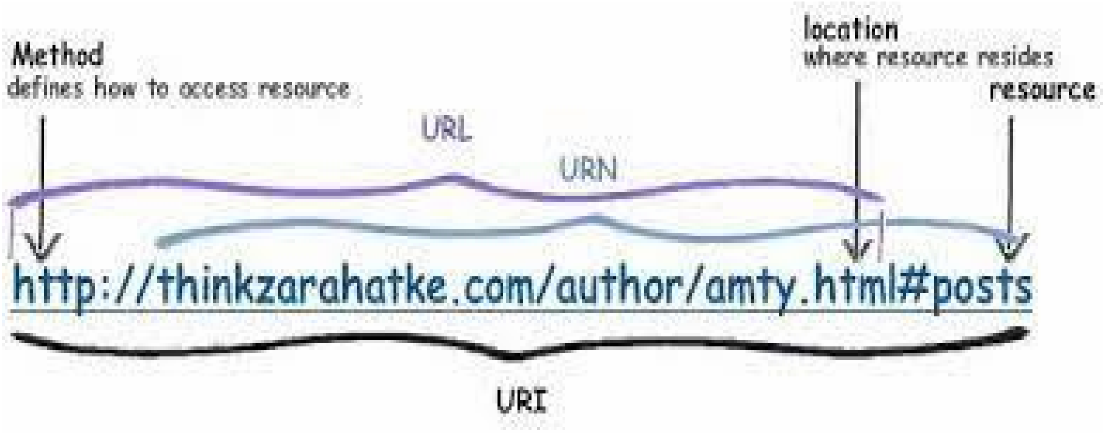
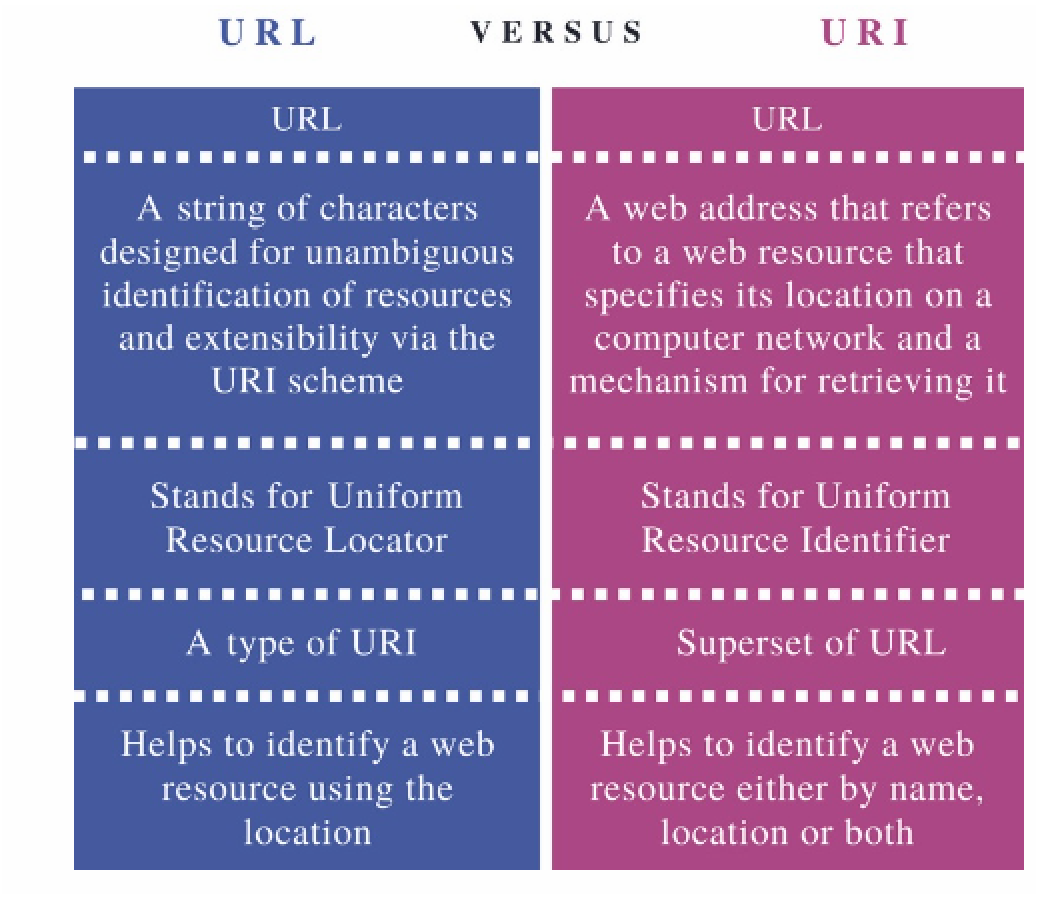

Method
defines how to access resource
location
where resource resides
resource
URL
URN
http://thinkzarahatke.com/author/amty.html#posts
URI

Figure 1.10 URL and URL Example



URL	URL
A string of characters designed for unambiguous identification of resources and extensibility via the URI scheme	A web address that refers to a web resource that specifies its location on a computer network and a mechanism for retrieving it
Stands for Uniform Resource Locator	Stands for Uniform Resource Identifier
A type of URI	Superset of URL
Helps to identify a web resource using the location	Helps to identify a web resource either by name, location or both

Figure 1.11 Differences between URL and URI

1.11 REST API

Representational State Transfer (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 a web URL as HTTP GET or POST or PUT or DELETE request. After that, a response comes back from the 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.

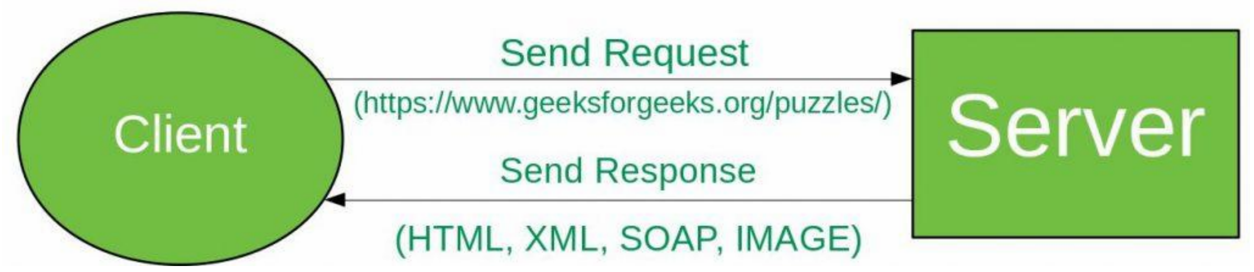


Figure 1.12 REST

In HTTP there are five methods that 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.

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).

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.

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.

PATCH: It is used to 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.

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.

1.12 References

1. MHSS Class and Practical's Notes.
2. Tutorials List - Javatpoint
3. https://www.w3schools.com/
4. https://www.geeksforgeeks.org/
5. https://www.tutorialspoint.com/
6. https://www.cloudflare.com/
7. Wikipedia

Acknowledgement

The Notes as per the chapter Number 1 Web computing, however if any students like to suggest or add the content example or figure etc, please let me know so that it can be incorporated and made available to all the students.

Theory Questions Chapter 1

Q1. Discuss Three Tier Web Development, Also Discuss Forint End and Back End Web developments with technological Stacks

Q2. Explain what full stack web development is, compare different frameworks for full stack web development.

Q3. Explain Http and Https protocol in detail.

Q 4. What is DNS. .Explain the steps in DNS lookup.

Q 5. Explain TLS in detail.

Q 6. Explain XML and JSON.

Q 7. Differentiate between URL and URI.

Q 8. Discuss REST API,

Compiled By
Dr. Ashfaq Shaikh

Subject In charge