Module 1: Web programming fundamentals

- Working of web browser
- HTTP protocol, HTTPS
- DNS, TLS, XML introduction
- Json introduction
- DOM, URL, URI, REST API.

- Working of web browser

What is a browser ?

A web browser also referred as browser is a software application used to access information on world wide web.

As a client/server model, the browser is the client run on a computer that contacts the Web server and requests information.

The Web server sends the information back to the Web browser which displays the results on the computer or other Internet-enabled device that supports a browser.

The main functionality of a browser is to retrieve web resources from the server and display it on the web browser window.

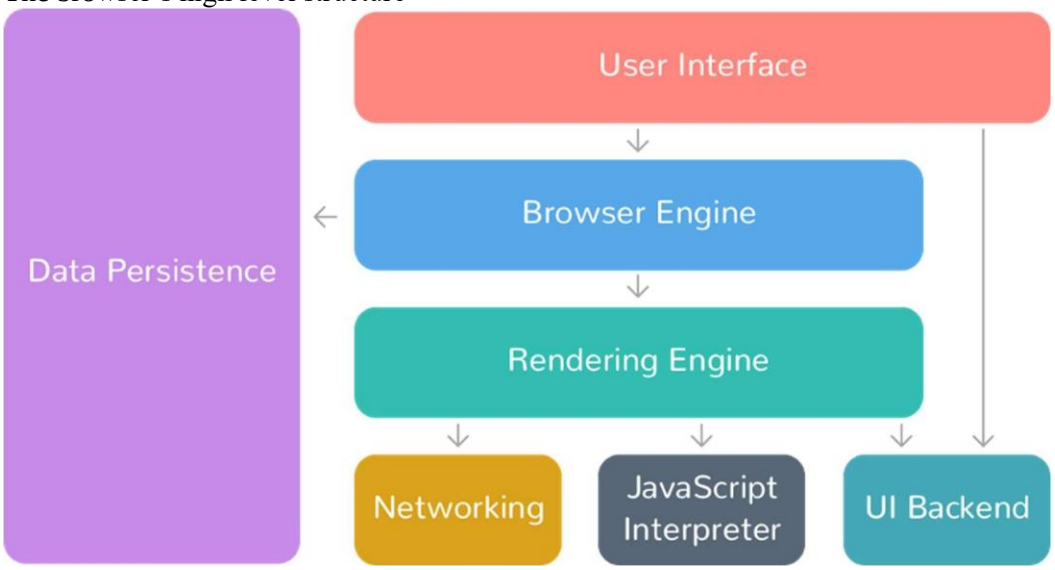
Today's browsers are fully-functional software suites that can interpret and display HTML Web pages, applications, JavaScript, AJAX and other content hosted on Web servers.

Many browsers offer plug-ins which extend the capabilities of the software so it can display multimedia information (including sound and video), or the browser can be used to perform tasks such as videoconferencing, to design web pages or add anti-phishing filters and other security features to the browser.

The web resource is usually an HTML document, but may also be a PDF, image, audio, videos or some other type of content. The location of the resource is specified by using a URI (Uniform Resource Identifier).

A browser contains a well-structured component which performs a series of task to display web resources on the browser window.

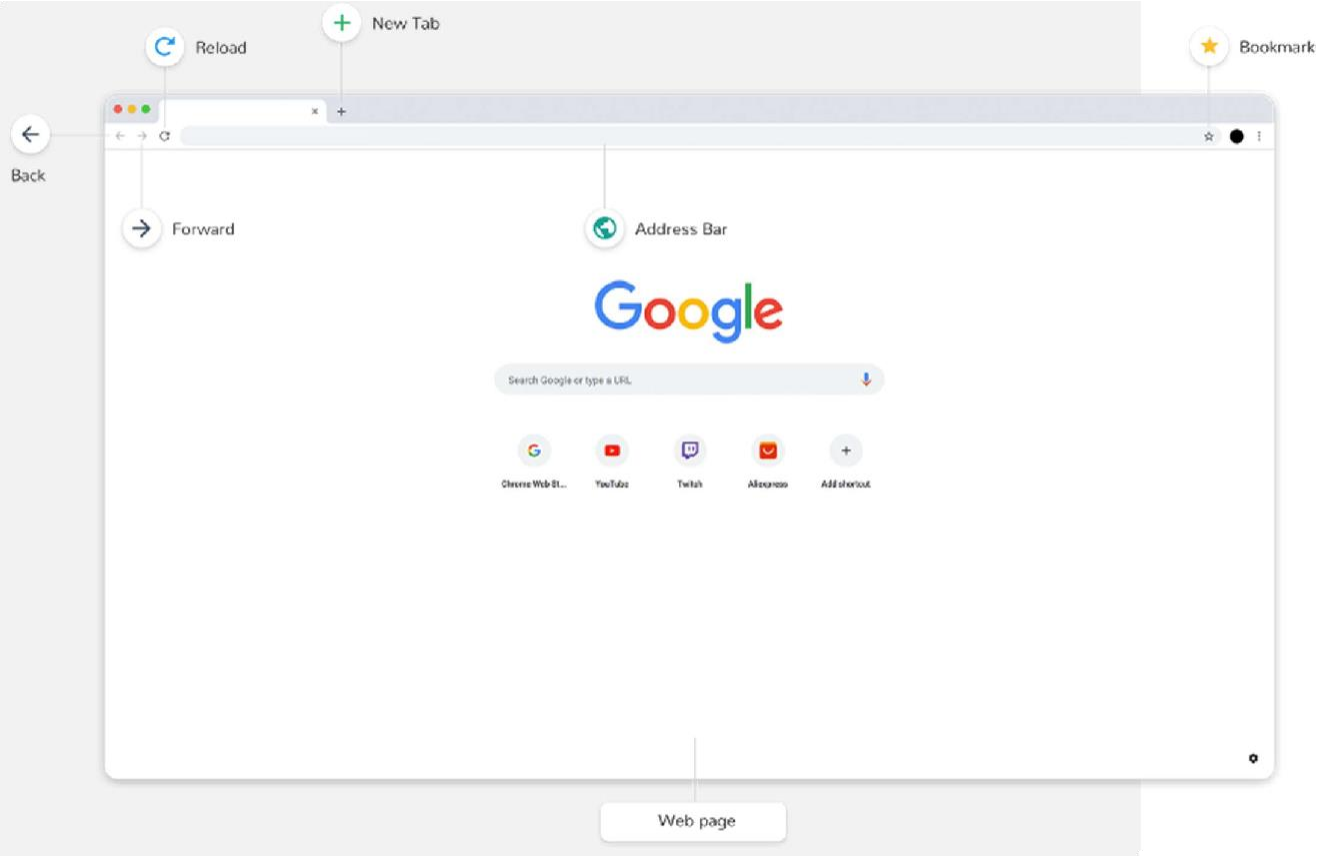
The browser's high level structure



High level structure of a browser

The main components of a web browser are:

1. User Interface (UI): It is the layout of elements available for user to interact in the browser window except for the web page itself. The elements for interaction includes address bar, refresh, back, forward, bookmark options and etc.



User Interface of Chrome Browser

2. Browser Engine: It is the bridge between the UI and the Rendering Engine. Based on inputs from the UI, it queries HTML document and other resources of a web page into an interactive visual representation on the browser window by manipulating the rendering engine.

3. Rendering Engine: It is the component responsible for displaying the requested content on the browser window.

The rendering engine interprets the HTML, XML documents and images that are formatted using CSS and generates the layout that is displayed in the User Interface. However, using plugins or extensions, it can display other types data also.

For example, if an HTML page is requested, then it is responsible for parsing HTML document and the CSS and display the parsed and formatted content on the screen.

By default, it can display HTML , XML documents and images. It can display other formats of data by using browser plugins and extensions.

There are different rendering engines for different browsers. For example, Internet Explorer uses Gecko, Safari uses WebKit, Chrome and Opera (version 15 onwards) uses Blink, an extended implementation of WebKit.

WebKit is an open source rendering engine which started as an engine for the Linux platform and was modified by Apple to support Mac and Windows. See webkit.org for more details.

It is important to note that browsers such as Chrome run multiple instances of the rendering engine: one for each tab. Each tab runs in a separate process.

The Main Flow

The rendering engine gets the content of the requested document from the Networking component in chunks of 8kb.

Once the chunks are received, following will be the basic flow of rendering engine:

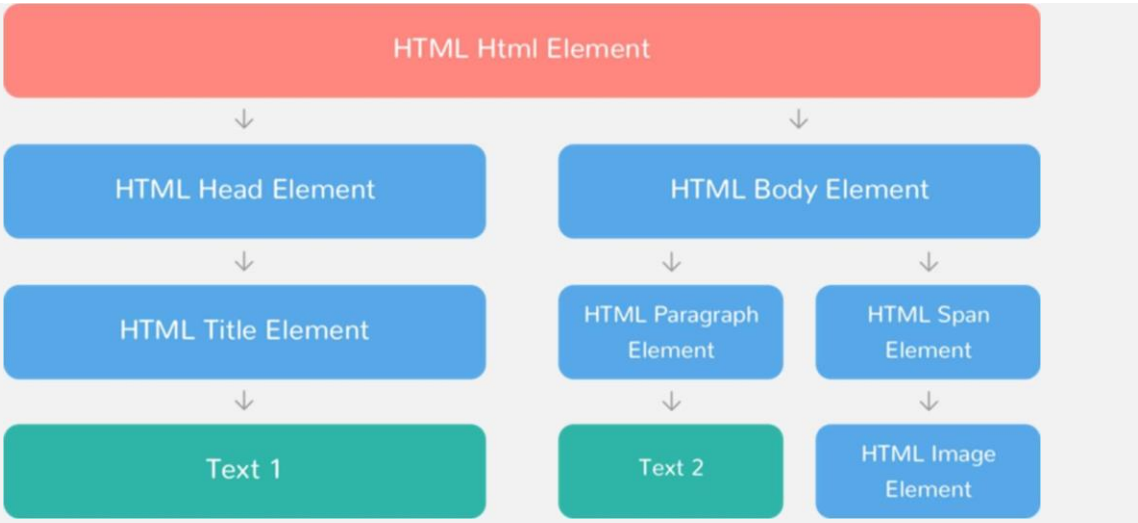
The main flow

So here, in this flow, we can see how a HTML document that was fetched over the network is parsed and rendered into a visually appealing view on the screen. The flow contains primarily four stages:

1. Parsing HTML to construct the DOM tree: The rendering engine will start parsing the HTML document and convert elements to Document Object Models (DOM) nodes in a tree called the "content tree".

```
1   <html>
2     <head>
3       <title>Text 1</title>
4     </head>
5     <body>
6       <p>Text 2</p>
7       <span>
8         <img src="https://bit.ly/1PO8llN" alt="HTML 5 Logo">
9       </span>
10    </body>
11  </html>
```

HTML Document

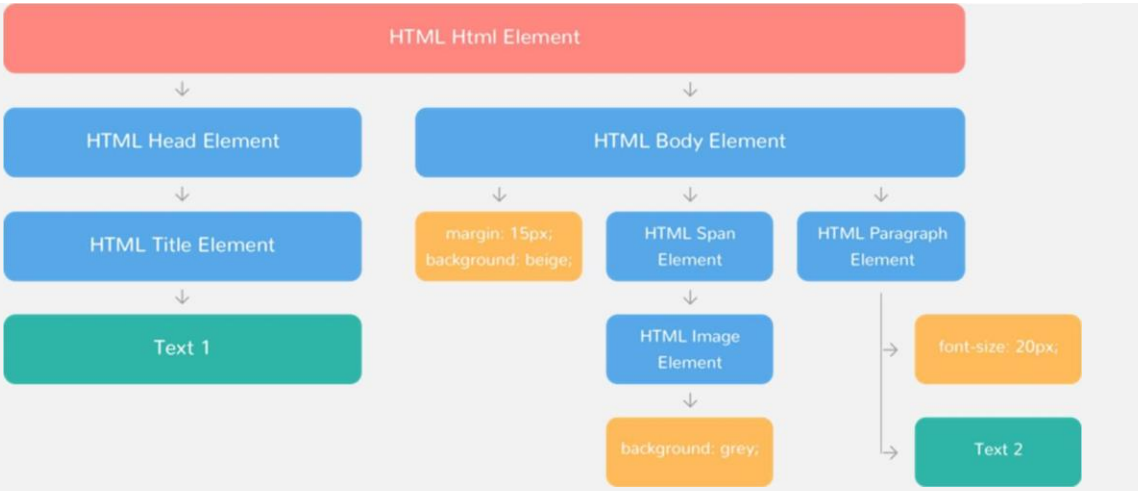


Content Tree

Collection of HTML elements

2. Render tree construction: The engine will parse the style data, both in external CSS files and in style elements. Styling information together with visual instructions in the HTML will be used to create another tree called the “render tree”.

The render tree contains rectangles with visual attributes like colour and dimensions. The rectangles are in the right order to be displayed on the screen.

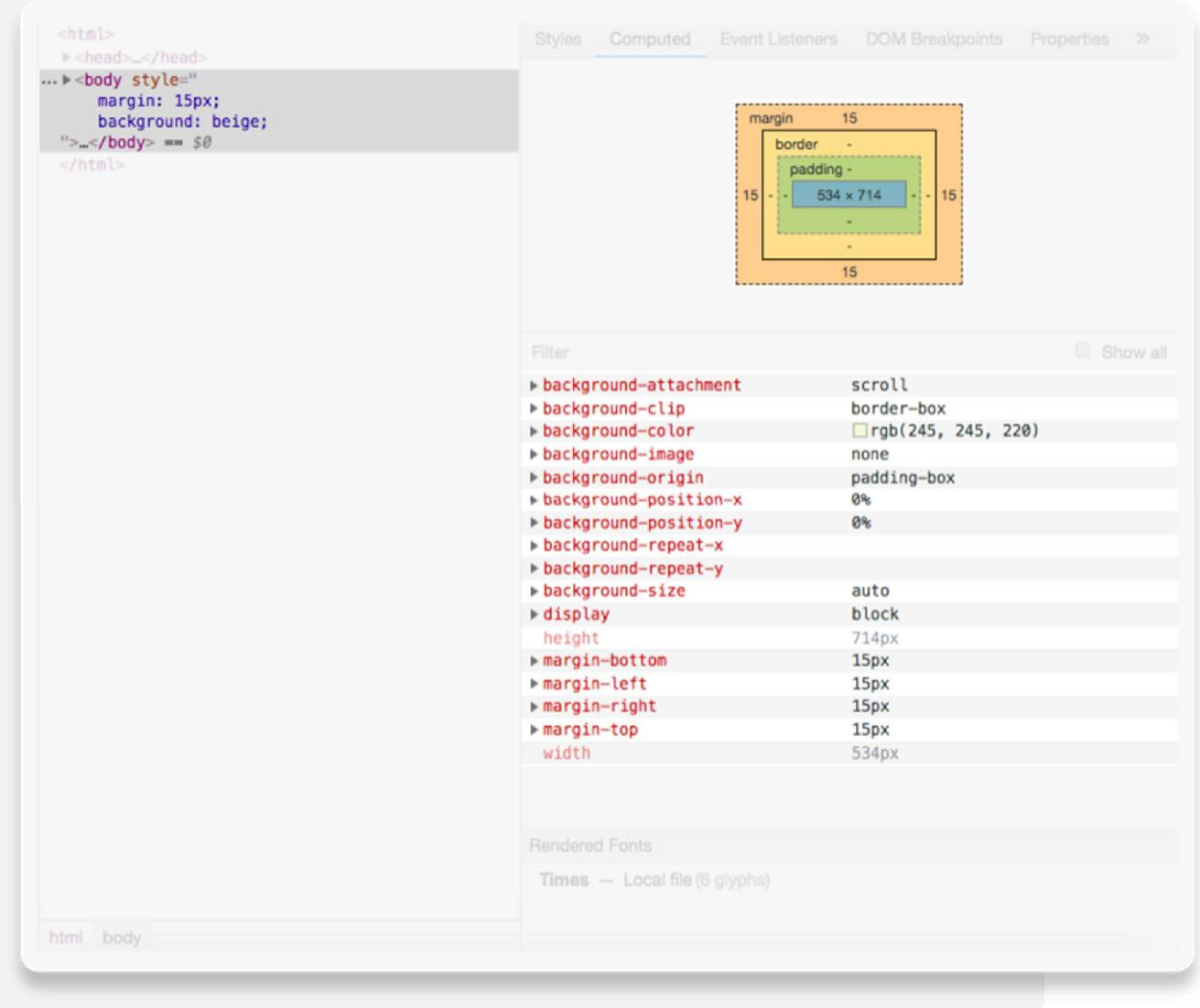


Render Tree

Inline Style for body element

3. Layout of the render tree: After the construction of the render tree it goes through a “layout” process. The output of the layout process is a “box model,” which precisely captures the exact position and size of each element within the viewport: all of the relative measurements are converted to absolute pixels on the screen.

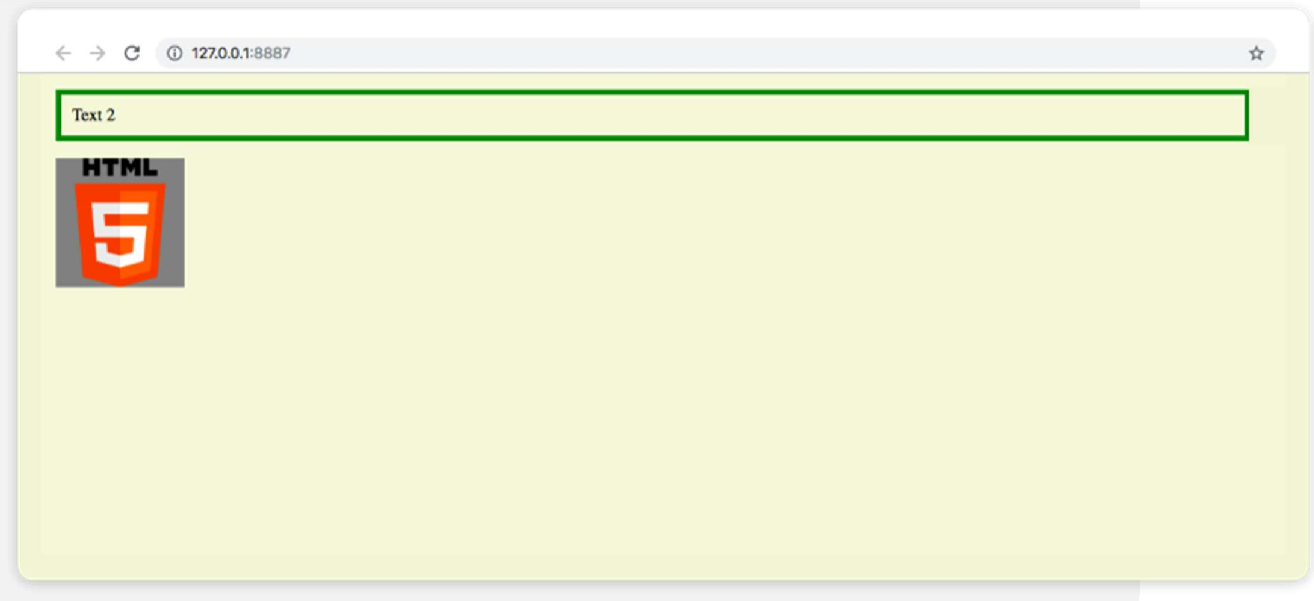
In the below screenshot, you can see the “box model” (margin, border, padding, width and height) information computed for body element.



Computed style and box model details

4. The render tree will be traversed and each node will be painted using the UI backend layer. This process is also referred to as “rasterising”.

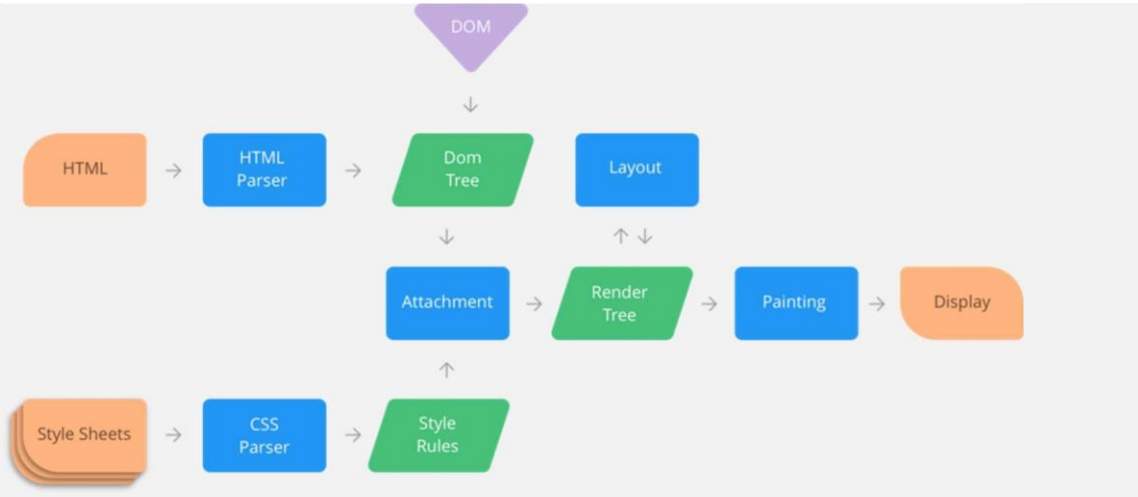
This is the stage, where the computed layout information for each node in the render tree is converted to actual pixels on the screen.



Painted View

It's important to understand that this is a gradual process. For better user experience, the rendering engine will try to display contents on the screen as soon as possible. It will not wait until all HTML is parsed before starting to build and layout the render tree. Parts of the content will be parsed and displayed, while the process continues with the rest of the contents that keeps coming from the network.

Main Flow Examples



WebKit Main Flow

Repaint occurs when changes are made to the element's appearance but do not affect its layout. For example, like changing the background colour, opacity, visibility of an element.

Reflow occurs when changes are made to the element that changes a part of the layout or the entire page. Reflow causes the subsequent reflow of all the child and parent elements as well as any elements following it in the Render/Frame tree.

4. Networking: It is the component that handles all kind of network communication on the browser. It uses a set of common networking protocols like HTTP, HTTPS, FTP and etc while fetching the requested resources through URLs. The network component may implement a cache of retrieved documents in order to reduce network traffic.

5. UI Backend: It is used to draw basic widgets like combobox, alert, pop-up windows, frames etc. This backend exposes a generic interface that is not platform specific. Underneath it uses operating system user interface methods.

6. JavaScript Interpreter: It is a component used to parse and execute the JavaScript code embedded in a web page.

The interpreted results are sent to the rendering engine for display. If the script is external then first the resource is fetched from the network. Parser keeps on hold until the script is executed.

There are different types of JS engines used by different browsers to analyse, parse and execute, by generating the most optimised JS code in the shortest time. A few are,

- Chrome uses V8 Engine
- Mozilla uses SpiderMonkey
- IE and Edge uses Chakra
- Safari uses JavaScriptCore

7. Data Storage: This is a persistence layer. The browser may need to save all sorts of data locally, such as cookies. Browsers also support storage mechanisms such as localStorage, IndexedDB, WebSQL and FileSystem.

It is a small database created on the local drive of the computer where the browser is installed. It manages user data such as cache, cookies, bookmarks and preferences.

URL vs URI: Most important Differences & Full Form You Must Know

What is the URL?

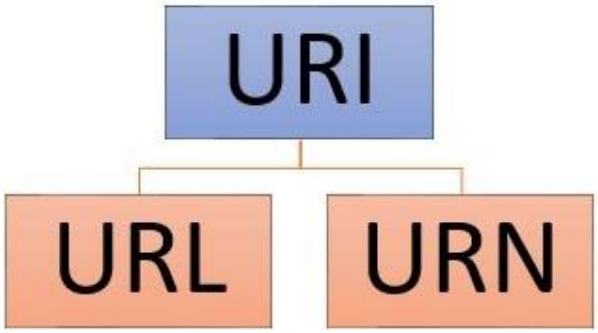
A URL is a global address of documents and protocols to retrieve resource on a computer network. URLs occur most frequently in reference to web pages (HTTP) but can also be used for database access using JDBC, email (mailto), file transfer (FTP), and many other applications. The full form of URL is Uniform Resource Locator.

In this URL Vs. URI tutorial, you will learn:

What is URI?

A URI is a string containing characters that identify a physical or logical resource. URI follows syntax rules to ensure uniformity. Moreover, it also maintains extensibility via a hierarchical naming scheme. The full form of URI is Uniform Resource Identifier.

21.6Minux Linux Beginner Tutorial



Types of URI

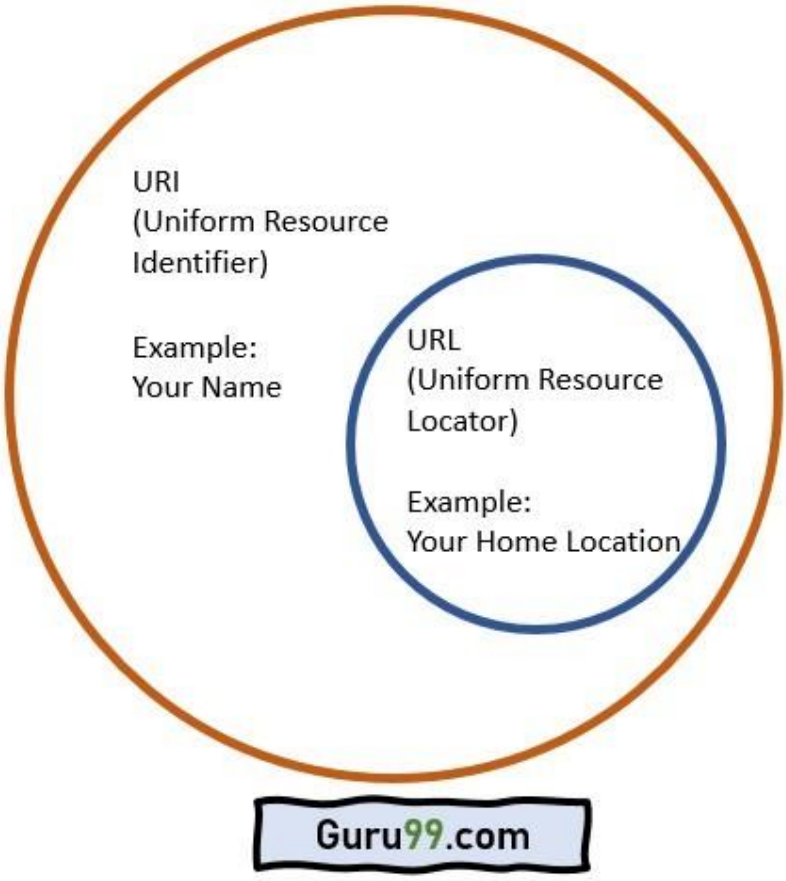
As mention in the above figure, there are two types of URI:

1. URL: URL specifies a location on the computer network and technique for retrieving it.
2. URN: Uniform Resource Name (URN) is an internet resource that specifies URN scheme.

KEY DIFFERENCES:

- URL is a subset of URI that specifies where a resource exists and the mechanism for retrieving it, while URI is a superset of URL that identifies a resource
- The main aim of URL is to get the location or address of a resource whereas the main aim of URI is to find a resource.
- URL is used to locate only web pages, on the other hand, URI in used in HTML, XML and other files.
- URL contains components such as protocol, domain, path, hash, query string, etc. while, URI contains components like scheme, authority, path, query, etc.
- Example of URL is : https://google.com while example of URI is :urn:isbn:0-486-27557-4.

Ven Diagram of URIs and URL



Ven diagram of URI and URL

As mention in the above diagram, "your name" can be a URI because it identifies you. It cannot be URL since it does not assist any person to find your home location.

On the other hand, "your home location" can be URI as well as URL. The reason is both identify you and gives a home location for you.

Syntax of URL

Here is a Syntax of URL:

http://www.domainname.com/folder-name/web page-file-name.htm

We can divide the above URL into the following parts:

- Protocol: It is the first part of the URL. Here, the protocol name is Hypertext Transfer Protocol (HTTP).
- http://www.domainname.com/: It is your domain name. It is also known as server id or the host.
- /folder-name/: It indicates that the website page referenced in "filed" in a given folder on the webserver.
- web-page-file-name.htm: It is actually a web page file name. The ".htm" is an extension for the HTML file, which shows that it is a static web page. File names can have different extensions or it is depend on how you have set up a web server. There could be no extension at all, and the URL could end with a slash line (/).

Example:

This example URL has a folder but no extension

https://career.guru99.com/category/heavy-industries/

This example URL has no folder

https://www.guru99.com/what-is-sap.html

This example URL has no extension

https://career.guru99.com/top-33-investment-banking-interview-questions-answers/

Syntax of URI

Here is a syntax of URI:

URI = scheme:[//authority]path[?query][#fragment]

The URI includes the following parts:

- Scheme component: It is a non-empty component followed by a colon (:). Scheme contains a sequence of characters starting with a letter and followed by any combination of digits, letters, period (.), hyphen (-), or plus (+).

  Examples of well-known schemes include HTTP, HTTPS, mailto, file, FTP, etc. URI schemes must be registered with the Internet Assigned Numbers Authority (IANA).

- Authority component: It is an optional field and is preceded by //. It consists of

1. Optional userinfo subcomponent that might consist of a username and password (optional).
2. A host subcomponent containing either an IP address or a registered name.
3. An optional port subcomponent that is followed by a colon (:)

- Path: A path contains a sequence of segments that are separated by a slash.
- Query component: It is optional and preceded by a question mark (?). Query component contains a query string of non-hierarchical data.
- Fragment component: It is an optional field and preceded by a hash (#). Fragment component includes a fragment identifier giving direction to a secondary resource.

Example of URI

No protocol mentioned

www.guru99.com

Domain not mentioned

what-is-sap.html

Protocol mentions

ldap://[2001:db8::7]/c=GB?objectClass?one
mailto:abc@example.com
tel:+1-816-555-1212
telnet://192.0.2.16:80/

Confusion about URN

There is a confusion about URN that, if you implement protocols like https, ftp, etc, then it is called a URL, even though it is a URI.

The problem with such a debate is that appropriate RFC are extremely dense and sometimes even contradictory. For example, RFC 3986 says, URI can be either a name, locator or both.

Why URL?

Here are the important reasons of using URL:

- The Information written in the URL gives you the ability to switch from one web page to another with just one mouse click.

- URL tells you how to access a particular resource.
- Whenever you type a URL into your browser or click any hypertext link, your web browser sends a request to a web server to download one or more files.
- Every URL is unique and identifies one particular file.
- A website URL or domain is one of the most crucial parts of your website. By using simple words or string that usually end with a .org, .com, or .net, you would be able to get traffic to your website.

Why URI?

Here are the important reasons of using URI:

- A Uniform Resource Identifier is essential to the semantic web because it prevents ambiguity.
- A URI search the name as well as the location of a resource or file, which is in a uniform format.
- It has a string of characters for the specific filename and path.
- URI provides a method for resources to be accessed by other systems over the World Wide Web or across a network. It is used by web browsers and P2P (Peer to Peer) file-sharing software to find and download files.
- URI allows new file types to be defined without affecting old files you have.
- You can assign a single resource to associate with multiple representations.

URL Vs. URI

URL	URI
URL stands for Uniform Resource Locator.	URI stands for Uniform Resource Identifier.

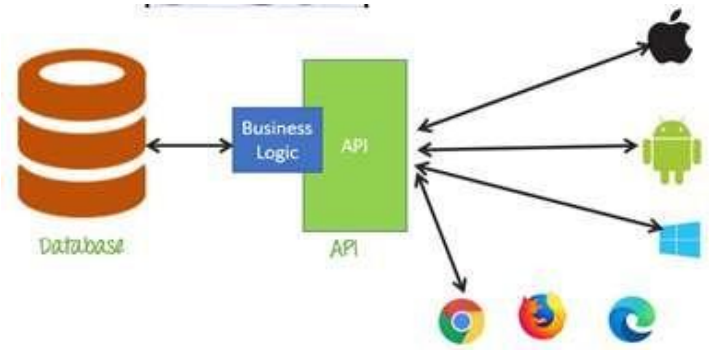
URL is a subset of URI that specifies where a resource is exists and the mechanism for retrieving it.	A URI is a superset of URL that identifies a resource either by URL or URN (Uniform Resource Name) or both.
The main aim is to get the location or address of a resource	The main aim of URI is to find a resource and differentiate it from other resources using either name or location.
URL is used to locate only web pages	Used in HTML, XML and other files XSLT (Extensible Stylesheet Language Transformations) and more.
The scheme must be a protocol like HTTP, FTP, HTTPS, etc.	In URI, the scheme may be anything like a protocol, specification, name, etc.
Protocol information is given in the URL.	There is no protocol information given in URI.
Example of URL: https://google.com	Example of URI: urn:isbn:0-486-27557-4
It contains components such as protocol, domain, path, hash, query string, etc.	It contains components like scheme, authority, path, query, fragment component, etc.
All URLs can be URIs	Not all URIs are URLs since a URI can be a name instead of a locator.

What is an API?

Application Programming Interface (API) is a software interface that allows two applications to interact with each other without any user intervention. API is a collection of software functions and procedures. In simple terms, API means a software code that can be accessed

or executed. API is defined as a code that helps two different software's to communicate and exchange data with each other.

It offers products or services to communicate with other products and services without having to know how they're implemented.



How does it work?

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

Example 1:

Let see how API works using simple daily life example. Imagine that you went to a restaurant to take lunch or dinner. The waiter comes to you gives you a menu card, and you will provide personalize it order like you want a veg sandwich but without onion.

After some time, you will get your order from the waiter. However, it is not that simple as it looks as there is some process that happens in between.

Here, the waiter plays an important part as you will neither go to the kitchen to collect your order nor will you tell the kitchen staff what you want all this done by the waiter.

API also does the same by taking your request, and just like the waiter tell the system what you want and give a response back to you.

Example 2:

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

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

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

Why would we need an API?

Here, are some reason for using API:

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

Features of API

Here are some important features of API:

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

Types of API

There are mainly four main types of APIs:

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

Communication level of APIs:

Here, are some communication level of APIS:

High-Level APIs:

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

Low-Level APIs:

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

What is Web APIs?

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

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

1.Server-side:

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

2. Client-side:

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

Examples of web API:

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

Restful Architecture

An application or architecture considered RESTful or REST-style has the following characteristics

1. State and functionality are divided into distributed resources – This means that every resource should be accessible via the normal HTTP commands of GET, POST, PUT, or DELETE. So if someone wanted to get a file from a server, they should be able to issue the GET request and get the file. If they want to put a file on the server, they should be able to either issue the POST or PUT request. And finally, if they wanted to delete a file from the server, they an issue the DELETE request.
2. The architecture is client/server, stateless, layered, and supports caching –

- Client-server is the typical architecture where the server can be the web server hosting the application, and the client can be as simple as the web browser.
- Stateless means that the state of the application is not maintained in REST.

  For example, if you delete a resource from a server using the DELETE command, you cannot expect that delete information to be passed to the next request.

  In order to ensure that the resource is deleted, you would need to issue the GET request. The GET request would be used to first get all the resources on the server. After which one would need to see if the resource was actually deleted.

RESTFul Principles and Constraints

The REST architecture is based on a few characteristics which are elaborated below. Any RESTful web service has to comply with the below characteristics in order for it to be called RESTful. These characteristics are also known as design principles which need to be followed when working with RESTful based services.

1. RESTFul Client-Server

This is the most fundamental requirement of a REST based architecture. It means that the server will have a RESTful web service which would provide the required functionality to the client. The client send's a request to the web service on the server. The server would either reject the request or comply and provide an adequate response to the client.

2. Stateless

The concept of stateless means that it's up to the client to ensure that all the required information is provided to the server. This is required so that server can process the response appropriately. The server should not maintain any sort of information between requests from the client. It's a very simple independent question-answer sequence. The client asks a question, the server answers it appropriately. The client will ask another question. The server will not remember the previous question-answer scenario and will need to answer the new question independently.

3. Cache

The Cache concept is to help with the problem of stateless which was described in the last point. Since each server client request is independent in nature, sometimes the client might ask the server for the same request again. This is even though it had already asked for it in the past. This request will go to the server, and the server will give a response. This increases the traffic across the network. The cache is a concept implemented on the client to store requests which have already been sent to the server. So if the same request is given by the client, instead of going to the server, it would go to the cache and get the required information. This saves the amount of to and fro network traffic from the client to the server.

4. Layered System

The concept of a layered system is that any additional layer such as a middleware layer can be inserted between the client and the actual server hosting the RESTFul web service (The middleware layer is where all the business logic is created. This can be an extra service created with which the client could interact with before it makes a call to the web service.). But the introduction of this layer needs to be transparent so that it does not disturb the interaction between the client and the server.

5. Interface/Uniform Contract

This is the underlying technique of how RESTful web services should work. RESTful basically works on the HTTP web layer and uses the below key verbs to work with resources on the server

- POST - To create a resource on the server
- GET - To retrieve a resource from the server
- PUT - To change the state of a resource or to update it
- DELETE - To remove or delete a resource from the server

JSON

Working with JavaScript Object Notation (JSON):

- Create data in JSON format
- JSON Parser.

☐ Introduction

- JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable and for exchanging data on the web.

Exchanging Data

- When exchanging data between a browser and a server, the data can only be text.
- JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from the server into JavaScript objects.
- This way we can work with the data as JavaScript objects, with no complicated parsing and translations.
- It supports data structures like object and array. So it is easy to write and read data from JSON.
- Conventions used by JSON are known to programmers, which include C, C++, Java, Python, Perl, etc.

JSON

- JSON stands for JavaScript Object Notation.
- The format was specified by Douglas Crockford.
- JSON is an open standard data-interchange format.
- JSON is lightweight and self describing.
- JSON is originated from JavaScript.
- JSON is easy to read and write.

- It has been extended from the JavaScript scripting language.
- The filename extension is **.json**.
- JSON Internet Media type is **application/json**.
- The Uniform Type Identifier is public.json.

Uses of JSON

- It is used while writing JavaScript based applications that includes browser extensions and websites.
- JSON format is used for serializing and transmitting structured data over network connection.
- It is primarily used to transmit data between a server and web applications.
- Web services and APIs use JSON format to provide public data.
- It can be used with modern programming languages.

Characteristics of JSON

- JSON is easy to read and write.
- It is a lightweight text-based interchange format.
- 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.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

Example "name":"John"
JSON names require double quotes.

JSON Values

In **JSON**, *values* must be one of the following data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null

JSON values **cannot** be one of the following data types:

- a function
- a date
- *undefined*

JSON Strings

Strings in JSON must be written in double quotes.

Example

```
{ "name":"John" }
```

JSON Numbers

Numbers in JSON must be an integer or a floating point.

Example

```
{ "age":30 }
```

JSON Objects

Values in JSON can be objects.

Example

```
{
"employee":{ "name":"John", "age":30, "city":"New York" }
}
```

Objects as values in JSON must follow the same rules as JSON objects.

JSON Nested Object Example

A JSON object can have another object also. Let's see a simple example of JSON object having another object.

```
{
"firstName": "Sonoo", "lastName": "Jaiswal", "age": 27,
"address" : {
"streetAddress": "Plot-6, Mohan Nagar", "city": "Ghaziabad",
"state": "UP", "postalCode": "201007"
}
}
```

JSON Arrays

Values in JSON can be arrays.

Example

```
{
"employees":[ "John", "Anna", "Peter" ]
}
```

Let's see the example of JSON array having objects.

```
[
{"name":"Ram", "email":"Ram@gmail.com"},
{"name":"Bob", "email":"bob32@gmail.com"}
]
```

JSON Booleans

Values in JSON can be true/false.

Example

```
{ "sale":true }
```

JSON null

Values in JSON can be null.

Example

```
{ "middlename":null }
```

JSON vs XML

Both JSON and XML can be used to receive data from a web server.

The following JSON and XML examples both define an employees object, with an array of 3 employees:

JSON Example

```
{"employees":[
{ "firstName":"John", "lastName":"Doe" },
{ "firstName":"Anna", "lastName":"Smith" },
{ "firstName":"Peter", "lastName":"Jones" }
]}
```

XML Example

```
<employees>
<employee>
<firstName>John</firstName> <lastName>Doe</lastName>
</employee>
<employee>
<firstName>Anna</firstName> <lastName>Smith</lastName>
```

</employee>
<employee>
<firstName>Peter</firstName> <lastName>Jones</lastName>
</employee>
</employees> 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: Using XML

- Fetch an XML document
- Use the XML DOM to loop through the document
- Extract values and store in variables Using JSON
- Fetch a JSON string
- JSON.Parse the JSON string

JSON Schema

JSON Schema is a specification for JSON based format for defining the structure of JSON data. It was written under IETF draft which expired in 2011. JSON Schema −

- Describes your existing data format.
- Clear, human- and machine-readable documentation.
- Complete structural validation, useful for automated testing.
- Complete structural validation, validating client-submitted data.

JSON Schema Validation Libraries

There are several validators currently available for different programming languages. Currently the most complete and compliant JSON Schema validator available is JSV.

Languages	Libraries
C	WJElement (LGPLv3)
Java	json-schema-validator (LGPLv3)
.NET	Json.NET (MIT)
ActionScript 3	Frigga (MIT)
Haskell	aeson-schema (MIT)
Python	Jsonschema
Ruby	autoparse (ASL 2.0); ruby-jsonschema (MIT)
PHP	php-json-schema (MIT). json-schema (Berkeley)
JavaScript	Orderly (BSD); JSV; json-schema; Matic (MIT); Dojo; Persevere (modified BSD or AFL 2.0); schema.js.

JSON Schema Example

Given below is a basic JSON schema, which covers a classical product catalog description −

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type": "object",

  "properties": {

    "id": {
      "description": "The unique identifier for a product",
      "type": "integer"
    },

    "name": {
      "description": "Name of the product",
      "type": "string"
    },

    "price": {
      "type": "number",
      "minimum": 0,
      "exclusiveMinimum": true
    }
  },

  "required": ["id", "name", "price"]
}
```

Let's the check various important keywords that can be used in this schema −

Sr.No.	Keyword & Description
1	**$schema** The $schema keyword states that this schema is written according to the draft v4 specification.
2	**title**

	You will use this to give a title to your schema.
3	**description** A little description of the schema.
4	**type** The type keyword defines the first constraint on our JSON data: it has to be a JSON Object.
5	**properties** Defines various keys and their value types, minimum and maximum values to be used in JSON file.
6	**required** This keeps a list of required properties.
7	**minimum** This is the constraint to be put on the value and represents minimum acceptable value.
8	**exclusiveMinimum** If "exclusiveMinimum" is present and has boolean value true, the instance is valid if it is strictly greater than the value of "minimum".
9	**maximum** This is the constraint to be put on the value and represents maximum acceptable value.
10	**exclusiveMaximum** If "exclusiveMaximum" is present and has boolean value true, the instance is valid if it is strictly lower than the value of "maximum".

11	**multipleOf** A numeric instance is valid against "multipleOf" if the result of the division of the instance by this keyword's value is an integer.
12	**maxLength** The length of a string instance is defined as the maximum number of its characters.
13	**minLength** The length of a string instance is defined as the minimum number of its characters.
14	**pattern** A string instance is considered valid if the regular expression matches the instance successfully.

JSON Parser

JSON.parse()

- A common use of JSON is to exchange data to/from a web server.
- When receiving data from a web server, the data is always a string.
- Parse the data with JSON.parse(), and the data becomes a JavaScript object.
- The **JSON.parse()** method parses a JSON string, constructing the JavaScript value or object described by the string.

Example - Parsing JSON

- Imagine we received this text from a web server: '{ "name":"John", "age":30, "city":"New York"}'
- Use the JavaScript function JSON.parse() to convert text into a JavaScript object:

```
var obj = JSON.parse('{ "name":"John", "age":30, "city":"New York"}');
```

Make sure the text is written in JSON format, or else you will get a syntax error.

JSON.stringify()

- A common use of JSON is to exchange data to/from a web server.
- When sending data to a web server, the data has to be a string.
- Convert a JavaScript object into a string with JSON.stringify().
- The **JSON.stringify()** method converts a JavaScript object or value to a JSON string, optionally replacing values if a replacer function is specified or optionally including only the specified properties if a replacer array is specified.

Stringify a JavaScript Object

- Imagine we have this object in JavaScript:

```
var obj = { name: "John", age: 30, city: "New York" };
```

- Use the JavaScript function JSON.stringify() to convert it into a string. var myJSON = JSON.stringify(obj);

m

The result will be a string following the JSON notation.

Example

```
var obj = { name: "John", age: 30, city: "New York" };
var myJSON = JSON.stringify(obj); document.getElementById("demo").innerHTML = myJSON;
```

Stringify a JavaScript Array

It is also possible to stringify JavaScript arrays: Imagine we have this array in JavaScript:
var arr = ["John", "Peter", "Sally", "Jane"];

Use the JavaScript function JSON.stringify() to convert it into a string. var myJSON = JSON.stringify(arr);

The result will be a string following the JSON notation.

myJSON is now a string, and ready to be sent to a server:

Example

```
var arr = [ "John", "Peter", "Sally", "Jane" ];
var myJSON = JSON.stringify(arr); document.getElementById("demo").innerHTML = myJSON;
```