**JSON (JavaScript Object Notation)**

-a popular and agreed format for sending, receiving and storing data

-we use JSON.stringify() to turn the data into JSON format

-we use JSON.parse() to turn back the data into its original form

**AJAX (Asynchronous JavaScript and XML)**

-process of sending and recieving data without a page reload is known as AJAX

-web browsers have a built-in tool named XMLHttpRequest

-asynchronous = in the background/not requiring a page refresh

-XML is a data format very similar to JSON (JSON is more widely used)

-we are actually not using AJAX here => could be called AJAJ (Asynchronous JS and JSON) => the name XML is inside only because of its importance in the history of development

how do we use it:

var ourRequest = new XMLHttpRequest(); //using the XHR instance to call in the downloading of the data

ourRequest.open('get', `animals-1.json`, true); //getting the data out of JSON, what is the location of the data, ?

ourRequest.onload = function() { //what to do after onload

insert code!

}

ourRequest.send() //send the response

**HTTP, HTTPS and WWW**

**HTTP** stands for **H**yper **T**ext **T**ransfer **P**rotocol **(HTTP is the action)**

**WWW** is about communication between web **clients** and **servers (WWW is the phenomen)**

Communication between client computers and web servers is done by sending **HTTP Requests** and receiving **HTTP Responses**

**World Wide Web Communication**

The World Wide Web is about communication between web **clients** and web **servers**.

**Clients** are often browsers (Chrome, Edge, Safari), but they can be any type of program or device.

**Servers** are most often computers in the cloud.

## HTTP Request / Response

Communication between clients and servers is done by **requests** and **responses**:

1. A client (a browser) sends an **HTTP request** to the web
2. An web server receives the request
3. The server runs an application to process the request
4. The server returns an **HTTP response** (output) to the browser
5. The client (the browser) receives the response

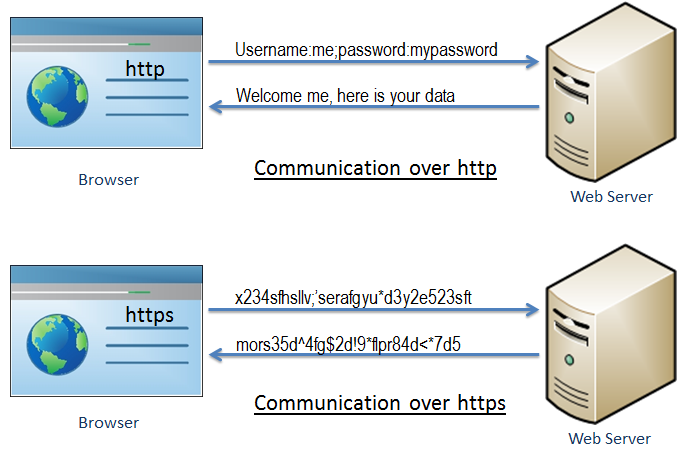
**The HTTP Request Circle**

A typical HTTP request / response circle:

1. The browser requests an HTML page. The server returns an HTML file.
2. The browser requests a style sheet. The server returns a CSS file.
3. The browser requests an JPG image. The server returns a JPG file.
4. The browser requests JavaScript code. The server returns a JS file
5. The browser requests data. The server returns data (in XML or JSON).

**HTTP and HTTPS**

-HTTPS stands for Hyper Text Transfer Protocol Secure. It is a protocol for securing the communication between two systems e.g. the browser and the web server.



-As you can see in the above figure, [http](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) transfers data between the browser and the web server in the hypertext format, whereas https transfers data in the encrypted format. Thus, https prevents hackers from reading and modifying the data during the transfer between the browser and the web server. Even if hackers manage to intercept the communication, they will not be able to use it because the message is encrypted.

**XHR - XML Http Request**

All browsers have a built-in **XMLHttpRequest Object (XHR)**.

XHR is a JavaScript object that is used to transfer data between a web browser and a web server.

XHR is often used to request and recieve data for the purpose of modifying a web page.

Despite the XML and Http in the name, XHR is used with other protocols than HTTP, and the data can be of many different types like[HTML](https://www.w3schools.com/whatis/whatis_html.asp), [CSS](https://www.w3schools.com/whatis/whatis_css.asp), [XML](https://www.w3schools.com/whatis/whatis_xml.asp), [JSON](https://www.w3schools.com/whatis/whatis_json.asp), and plain text.

The XHR Object is a **Web Developers Dream**, because you can:

* Update a web page without reloading the page
* Request data from a server - after the page has loaded
* Receive data from a server - after the page has loaded
* Send data to a server - in the background

The XHR Object is the underlying concept of [**AJAX**](https://www.w3schools.com/whatis/whatis_ajax.asp) and [**JSON**](https://www.w3schools.com/whatis/whatis_json.asp):



**API**

-API stands for **Application Programming Interface**

-most companies have built API’s for their customers or for internal use

-how different applications and devices connect together

-its the thing that makes all the interactivity possible (booking a reservation, flight, ordering new clothes and products

-API is the **messenger** that takes the request and tells the system what you want to do and then returns the response back to you

**Analogy:**

-You sit a restaurant with a menu in front of you

-**You are the client** and the **kitchen is the server/system**

-You need a **waiter/messenger (API)** to tell what you want to order

-You give the **request** **(what food)** to the waiter **(messenger)**

-The **messenger takes the request** and takes it to the **kitchen** **(server)**

-In the kitchen the food is prepared **(response/data)**

-The waiter brings back the food to you **(response)**

**Example:**

**1.**

You book a flight on **FlyAirways** and you request a response from ther **FlyAirways server** => **Has direct access to the information/data**

**2.**

-when you book a flight from a online travel service => **Aggregates information from many different airlines)**

-the travel service interracts with the airlines API => Requests the information from the direct server => brings back the info to you (client)

**URI**

# URL -- [Uniform Resource Locator](https://en.wikipedia.org/wiki/URL)

Contains information about how to fetch a resource from its location. For example:

* http://example.com/mypage.html
* ftp://example.com/download.zip
* mailto:user@example.com
* file:///home/user/file.txt
* http://example.com/resource?foo=bar#fragment
* /other/link.html (A relative URL, only useful in the context of another URL)

URLs always start with a protocol (http) and usually contain information such as the network host name (example.com) and often a document path (/foo/mypage.html). URLs may have query parameters and fragment identifiers.

# URN -- [Uniform Resource Name](https://en.wikipedia.org/wiki/Uniform_Resource_Name)

Identifies a resource by name. It always starts with the prefix urn: For example:

* urn:isbn:0451450523 to identify a book by its ISBN number.
* urn:uuid:6e8bc430-9c3a-11d9-9669-0800200c9a66 a globally unique identifier
* urn:publishing:book - An XML namespace that identifies the document as a type of book.

URNs can identify ideas and concepts. They are not restricted to identifying documents. When a URN does represent a document, it can be translated into a URL by a "resolver". The document can then be downloaded from the URL.

# URI -- [Uniform Resource Identifier](https://en.wikipedia.org/wiki/Uniform_Resource_Identifier)

URIs encompasses both URLs, URNs, and other ways to indicate a resource.

An example of a URI that is neither a URL nor a URN would be a [data URI](https://en.wikipedia.org/wiki/Data_URI_scheme) such as data:,Hello%20World. It is not a URL or URN because the URI contains the data. It neither names it, nor tells you how to locate it over the network.

There are also uniform resource citations (URCs) that point to meta data about a document rather than to the document itself. An example of a URC would be an indicator for viewing the source code of a web page: view-source:http://example.com/. A URC is another type of URI that is neither URL nor URN.

**REST (RESTful API)**

-Stands for **Represntational State Transfer**

-its a **architecture style** for designing networked applications

-relies on a **stateless, client-server** protocol, almost always **HTTP**

-treats server objects as resources that can be created or destroyed

-can be used by virtually any programming language **(in most good languages you can make a HTTP request)**

**Explanation**

-we have a company named vidly for renting out movies

-we have a client app where we manage the list of our customers

-on the server side we should expose a service on the endpoint like this

<http://vidly.com/api/customers>

-our client can use a http talk to our service

-the URL can start with HTTP or HTTPS (**http://**)

-then comes the domain (**vidly.com**)

-then comes api => not compulsory/necessity (**api**)

-the comes the resource (**customers**)

**HTTP methods**

-get: retrieve data from a specified resource

-post: submit data to be processed to a specified resource (filling a website form)

-put: update a specified resource

-delete: delete a specified resource

**Endpoints**

-the URI/URL where API/service can be accessed by a client application

GET <https://mysite.com/api/users> (gets the list of users)

GET <https://mysite.com/api/users>/1 OR <https://mysite.com/api/users>/details/1 (get the specific one)

POST <https://mysite.com/api/users> (post a list of users)

PUT <https://mysite.com/api/users>/1 OR <https://mysite.com/api/users>/details/1 (modify the specific one)

DELETE <https://mysite.com/api/users>/1 OR <https://mysite.com/api/users>/details/1 (delete the specific one)

**Cloud**

-virtual storage where we can store our data

-when you upload a file to a cloud-based service like Google Drive, Icloud, Dropbox, etc… => the file gets copied into a **data server database** (actuall **physical places**, where companies store your data on mutliple **hard drives**)

-the database servers are also known as server farms as they exist in multiple location => for backup, if something bad happened (redundancy)