Web Development

Introduction

What Is Web?

Node.js Backend (Master Course) - #077 - Web & Web Development - What is Web

https://youtu.be/8sdDswy47LY?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Key Points:

1. **Why is a Server Created?**
   - To respond to requests.
2. **Why is the Internet (Network) Created?**
   - Acts as a mediator to direct requests to the Server.
3. **Types of Networks:**
   - **Local Network**: All devices within one network.
   - **Global Network**: Devices across multiple networks.
4. **Public IP, Domain, and DNS Name**:
   - **Public Address**: Makes the webpage accessible to all users.
   - DNS: Domain Name System.

   ▼ Public IP, Domain, and DNS

   Public IP, **Domain**, and **DNS** are not the same, although they are related. Here's how they differ:

   1. **Public IP**:
      - This is the unique address assigned to a device (such as a server) on the internet.
      - It is a numeric address (e.g., `192.168.1.1`) that identifies a device over the global network.
      - Public IP addresses are assigned by Internet Service Providers (ISPs) or network providers.

2. **Domain**:
    - A **domain** is a human-readable address used to identify a website (e.g., `example.com`).
    - It is easier for people to remember than a public IP address, which is a string of numbers.
    - Domains are purchased or registered through domain registrars (like GoDaddy, Namecheap, etc.).
3. **DNS (Domain Name System)**:
    - **DNS** is like a phonebook for the internet. It translates a domain name (e.g., `example.com`) into a public IP address (e.g., `192.168.1.1`) so that browsers can load the website.
    - When you type a domain into your browser, DNS servers look up the corresponding IP address and direct you to the correct website.
    - DNS servers store and manage this mapping of domain names to IP addresses.

In Summary:

- **Public IP**: The numeric address of a device on the internet.
- **Domain**: The human-readable address (URL) for accessing websites.
- **DNS**: The system that translates domain names into public IP addresses.

So, while they all work together to allow users to access websites, they serve different purposes and are not the same.

5. **ACL (Access Control List)**:
    - A list of permissions associated with system resources (objects).
    - Specifies which users or processes can access objects and what operations they can perform on them.
    - **Usage in Web Applications**: Controls access to resources, such as fetching data from a web application.

Client-Server Model

Node.js Backend (Master Course) - #078 - Web & Web Development - Client-Server Model

https://youtu.be/5PbjHdCHSIQ?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Key Points:

- **Client Side**:
    - Sends a request.
    - Typically involves the **Front-End** (UI) of the application.
- **Server Side**:
    - Receives the request, processes it, and interacts with the **Back-End** and **Database**.
    - Returns data from the database to the front-end.

```
Client Side              ||                    Server Side
```

```
Front-End                ||        Front-End   ||   Back-End   ||   Database
```

Request-Response Model

Node.js Backend (Master Course) - #079 - Web & Web Development - Request-Response Model

https://youtu.be/pK-Iyqgsevw?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

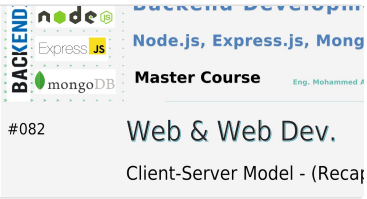
Key Points:

- **HTTP Request**: From Client-side to Server-side.
- **HTTP Response**: From Server-side to Client-side.
- **Real-Time Model**:
  - Different from the Request-Response Model.
  - Difference from Real-Time Model: The request-response model operates in a request/response cycle, whereas the real-time model allows continuous communication without the need for multiple requests and responses.
  - Real-time communication happens instantly, without waiting for a response.

Client-Server Model - Recap

Node.js Backend (Master Course) - #082 - Web & Web Development - Client-Server Model - (Recap)

https://youtu.be/uLBNKWBYynI?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H



Key Points:

1. **Client**:
   - The browser or application that acts as the software requesting information from the server.
2. **Server**:
   - The software that processes requests and returns data.
3. **UI vs. Data with Format**:
   - The **UI** is the visual representation, which **should not** be confused with the **data with format**.
   - The **data** is returned in a standard communication format like **HTML**, **JSON**, or **XML**.
   - These formats ensure that the communication between the client and server is standardized.
   - So, the data returned with these format HTML, JSON, or XML, not as a UI.

     HTML

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Grades</title>
</head>
<body>
    <table>
        <thead>
            <tr>
                <th>Course Name</th>
                <th>Course Teacher</th>
                <th>Grade</th>
            </tr>
        </thead>
        <tbody>
            <!-- > data is here, static or dynamic from backend <-->
            <tr>
                <td>Java</td>
                <td>Mohammed</td>
                <td>88</td>
            </tr>
            <tr>
                <td>Web</td>
                <td>Mohammed</td>
                <td>70</td>
            </tr>
        </tbody>
    </table>
</body>
</html>
<!--
    لازم يكون ملف الhtml جاهز عشان احط فيها data
    اما الjson&xml لا
 -->
```

JSON

```json
[
    {
        "course_name": "Java",
        "teacher_name": "Mohammed",
        "grade": 88
    },
    {
        "course_name": "Web",
```

```
        "teacher_name": "Mohammed",
        "grade": 70
    }
]
```

XML

```
<?xml version="1.0" encoding="utf-8" ?>

<grades>
    <grade>
        <course>Java</course>
        <teacher>Mohammed</teacher>
        <grade>88</grade>
    </grade>
    <grade>
        <course>Web</course>
        <teacher>Mohammed</teacher>
        <grade>70</grade>
    </grade>
</grades>
```

HTTP

HTTP Overview

Node.js Backend (Master Course) - #080 - Web & Web Development - HTTP

https://youtu.be/gV8Jy3eWkLQ?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Request-Response Operation

- **Responsible Party:**
  Not the Client or Server side; it's the
  Network Connection, which acts as the mediator.
  - The most common network connection is the **Internet**.

Protocol Definition

- A **protocol** is a set of rules governing data exchange or transmission between devices, e.g., **network protocols**.

Network Protocol

- Established rules that format, send, and receive data to enable communication between network endpoints (computers, servers, routers, etc.), regardless of infrastructure or design differences.
- Types of protocols used in request-response operations:
  - **HTTP** (HyperText Transfer Protocol)
  - **HTTPs** (Secure HTTP)

HTTP Responsibilities

1. Handling HTTP Requests
2. Handling HTTP Responses
3. Managing the type of connection

HTTP Request

Node.js Backend (Master Course) - #083 - Web & Web Development - HTTP Request - (1)

https://youtu.be/aHD340qzCkI?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Node.js Backend (Master Course) - #084 - Web & Web Development - HTTP Request - (2)

https://youtu.be/emBILkt8thg?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Structure of an HTTP Request:

1. **URI (Uniform Resource Identifier)**

   A unique identifier for a resource on the web, composed of:

   - **URL (Uniform Resource Locator)**: e.g., `https://website`
   - **URN (Uniform Resource Name)**: e.g., `/login`
   - `URI[https://website/login]` = `URL[https://website]` + `URN[/login]`
   - I L N ⇒ ILN

2. **Headers**: Metadata about the request (معلومات عن الطلب)

   Common headers include:

   - `user-agent`: Automatically sent, identifies the client.
   - `content-type`: Specifies the type of data in the body (e.g., JSON, XML).

3. **Body**: Key-Value (K-V) data sent with the request (المكان الذي أرسل فيه البيانات)

   Body forms depend on the `content-type` specified in the headers:

   - **form-data (default)**: For standard K-V pairs.
   - **form-data/multipart**: Used for file uploads.
   - **www-url-encoded**: Lightweight encoding for sending form data (rarely used, similar to `formData`).
   - **json, xml, plaintext**: Structured or unstructured data formats.

HTTP Response

Node.js Backend (Master Course) - #085 - Web & Web Development - HTTP Response

https://youtu.be/bd5Clor-uus?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Structure:

1. **(No URI)**

   HTTP responses do not include a URI since they are sent back to the client in response to a specific request.

2. **Headers:**

   Metadata about the response, similar to request headers.

   - `content-type`: Indicates the format of the response body (e.g., JSON, XML, HTML, plaintext).

3. **Response Body:**

   Contains the actual data being sent back to the client.

   - Body formats (defined by the `content-type` header):
     - **JSON**, **XML**, **plaintext**, **HTML** (the most common).
     - These formats act as **standard communication languages** between the server and the client.

HTTP Methods

Node.js Backend (Master Course) - #086 - Web & Web Development - HTTP Method - (1)

https://youtu.be/bGf1An-gbEw?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Node.js Backend (Master Course) - #087 - Web & Web Development - HTTP Method - (2)

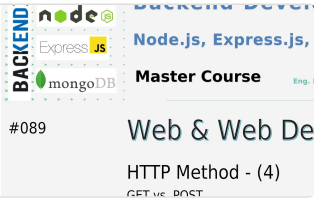
https://youtu.be/TTLIc1yxMl8?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Node.js Backend (Master Course) - #088 - Web & Web Development - HTTP Method - (3) - GET vs. POST

https://youtu.be/tJZIKiKIKJA?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Node.js Backend (Master Course) - #089 - Web & Web Development - HTTP Method - (4) - GET vs. POST

https://youtu.be/eXhvxlUGn0o?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H



Node.js Backend (Master Course) - #093 - Web & Web Development - HTTP Method - (7) - Other Methods

https://youtu.be/vylGdiFCBqo?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

GET VS POST

Aspect	GET	POST
Data Location	Data is sent in the **URI** as a query string. (Automatic)	Sent in the body and URI. (Automatic)
Visibility	Information is **not hidden** (visible in URI).	Information is **hidden** (in body).
Example URI	`http://www.google.com?search_key=coffee&price=40`	Not applicable. Data in body.
Headers Include	Headers include the query string and body.	Headers include the query string and body.
Body Access	**Cannot** access the body.	Data in both the URI and the body.
Data Size	Small (lightweight requests).	Large (e.g., file uploads).
Speed	Faster in handling requests.	Slightly slower due to larger data size.
Primary Use	Fetching data (e.g., product browsing, news viewing).	Modifying sensitive data (e.g., adding, updating, deleting).
Examples	Browsing products, viewing pages, reading news.	Adding: Upload photo/comment/reaction. Updating: Edit post/story/like. Deleting: Remove content.
Content Types	`JSON`, `XML`, `plaintext`, `HTML`.	`JSON`, `XML`, `plaintext`, `HTML`.

📝 **Notes:**

- Query string: `search_key=coffee&price=40`
  - `?` signifies the start of the query string.
- Both GET and POST can access the URI.
  - Only POST can access the body, GET can't.

Methods Summary

HTTP Method	Purpose
GET	Retrieve data (select).
POST	Add data (e.g., sensitive).
PUT	Update entire resource.
PATCH	Update part of a resource.
DELETE	Remove a resource.

Content Types for Requests/Responses

Type	Use Case
`form-data`	Requests.
`form-data/multipart`	Requests (e.g., files).
`www-url-encoded`	Requests (e.g., PUT).
`JSON`, `XML`, `plaintext`, `HTML`	Both requests and responses.

📝 **Notes:**

- You can't use PUT with `form-data`, `form-data/multipart`. الداتا بتصل فاضية
- `form-data` and `www-url-encoded` are similar but differ in encryption.

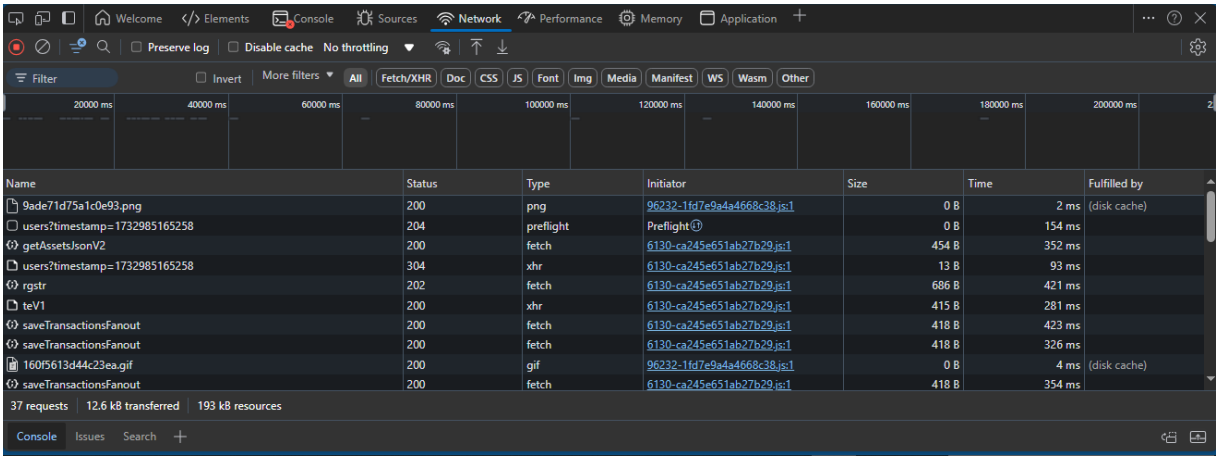
Dev Tools (Inspect Requests)

You may need to watch this lecture.

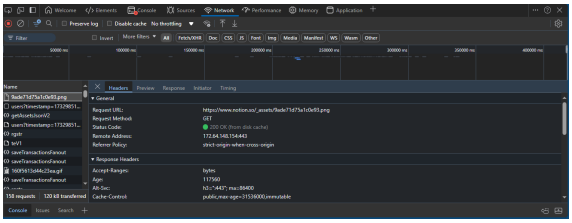
Node.js Backend (Master Course) - #090 - Web & Web Development - HTTP Method - (5) - Dev. Tools

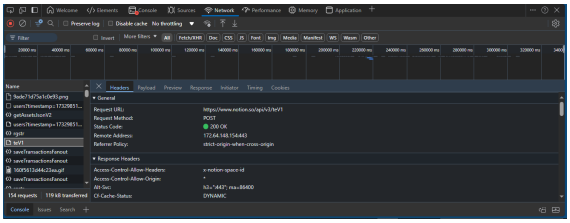
https://youtu.be/cVAUOfpE_Fk?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Open Inspect then Network Tab: You will view all requests.



Then click on one of the requests, you will see some available tabs: (the data in the website)





- **Headers**: Infos about Request Headers & Response Headers.

- **Payload**: Contains POST body.
- **Preview**: Renders response if it's HTML.
- **Response**: Displays the response in any format.

📝 **Note:** GET parameters are visible in the `Request URL` under the Headers tab.

Status Code

Node.js Backend (Master Course) - #091 - Web & Web Development - HTTP Method - (6) - Status Code

https://youtu.be/HZaiCk8vCVc?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

HTTP response status codes - HTTP | MDN

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

List of HTTP status codes

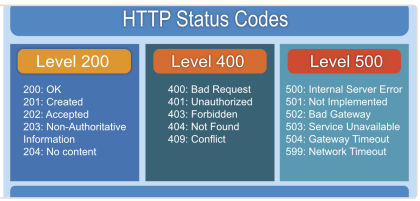
This is a list of Hypertext Transfer Protocol (HTTP) response status codes. Status codes are issued by a server in response to a client's request made to the server. It includes codes from IETF Request for Comments (RFCs), other specifications, and some additional codes used in some common applications of

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

HTTP Status Codes

HTTP specification defines these standard status codes divided into five categories that can be used to convey the results of a client's request.

https://restfulapi.net/http-status-codes/



`1xx` Informational Responses (100–199) [Processing]

These indicate that the request was received and understood, and the process is continuing.

- **100 Continue**
- **101 Switching Protocols**
- **102 Processing (WebDAV)** → Example: Payment processing.
- **103 Early Hints**

`2xx` Successful Responses (200–299) [Success]

These signify that the request was successfully received, understood, and accepted.

- **200 OK**
- **201 Created** (e.g., adding via POST).
- **202 Accepted**
- **203 Non-authoritative Information**
- **204 No Content**
- **205 Reset Content**
- **206 Partial Content**

- **207 Multi-Status (WebDAV)**
- **208 Already Reported (WebDAV)**
- **226 IM Used**

`3xx` Redirects (300–399) [Redirects to another link]

These inform the client that further action is needed to complete the request.

- **300 Multiple Choices**
- **301 Moved Permanently**
- **302 Found**
- **303 See Other**
- **304 Not Modified**
- **305 Use Proxy**
- **306 Switch Proxy** → Deprecated and no longer used.
- **307 Temporary Redirect**
- **308 Permanent Redirect**

`4xx` Client Errors (400–499) [Front End]

These indicate errors caused by the client.

- **400 Bad Request** (frontend issue)
- **401 Unauthorized**
- **402 Payment Required** → Reserved for future use.
- **403 Forbidden**
- **404 Not Found** (missing resource).
- **405 Method Not Allowed**
- **406 Not Acceptable**
- **407 Proxy Authentication Required**
- **408 Request Timeout**
- **409 Conflict**
- **410 Gone**
- **411 Length Required**
- **412 Precondition Failed**
- **413 Payload Too Large**
- **414 URI Too Long**
- **415 Unsupported Media Type**

- **416 Range Not Satisfiable**
- **417 Expectation Failed**
- **418 I'm a teapot** → A humorous status code defined in RFC 2324.
- **421 Misdirected Request**
- **422 Unprocessable Entity (WebDAV)**
- **423 Locked (WebDAV)**
- **424 Failed Dependency (WebDAV)**
- **425 Too Early**
- **426 Upgrade Required**
- **428 Precondition Required**
- **429 Too Many Requests**
- **431 Request Header Fields Too Large**
- **451 Unavailable For Legal Reasons**

`5xx` Server Errors (500–599) [Back End]

These signify that the server failed to fulfill a valid request.

- **500 Internal Server Error** (Backend Issue)
- **501 Not Implemented**
- **502 Bad Gateway**
- **503 Service Unavailable**
- **504 Gateway Timeout**
- **505 HTTP Version Not Supported**
- **506 Variant Also Negotiates**
- **507 Insufficient Storage (WebDAV)**
- **508 Loop Detected (WebDAV)**
- **510 Not Extended**
- **511 Network Authentication Required**

HTTPs

Node.js Backend (Master Course) - #092 - Web & Web Development - HTTPS

https://youtu.be/MPJe66xRM04?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

- **Definition**: HTTPS (HyperText Transfer Protocol Secure) is a protocol that ensures secure communication over a computer network.
  - **Key Feature**: Encrypts all data exchanged between the browser (client) and the server.
  - **Benefit**: Prevents third parties from intercepting or understanding the data (e.g., sensitive information like passwords or credit card details).
    - This means that if someone is trying to listen in on your communication, they won't be able to understand it.

When is POST More Secure than GET?

- The GET method is used to retrieve information from the given server using a given URI.
  Requests using GET should only retrieve data and should have no other effect on the data.
- The POST method is used to send data to the server, for example, customer information, file upload, etc.
  using HTML forms. So, The POST HTTP Method is more secure if the data is sensitive and if the protocol used is HTTPS.
- In one situation, POST is Safer than GET:
  - HTTPS = **HTTP + SSL (Secure Sockets Layer)**.
  - SSL encrypts the request body, ensuring that sensitive information cannot be viewed or intercepted.
  - كتير ناس بحكو انو بقدر اشوف الdata عن طريق الuri اذا استخدمت الGET
    هذا الكلام صحيح بس برضو بقدر اشوف الdata عن طريق الDevTools اذا استخدمت الPOST
    في حال الHTTPS: بتم تشفيير الBody وهيك بنقدرش نشوف الBody حتى من الDevTools وهذا هو الصح
- **GET Example**:
  - URI exposes query parameters, e.g., `http://example.com?user=JohnDoe&password=12345`.
  - Risks: Visible in browser history, bookmarks, and logs.
- **POST Example**:
  - Data is sent in the body and encrypted over HTTPS.
  - Risks: Data is still visible in developer tools (DevTools) but encrypted in transit.

📝 **Conclusion**: Use **POST with HTTPS** for secure transmission of sensitive data.

Feature	HTTP	HTTPS
Full Form	HyperText Transfer Protocol	HyperText Transfer Protocol Secure
Encryption	No encryption.	Encrypted using SSL/TLS.
Security	Data is vulnerable to interception.	Data is secure from interception.
Use Case	Non-sensitive data transfer.	Sensitive data (e.g., payments, forms).

SSL VS TSL

- **SSL**: Secure Sockets Layer
- **TLS**: Transport Layer Security
- **SSL**: An older protocol used to encrypt communication between a client (browser) and server.
- **TLS**: The successor to SSL, offering improved security and performance. It is the modern standard for secure communication.

> Note: Although "SSL" is often mentioned, most secure connections today use TLS.

API

Node.js Backend (Master Course) - #094 - Web & Web Development - API - (1)

https://youtu.be/khWkzFmqw28?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Node.js Backend (Master Course) - #095 - Web & Web Development - API - (2)

https://youtu.be/7IRsiSP0kxc?list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

Introduction

- **API** stands for **Application Programming Interface**.
- The server provides data, and the client or mobile application uses its own UI to interact with that data. The client fetches data from the server but does not handle the business logic or data processing.
  - **Web API**: A type of API specifically for web applications.
  - **Mobile Client**: In mobile applications, the client (like the mobile device) only fetches data from the backend server. It doesn't manage data itself.
- **Why is API named like this?**
  - The name "API" comes from the fact that the server only handles data, while the client or mobile app handles the UI and retrieves data.
  - **API as a Data Channel**: The API serves as a channel for data communication between the client and server.
- When there are no interfaces or data returned, it's considered part of the **Backend** (server-side).
- **Online Payment Systems**: In rare cases, APIs may also return the UI (front-end), especially in online payment systems.
- **Web View**: In some cases, an API might return the UI as well.

Types of APIs:

1. **REST API / RESTFul API**:
   - **Definition**: REST (Representational State Transfer) is an architectural style for designing networked applications.
   - **Return Format**: REST APIs return data in **JSON** or **XML** format, as opposed to traditional backends which typically return HTML.
   - **Technologies Used**:
     - **PHP**, **Node.js, NestJS**, **Python**, **Java**
   - **Most Common Data Format**: JSON
2. **SOAP Web Service**:
   - **Definition**: SOAP (Simple Object Access Protocol) is a protocol for exchanging structured information in web services.
   - **Communication**: SOAP APIs use HTTP/S for communication.
   - **Key Features**: SOAP is more complex than REST and supports more protocols.
   - **Technologies Used**: Primarily **Java**.
   - **Most Common Data Format**: **XML**

SOAP vs. REST

Feature	SOAP Advantages	REST Advantages
Platform Independence	SOAP works across different platforms and languages.	REST typically operates over HTTP, making it simpler but less platform-agnostic.
Use in Distributed Systems	Ideal for complex enterprise environments requiring intricate workflows.	Better suited for simpler, direct point-to-point communication scenarios.
Standardization	Comes with built-in standards like WS-Security for integrity and confidentiality.	Minimal standards allow for greater flexibility and customization.
Error Handling	Provides robust built-in error handling mechanisms.	Error handling is less standardized and depends on implementation.
Automation	Some tools can auto-generate code, streamlining integration.	Lacks built-in automation, but easier to implement without specialized tools.
Ease of Use	More complex to use and set up.	Simpler and more flexible compared to SOAP.
Tools Requirement	Often requires specialized tools for integration.	Requires no expensive tools, making it cost-effective.
Learning Curve	Steeper learning curve due to its complexity and standards.	Easier to learn and more intuitive.
Efficiency	Uses XML for all messages, which can be verbose.	Can use lighter message formats like JSON, making it more efficient.

Speed	Slower due to extensive processing requirements.	Faster as it avoids extensive processing.
Web Technology Alignment	Not inherently aligned with standard web technology design.	Closely aligns with the design philosophy of web technologies like HTTP.

API - (2)

API (Application Programming Interface)

A service provider layer that facilitates communication between the **client side** and the **server side**.

1. Client Side

- **Mobile Applications**
- **Web Applications (SPA: Single Page Application)**:
  - **Frontend Frameworks**:
    - React.js
    - Vue.js
    - Angular.js

2. Server/Host Side

- **API Hosting**: Platforms or languages used to develop and host APIs.
- **Web Applications**:
  - **Backend Frameworks**:
    - Laravel
    - Yii
    - Slim
    - ASP.NET MVC
    - ASP.NET Core
    - Node.js
    - NestJS

Key Notes

1. **Client**: Represents operations like sending an image, message, or story.
2. **Server**: Receives and processes these operations.
3. APIs act as intermediaries for server operations.
4. APIs utilize web protocols (e.g., HTTP, FTP).

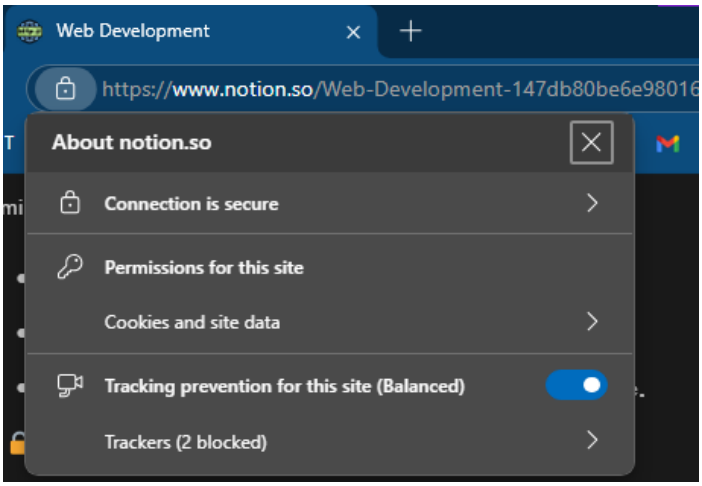
Web Protocols

▼ **1. [HTTP ⇒ 80 Port] || [🔒HTTPS ⇒ 443 Port]**

- HTTP: HyperText Transfer Protocol
- Essential for transferring data while browsing or interacting with APIs.
- **Unsecure Protocol** (data is not encrypted).
- Port: **80**.
- Common Use: Sending a request to upload an image.

🔒HTTPS ⇒ 443 Port

- HTTPS: HyperText Transfer Protocol Secure
- Encrypted and secure.
- Port: **443**.
- Icon Lock on the browser. [شهادة الحماية]



- Utilizes **SSL (Secure Socket Layer)** for data protection.
  - TLS (Transport Layer Security): A cryptographic protocol for secure connections.
  - **SSL/TLS** ensures secure communication.
  - SSL = TLS = SSL/TLS
  - [Communication Security, Cryptographic Protocol]
- Types:
  - HTTP Request: URI - Methods - Headers - Body
  - HTTP Response: Body - Status Code

▼ **2. FTP ⇒ 21 Port || [🔒SFTP ⇒ 22 Port]**

- FTP: File Transfer Protocol
- Used for file transfers.

- **Unsecure Protocol**.
- Port: **21**.
- Establishes a connection between a device and a server to transfer data/files.
- Domain has nothing to do with.

🔒**SFTP ⇒ 22 Port**

- SFTP: Secure File Transfer Protocol
- Encrypted and secure.
- Port: **22**.

▼ **Notes about domain**

A domain (e.g., `www.example.com`) represents:

1. **HTTP Request (Text):**

   A textual address that users type in the browser to access a website.

2. **IP Address:**

   The domain is mapped to an IP address, which is the server's address in the network.

3. **Cloudflare Technology:**
   - A service that manages operations related to system security and protection.
   - Primarily used to **protect domains** and enhance performance (e.g., handling DDoS attacks, caching).
4. **Hostinger Website:**
   - Offers web hosting plans tailored for agencies or individuals who need to host multiple websites for clients.

How a Domain is Linked to a Server

- **Nameservers (DNS):**
  - In **Hostinger**, each server is assigned **nameservers (DNS)**.
  - DNS records are configured to point the **domain** to the correct server.

Web Protocols and Ports

1. **HTTP (HyperText Transfer Protocol):**
   - Default port: **80**
   - Secure version (HTTPS): **443**
2. **FTP (File Transfer Protocol):**
   - Default port: **21**
   - Secure version (SFTP): **22**

> Note: Secure (S) ports are typically larger than their non-secure counterparts.

Key Notes

1. Domains rely on a mapping system (DNS) to link text-based names to IP addresses.
2. Cloudflare is a prominent technology for securing and managing domain traffic.
3. Hostinger is an example of a web hosting service that supports managing multiple websites.
4. HTTP and FTP are essential web protocols, each with secure versions for enhanced safety.

HTTP Request Structure

Generated by any interaction with a website or API.

Components:

1. **URI (Uniform Resource Identifier)**
   - Combination of **URL** (Locator) and **URN** (Name).
   - Example: `https://facebook.com/Ahmed`
     - URL: `https://facebook.com`
     - URN: `/Ahmed`
   - **Domain Components**:
     - **URL Example:** `https://blog.hubspot.com/what-is-a-domain`
       - **Protocol:** `https`
       - **Subdomain:** `blog`
         - Not all websites support subdomains, as mentioned.
       - **SLD (Second-Level Domain):** `hubspot`
       - **TLD (Top-Level Domain):** `com`
       - **Domain Name:** `hubspot.com`
       - **Page Path:** `/what-is-a-domain`
2. **Methods**
   - Define the purpose and type of operation:
     - **GET**: Fetch data. [Read Data]
     - **POST**: Save/process data (e.g., login or create an account).
     - **PUT/PATCH**: Update data.
       - **PUT**: Updates or creates new data if none exists. [Most Used]
       - **PATCH**: Updates only existing data.
     - **DELETE**: Removes data.
3. **Headers**
   - Key-value pairs describing the request.
   - Key is always string.
   - Common Keys:

- `Accept`: `application/json`
- `Content-Type`: `application/json`
- `Authorization`: Token used for user authentication. [TOKEN (تأشيرة)]
  - Token used to identify user in server side.
  - Each user has a token when logged in successfully.
  - Token has an expiration date defined by web developer.
  - Without the token, you can’t access web application.
  - Encryption.
  - Made in Server Side.

4. **Body** (if required)
   - Contains the data to be sent (used with **POST**, **PUT**, and **PATCH**).
   - Send data to server side.
   - Encryption Type.
   - Common Content Types:
     1. `x-www-form-urlencoded`:
        - Key: String.
        - Value: String.
     2. `form-data`: (multipart/form-data)
        - Key: String.
        - Value: String/File.

HTTP Response Structure

The server's response to a client request.

Components

1. **Body**
   - Contains received data in formats like **JSON** or **XML**.
   - JSON Example: [JavaScript Object Notation]

```
{
  "key": "value",
  "array": [1, 2, 3],
  "nestedObject": {"nestedKey": "nestedValue"}
}
```

2. **Status Code**
   - Numeric value indicating the result:

- **1xx**: Informational (Processing).
- **2xx**: Success.
- **3xx**: Redirection (e.g., link to link).
- **4xx**: Client error (frontend).
- **5xx**: Server error (backend).

Notes on Tools and Techniques

1. Postman

- A platform for designing, testing, and iterating APIs.

2. Cloudflare

- Protects and manages web domains.

3. Server and Domain Linking

- Done via **nameservers (DNS)**.
- Example: Linking `hostinger` DNS with your domain.

Server Response Representation

1. **Server-Side Response Encoding (تشفير):**
   - **Encode (تشفير):**
     - **Encoding (serialization):** Converting JSON (or any data format) into a string for transmission.
     - Converts the data from **JSON** to a **String** format for transmission.
2. **Client-Side Response Decoding (فك التشفير):**
   - **Decode (فك التشفير):**
     - **Decoding (deserialization):** Reverting the string back into JSON (or the original format) for use.
     - Converts the received **String** back to **JSON** format for processing.

JSON

Node.js Backend (Master Course) - #096 - Web & Web Development - JSON

https://www.youtube.com/watch?v=SRRoj0nqrqM&list=PLE5Mq0Nw_Flr1kMDtWmQYmAfURQpH3r8H

JSON Array Structure

1. **Empty JSON Array:**

```
[]
```

2. **Simple Array with Numbers:**

```
[0, 1, 2, 3, 4, 5]
```

3. **Array with Mixed Data Types:**

```
["StringValue", 10, 20.13, true, null]
```

4. **Nested Array and Object:**

```
[
  {
    "Name": "Nested Object"
  },
  [10, 20, true, 40, "Nested Array"]
]
```

The JSON array is a list of JSON data types.

JSON Object Structure

1. **Single Key-Value Pair:**

```
{
  "Name": "Object Name"
}
```

2. **Multiple Key-Value Pairs:**

```
{
  "Name": "Object Name",
  "Value": 10
}
```

3. **Nested Object:**

```
{
  "Name": "Object Name",
  "Value": 10,
  "NestedObject": {
    "Name": "Nested Object"
```

```
  }
}
```

4. **Nested Object and Array:**

```
{
  "Name": "Object Name",
  "Value": 10,
  "NestedObject": {
    "Name": "Nested Object"
  },
  "NestedArray": [10, 20, true, 40, "Nested Array"]
}
```

JSON Key and Value Characteristics

- **Key:** Always a string enclosed in double quotes (`""`).
- **Value:** Can be any JSON data type:
  - `String`
  - `Number`
  - `Boolean`
  - `null`
  - `Array`
  - `Object`

Comparison Between JavaScript Object and JSON

In JavaScript:

```
const person = {
  name: 'John',
  age: 30,
  hobbies: ['music', 'movies', 'sports'],
  address: {
    street: '50 main st',
    city: 'Boston',
    state: 'MA'
  }
};
```

In JSON (Equivalent):

```
{
  "name": "John",
  "age": 30,
  "hobbies": ["music", "movies", "sports"],
  "address": {
    "street": "50 main st",
    "city": "Boston",
    "state": "MA"
  }
}
```

Automatic JSON Conversion

- When sending data to a server, JavaScript objects are automatically converted to JSON using `.json()` methods.
- **Conversions:**
  - Any `Key: Value` pairs become `JSON Objects`.
  - `Lists` become `JSON Arrays`.