TECNICATURA SUPERIOR EN

Telecomunicaciones

Proyecto Integrador I

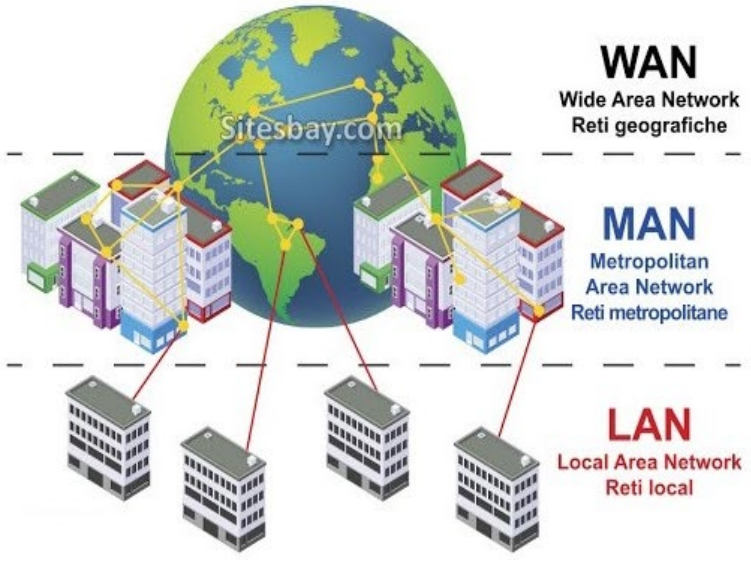
Rest Ful APIs – parte I

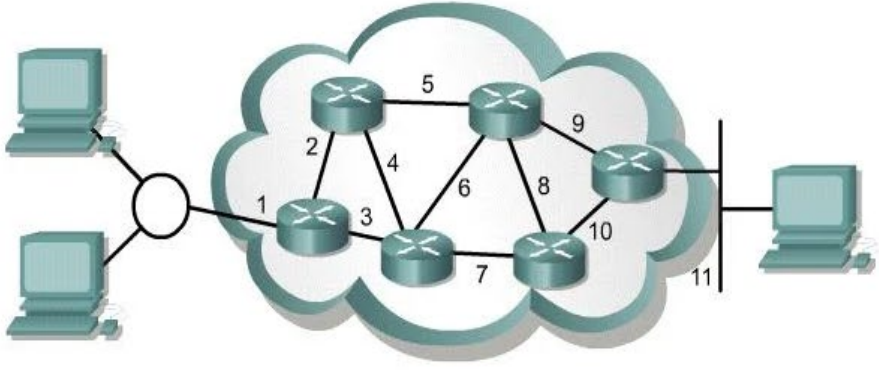
ISPC
INSTITUTO SUPERIOR POLITÉCNICO CÓRDOBA

Dirección General de EDUCACIÓN TÉCNICA Y FORMACIÓN PROFESIONAL
Ministerio de EDUCACIÓN
CÓRDOBA entre todos

Cómo Funciona Internet

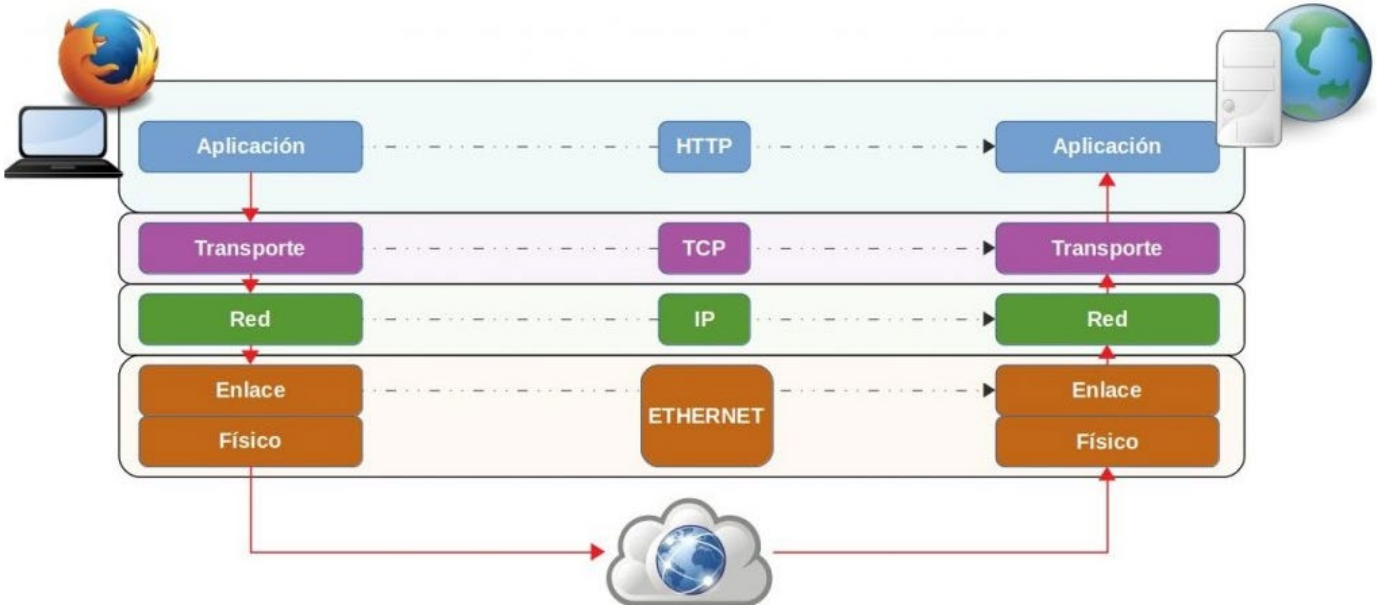
Internet, en su esencia, es una red global de redes. Esta estructura descentralizada permite la comunicación entre todos los dispositivos conectados. Los protocolos desempeñan un papel crucial en esta comunicación, proporcionando las reglas que definen cómo se deben enviar y recibir los datos. Uno de los protocolos fundamentales es el Protocolo de Internet o IP, que permite el envío de paquetes de información de un dispositivo a otro. Cada dispositivo tiene una dirección IP única, que puede ser estática o dinámica. En una interacción típica de Internet, un dispositivo cliente envía una solicitud a un servidor utilizando su dirección IP. El servidor, a su vez, responde con la información o los recursos solicitados.





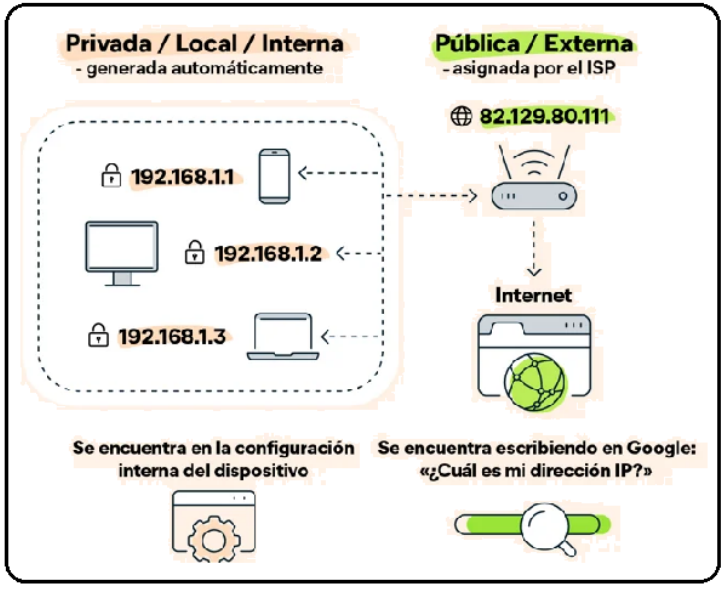
Concepto de Internet:

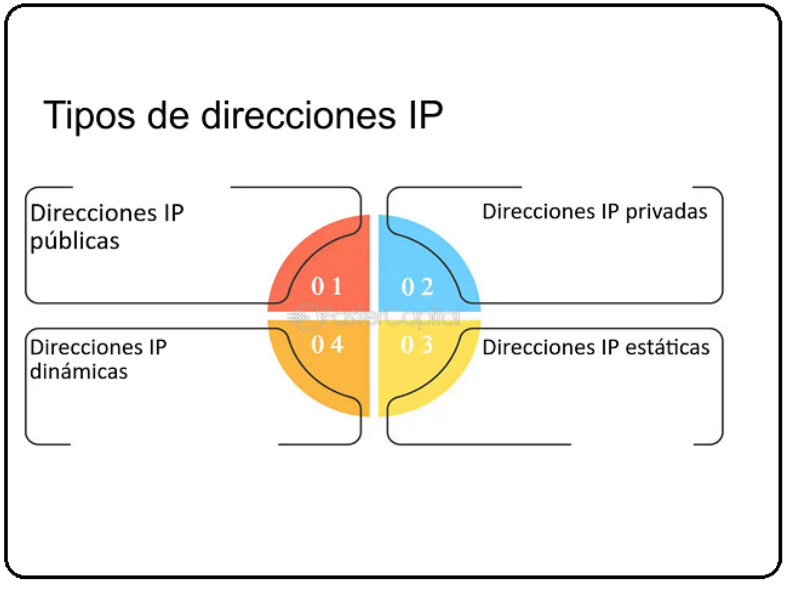
- Internet es una red global de redes que permite la comunicación entre dispositivos conectados.
- Estructura descentralizada que utiliza el protocolo de Internet (IP) para enviar paquetes de información de un dispositivo a otro.



Protocolos en Internet:

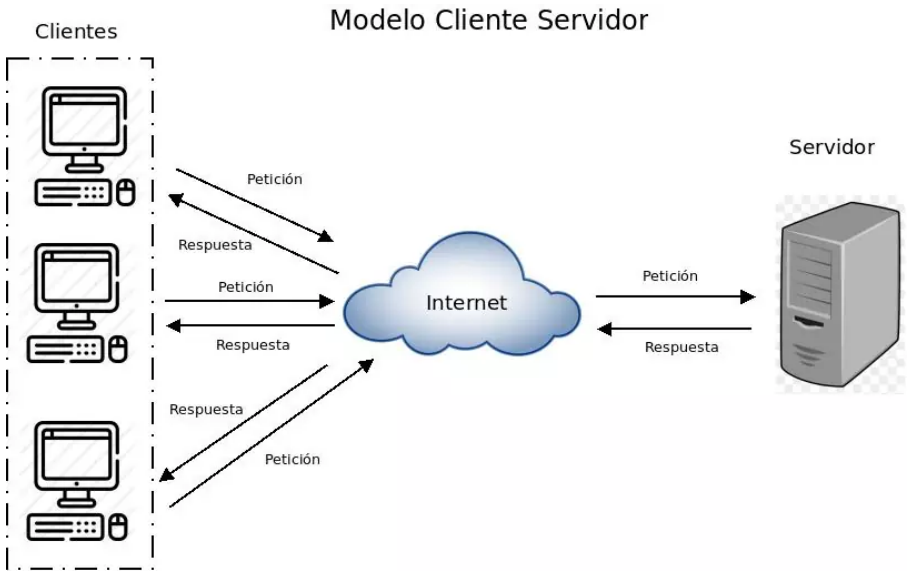
- Los protocolos definen cómo se deben enviar y recibir los datos en la red.
- Los protocolos importantes incluyen el Protocolo de Internet (IP), el Protocolo de Control de Transmisión (TCP) y el Protocolo de Transferencia de Hipertexto (HTTP).





Direcciones IP:

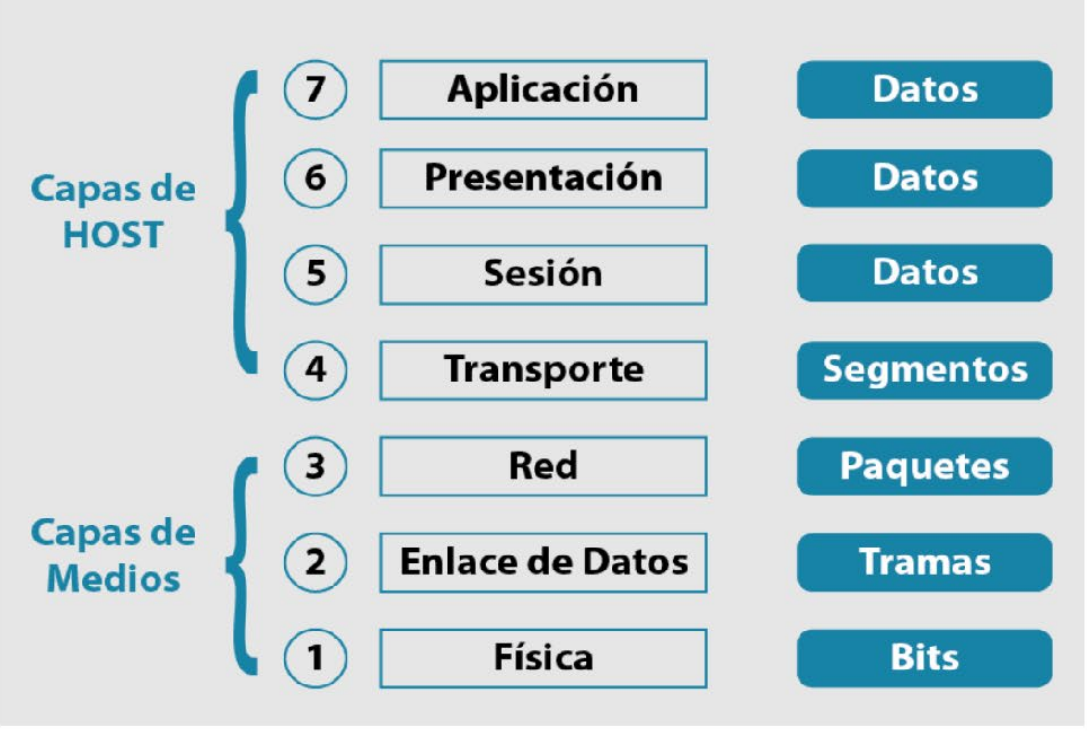
- Cada dispositivo en Internet tiene una dirección IP única que se utiliza para enviar y recibir información.
- Las direcciones IP pueden ser estáticas (permanentes) o dinámicas (cambian con el tiempo).



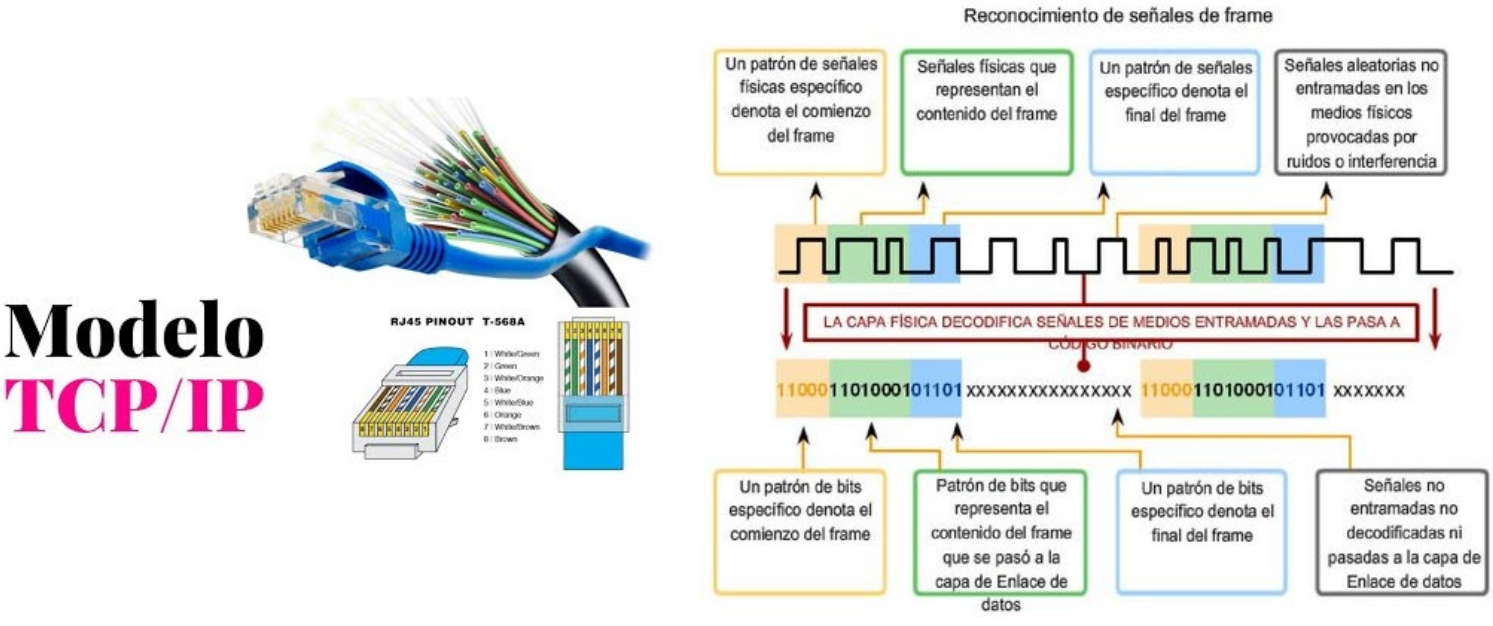
Servidores y Clientes:

- Los servidores son computadoras que almacenan información y recursos. Los clientes son dispositivos que solicitan estos recursos.
- Una interacción típica de Internet implica que un cliente envíe una solicitud a un servidor, que luego responde con la información o los recursos solicitados.

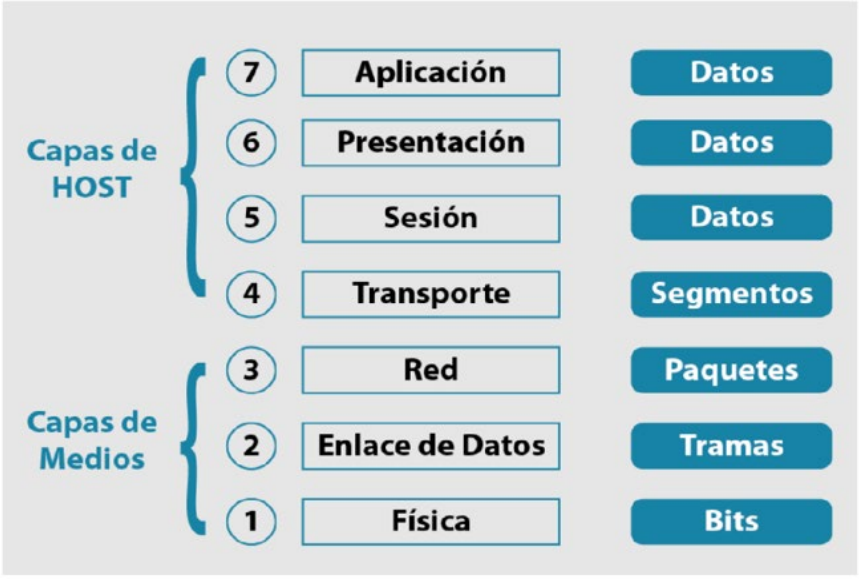
El Modelo de Interconexión de Sistemas Abiertos (OSI)



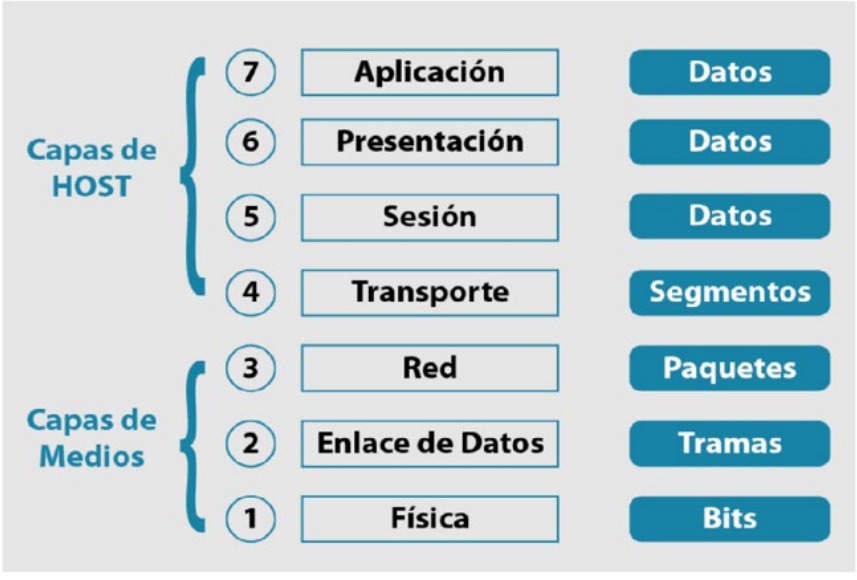
El Modelo OSI es una descripción conceptual que explica cómo las diferentes tecnologías y protocolos interactúan para proporcionar servicios de red. Aunque en la práctica, muchos protocolos de red no siguen estrictamente las siete capas del modelo OSI, este sigue siendo útil para entender y describir cómo funcionan las redes.



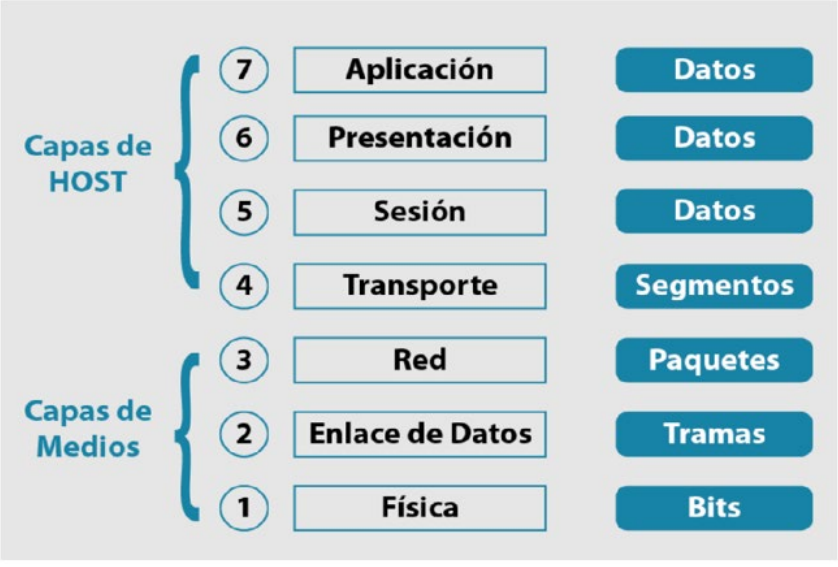
- **Capa 1 - Física:** Es la capa más baja y se encarga de la transmisión y recepción de datos brutos a través del medio físico de red. Esto incluye las características del cableado, las señales, las conexiones y la transmisión de bits en bruto.



- **Capa 2 - Enlace de datos:** Esta capa administra la transmisión de datos entre dispositivos en la misma red. Controla la forma en que los dispositivos acceden a los medios y manejan los errores en el nivel físico.
- **Capa 3 - Red:** Se encarga del enrutamiento de paquetes de datos a través de múltiples redes. Maneja cuestiones como la asignación de direcciones y la gestión del tráfico.



- **Capa 4 - Transporte:** Proporciona una transferencia de datos confiable y segura entre los sistemas, gestionando el control de flujo, el control de errores y la segmentación de datos.
- **Capa 5 - Sesión:** Gestiona la creación, el mantenimiento y la terminación de las sesiones de comunicación entre las aplicaciones.



- **Capa 6 - Presentación:** Esta capa gestiona la representación y codificación de datos. Traduce los datos entre el formato utilizado por las aplicaciones y el formato utilizado en la red.
- **Capa 7 - Aplicación:** Es la capa que interactúa directamente con las aplicaciones de software. Incluye servicios como correo electrónico, transferencia de archivos y administración de recursos.

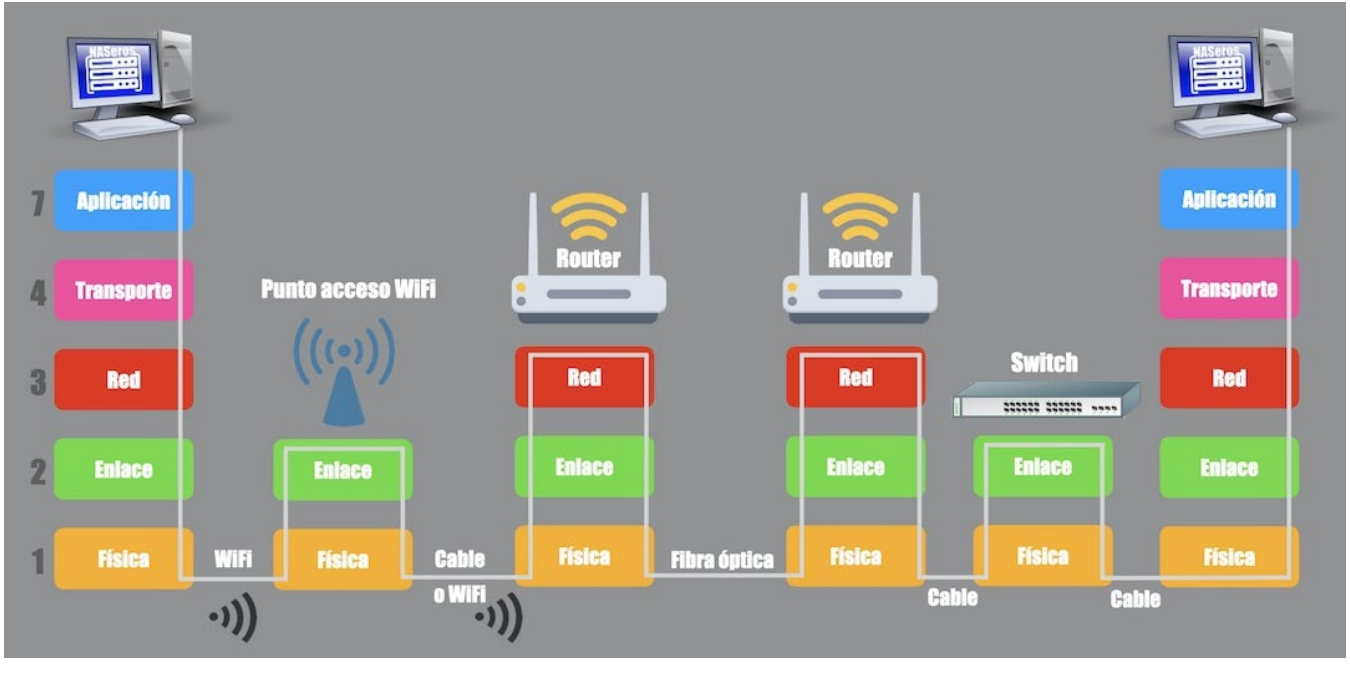
Cómo Funciona Internet

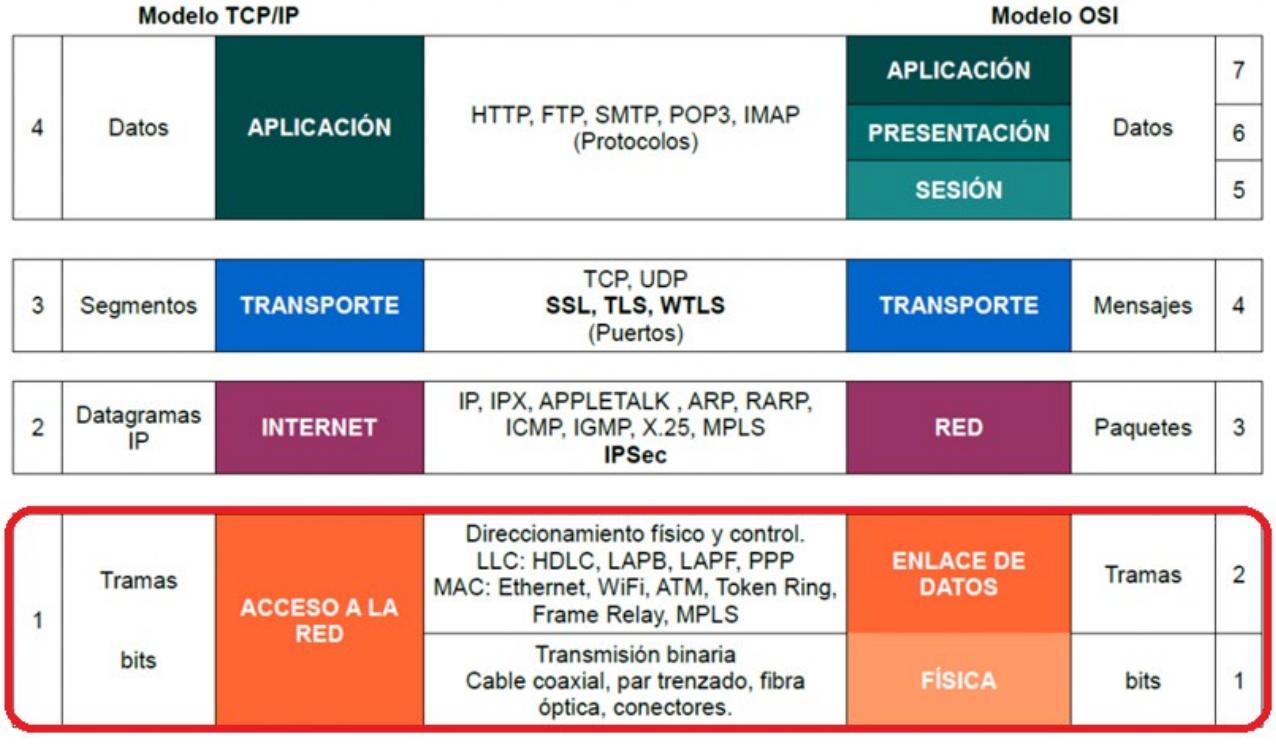
Modelo TCP/IP				Modelo OSI		
4	Datos	APLICACIÓN	HTTP, FTP, SMTP, POP3, IMAP (Protocolos)	APLICACIÓN	Datos	7
				PRESENTACIÓN		6
				SESIÓN		5
3	Segmentos	TRANSPORTE	TCP, UDP **SSL, TLS, WTLS** (Puertos)	TRANSPORTE	Mensajes	4
2	Datagramas IP	INTERNET	IP, IPX, APPLETALK , ARP, RARP, ICMP, IGMP, X.25, MPLS **IPSec**	RED	Paquetes	3
1	Tramas	ACCESO A LA RED	Direccionamiento físico y control. LLC: HDLC, LAPB, LAPF, PPP MAC: Ethernet, WiFi, ATM, Token Ring, Frame Relay, MPLS	ENLACE DE DATOS	Tramas	2
	bits		Transmisión binaria Cable coaxial, par trenzado, fibra óptica, conectores.	FÍSICA	bits	1

Capas de Protocolo:

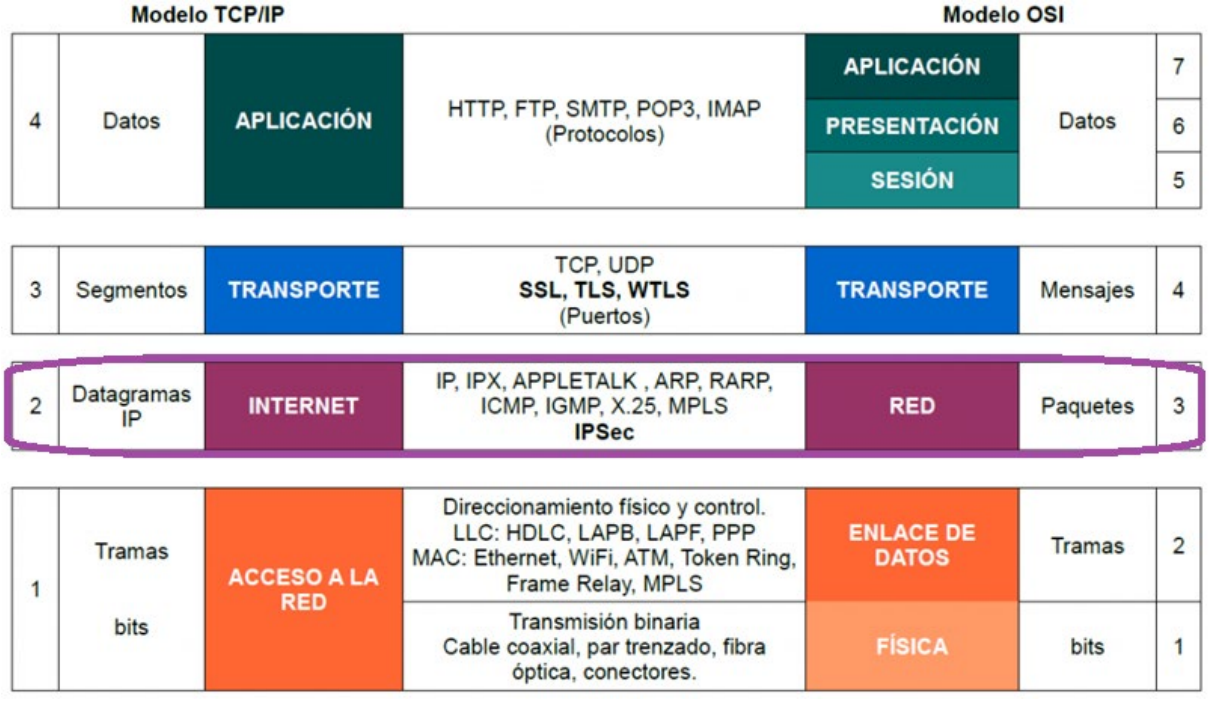
Los protocolos de Internet operan en una arquitectura de capas, con cada capa proporcionando diferentes funcionalidades.

Las cuatro capas principales son: Capa de Acceso a la Red, Capa de Internet (IP), Capa de Transporte (TCP/UDP), y Capa de Aplicación (HTTP, FTP, etc.).





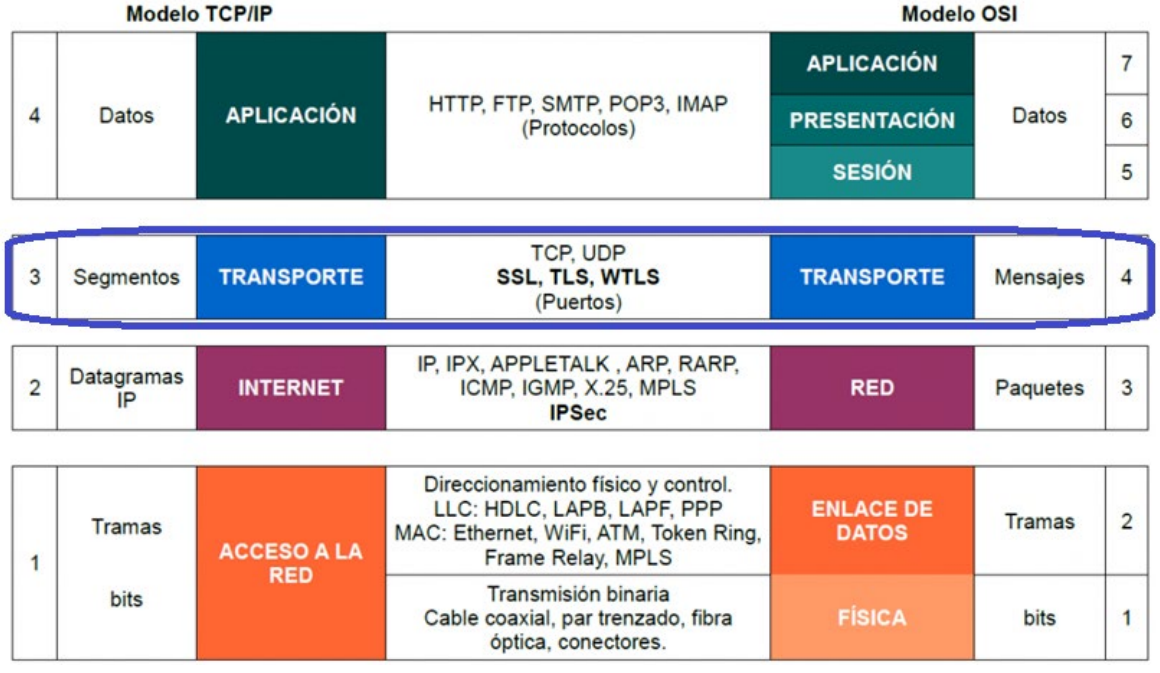
1. **Capa de Acceso a la Red:**
   Se ocupa de la conexión física a la red y la transmisión de datos en bruto. Incluye tecnologías como Ethernet y Wi-Fi.



2. **Capa de Internet (IP):**

   Responsable de la dirección y enrutamiento de paquetes.

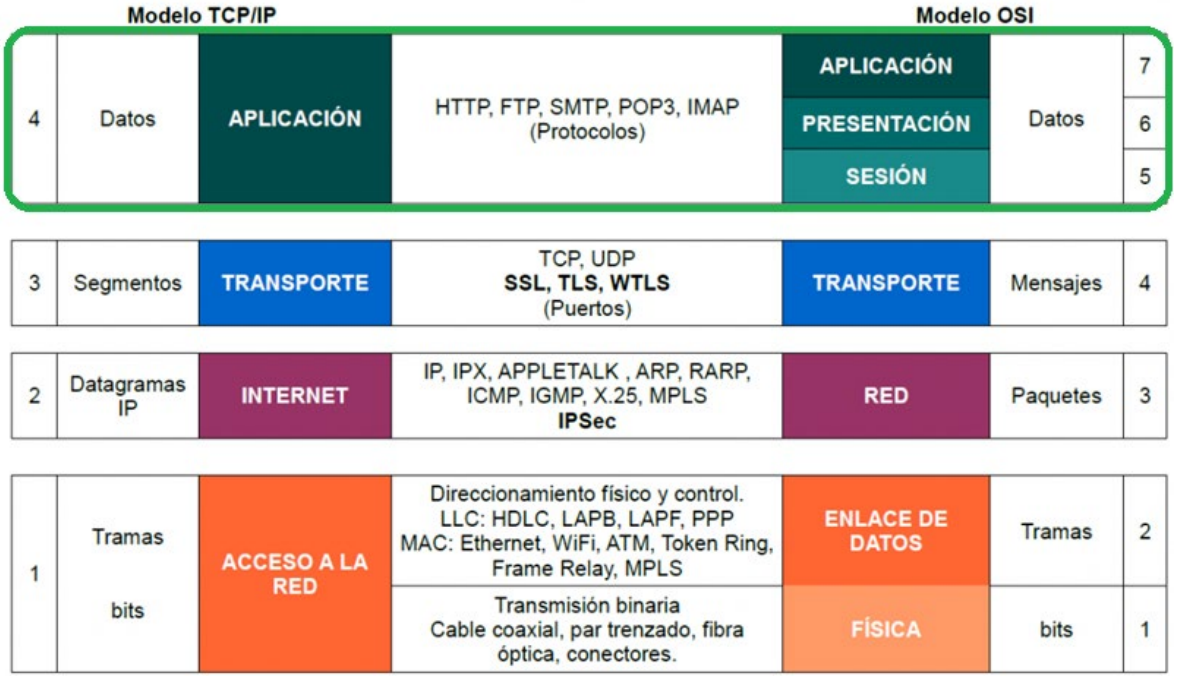
   Utiliza direcciones IP para identificar dispositivos y rutas de paquetes a través de la red.



3. **Capa de Transporte (TCP/UDP):**

   Facilita la comunicación entre aplicaciones en dispositivos.

   TCP se ocupa de la entrega confiable de paquetes, mientras que UDP es más rápido pero menos confiable.



4. **Capa de Aplicación:**

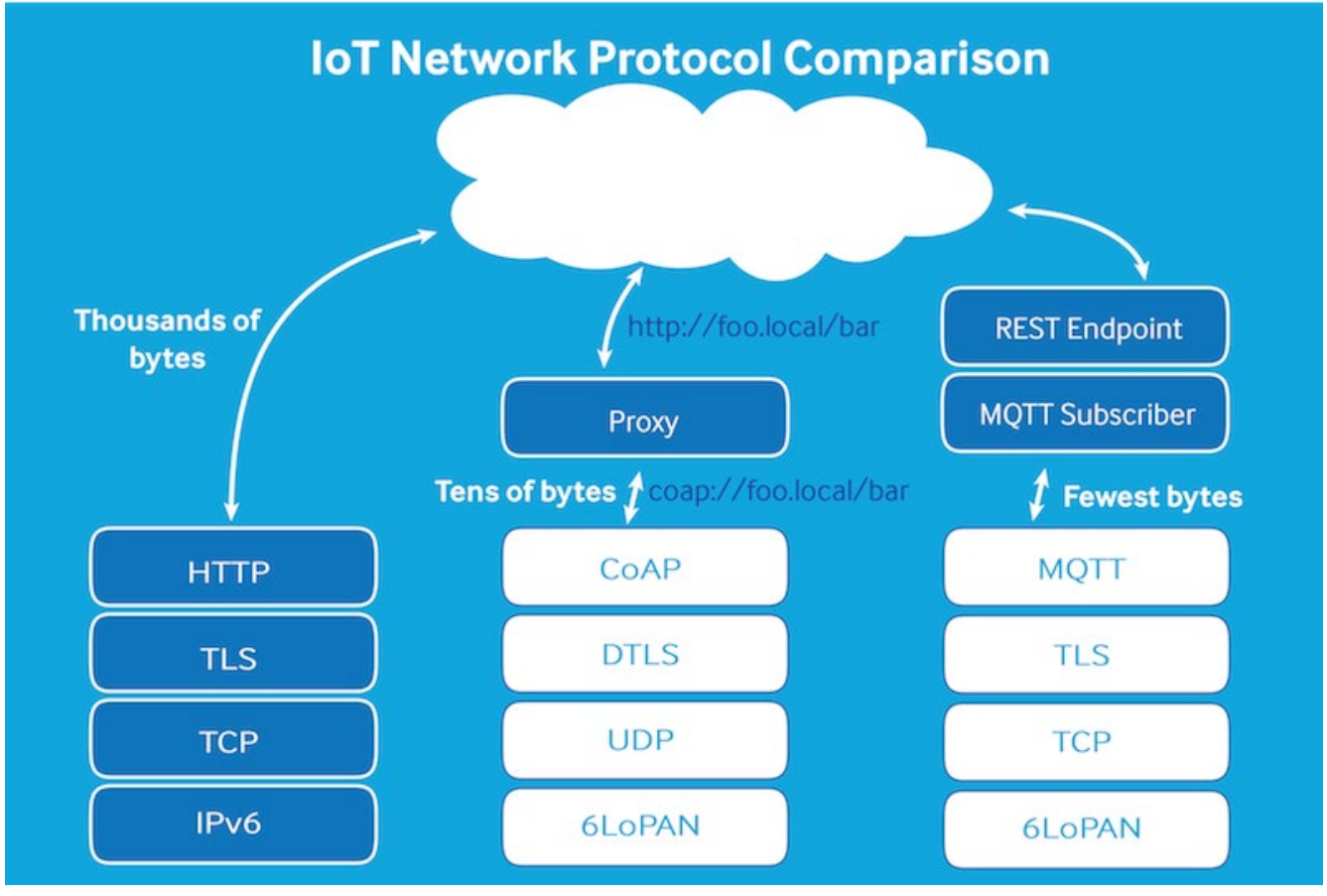
   Donde los protocolos como HTTP y FTP operan.

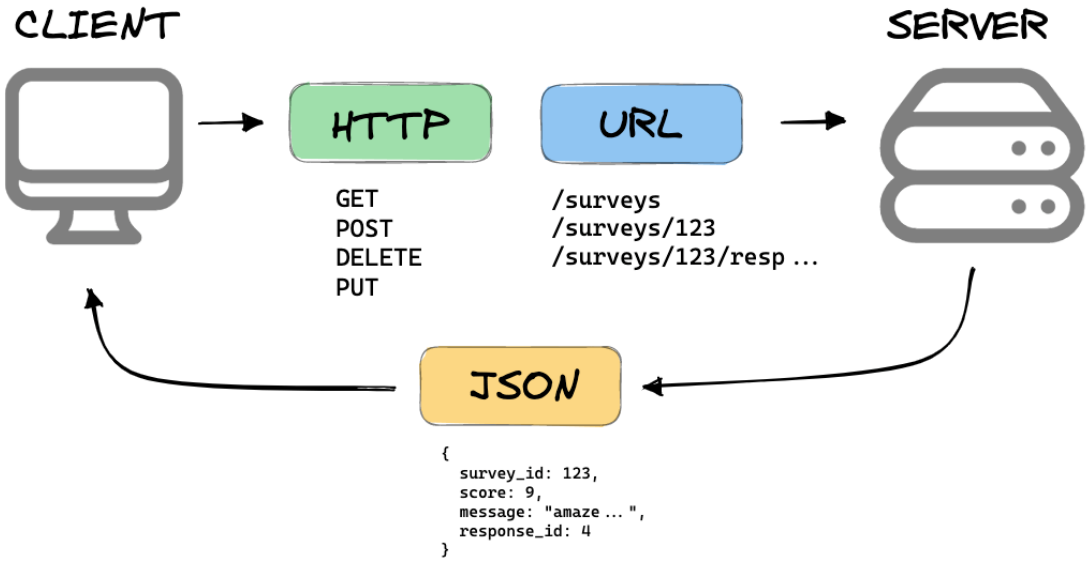
   Define cómo las aplicaciones en los dispositivos se comunican y cómo los datos son formateados para la transmisión.

El WWW y su Relación con HTTP

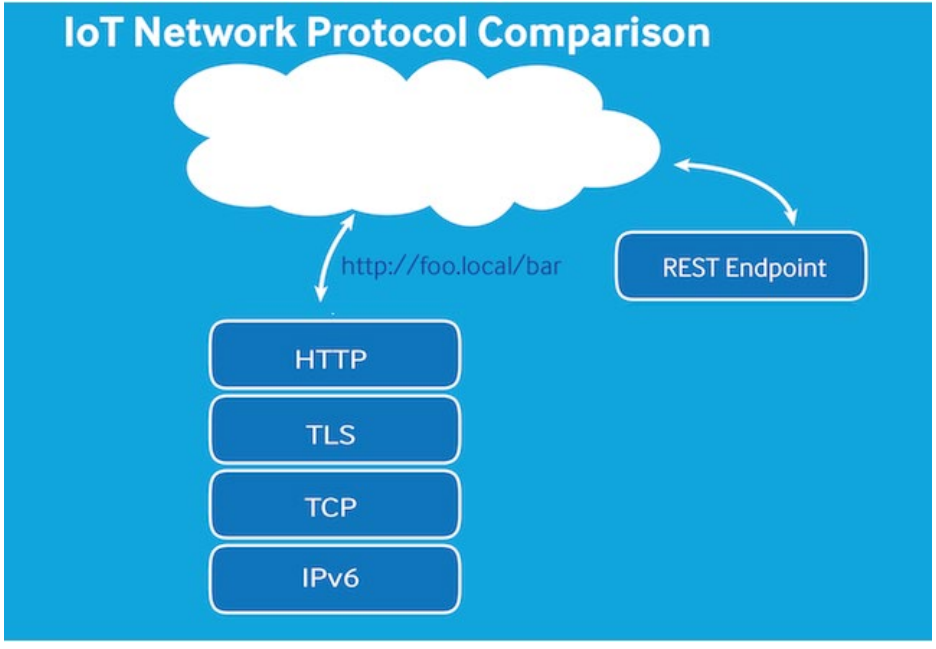
- **El World Wide Web (WWW):**
  El WWW, o simplemente la web, es un sistema de información en Internet que permite a los documentos y otros recursos web ser enlazados y accesibles a través de una dirección Uniform Resource Locator (URL).
  Los recursos en la web pueden ser accedidos y transmitidos utilizando HTTP.
- **HTTP y la Web:**
  HTTP es la base de cualquier intercambio de datos en la web.
  HTTP permite la transmisión de hipertexto y otros recursos entre el cliente y el servidor.
- **Navegación Web y HTTP:**
  Al navegar por la web, cada clic en un enlace o envío de un formulario desencadena una petición HTTP.
  Las respuestas HTTP del servidor pueden incluir contenido como texto HTML, imágenes, scripts, etc.

Los protocolos web en IoT

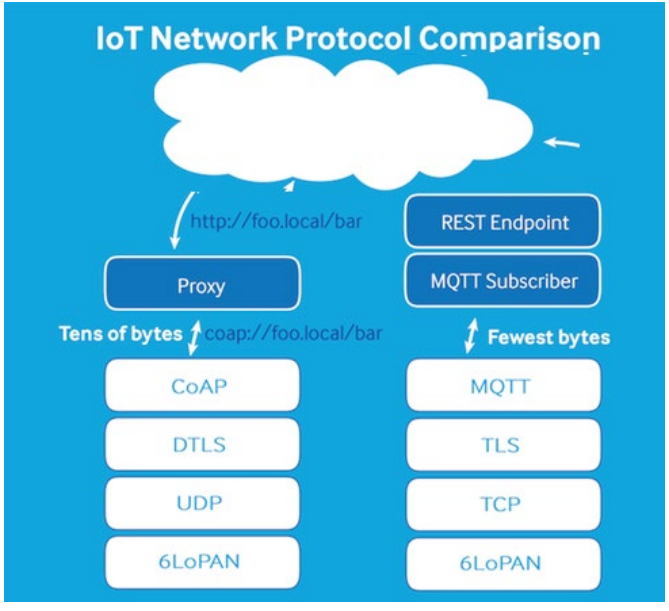




En el contexto del IoT, el uso de protocolos y estándares de comunicación adecuados puede establecer diferencias en el rendimiento, la escalabilidad y la eficiencia del sistema. Ahora veremos algunos conceptos claves: HTTP, JSON y REST APIs, que desempeñan roles fundamentales en la facilitación de la comunicación en los sistemas IoT.



- **1. HTTP (Protocolo de Transferencia de Hipertexto):** Es un protocolo de nivel de aplicación utilizado para transmitir hipermedia en la World Wide Web. Aunque no fue diseñado específicamente para IoT, se utiliza ampliamente debido a su simplicidad y su amplio soporte. HTTP es un protocolo de petición-respuesta, lo que significa que un cliente envía una petición y un servidor envía una respuesta.



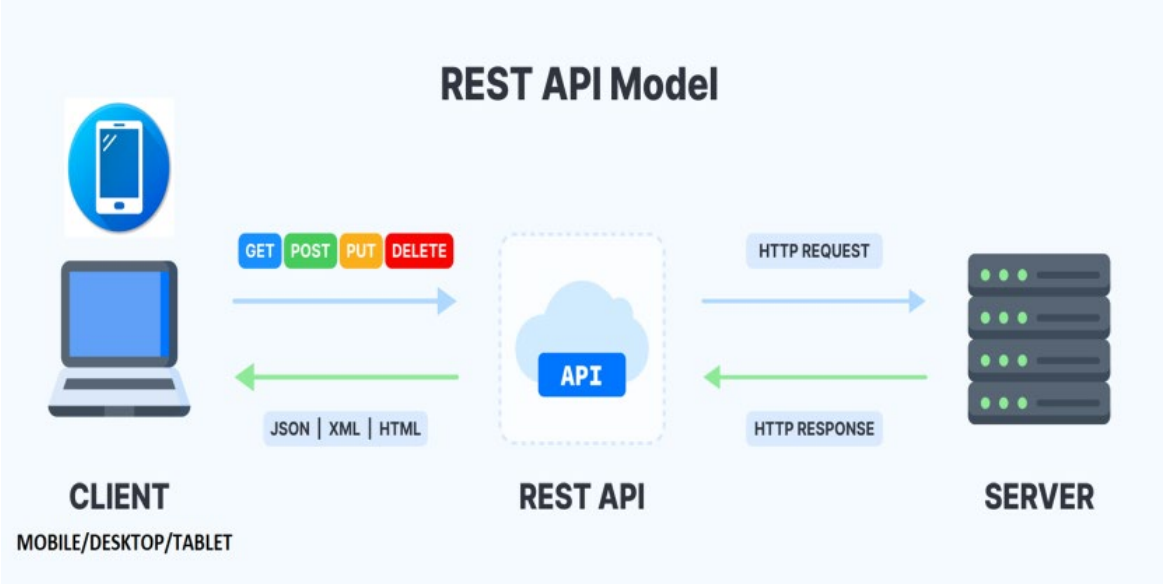
- **2. MQTT (Message Queue Telemetry Transport):** Es un protocolo de mensajería ligero diseñado para sensores y dispositivos móviles con recursos limitados. Es especialmente útil para situaciones donde se requiere una baja latencia y un consumo mínimo de ancho de banda y batería. MQTT utiliza un modelo de publicación/suscripción, en el que los dispositivos publican datos en "temas" y otros dispositivos se suscriben a esos temas para recibir los datos.

XML vs. JSON

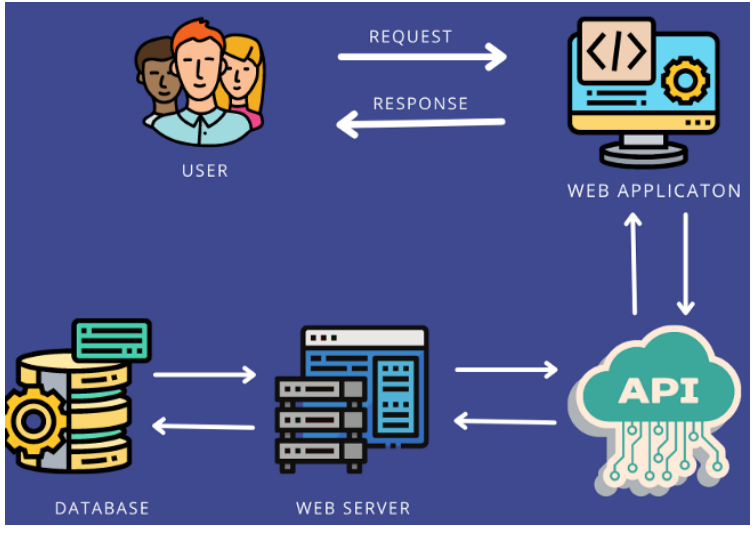
```xml
<?xml version="1.0" encoding="UTF-8"?>
<endereco>
    <cep>31270901</cep>
    <city>Belo Horizonte</city>
    <neighborhood>Pampulha</neighborhood>
    <service>correios</service>
    <state>MG</state>
    <street>Av. Presidente Antônio Carlos, 6627</street>
</endereco>
```

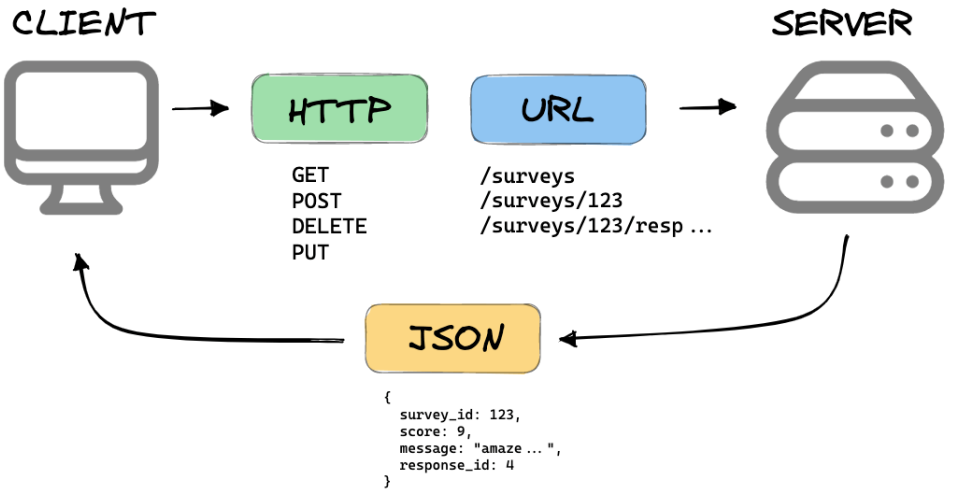
```json
{
    "endereco": {
        "cep": "31270901",
        "city": "Belo Horizonte",
        "neighborhood": "Pampulha",
        "service": "correios",
        "state": "MG",
        "street": "Av. Presidente Antônio Carlos, 6627"
    }
}
```

- **3. JSON (JavaScript Object Notation):** Es un formato de intercambio de datos que es fácil de leer tanto para humanos como para máquinas. Se utiliza para representar estructuras de datos simples y arrays asociativos (llamados objetos). En IoT, JSON se utiliza a menudo para enviar datos desde dispositivos a servidores.



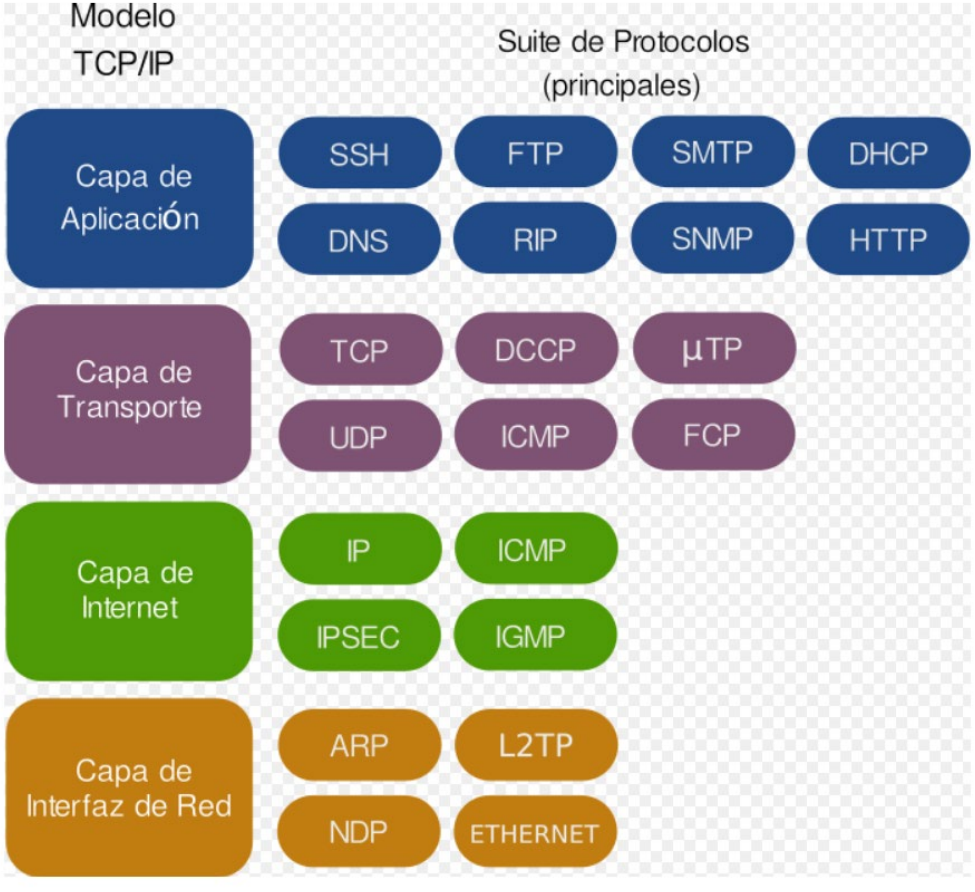
- **4. REST APIs (Representational State Transfer Application Programming Interfaces):** Las APIs REST son una forma de crear interacciones entre sistemas a través de HTTP. Las APIs REST se basan en los métodos estándar de HTTP como GET, POST, PUT y DELETE para realizar operaciones. En un contexto de IoT, las APIs REST se utilizan a menudo para permitir la comunicación entre los dispositivos IoT y los servidores de back-end.

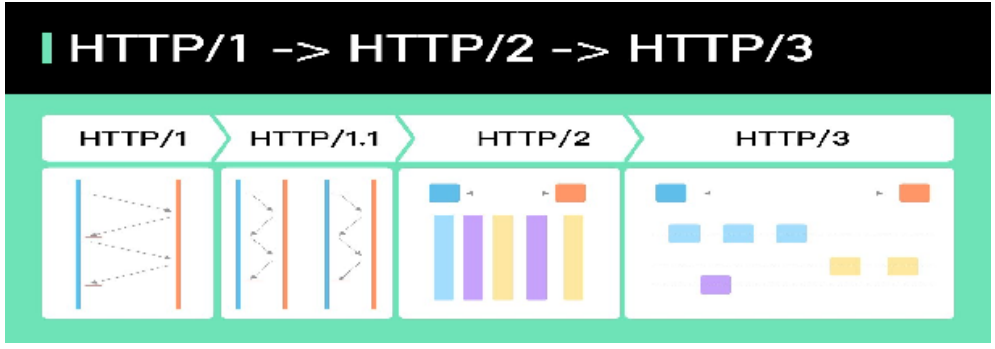




Cada uno de estos protocolos y formatos de datos tiene sus propias ventajas y puede ser la mejor elección dependiendo de las circunstancias específicas. Por ejemplo, si se necesita una latencia mínima y un consumo eficiente de energía, MQTT podría ser la mejor elección. Si se está integrando con sistemas web existentes, HTTP y las APIs REST podrían ser más adecuados. JSON es una excelente opción para la representación de datos debido a su simplicidad y legibilidad.

Historia y Actualidad del HTTP

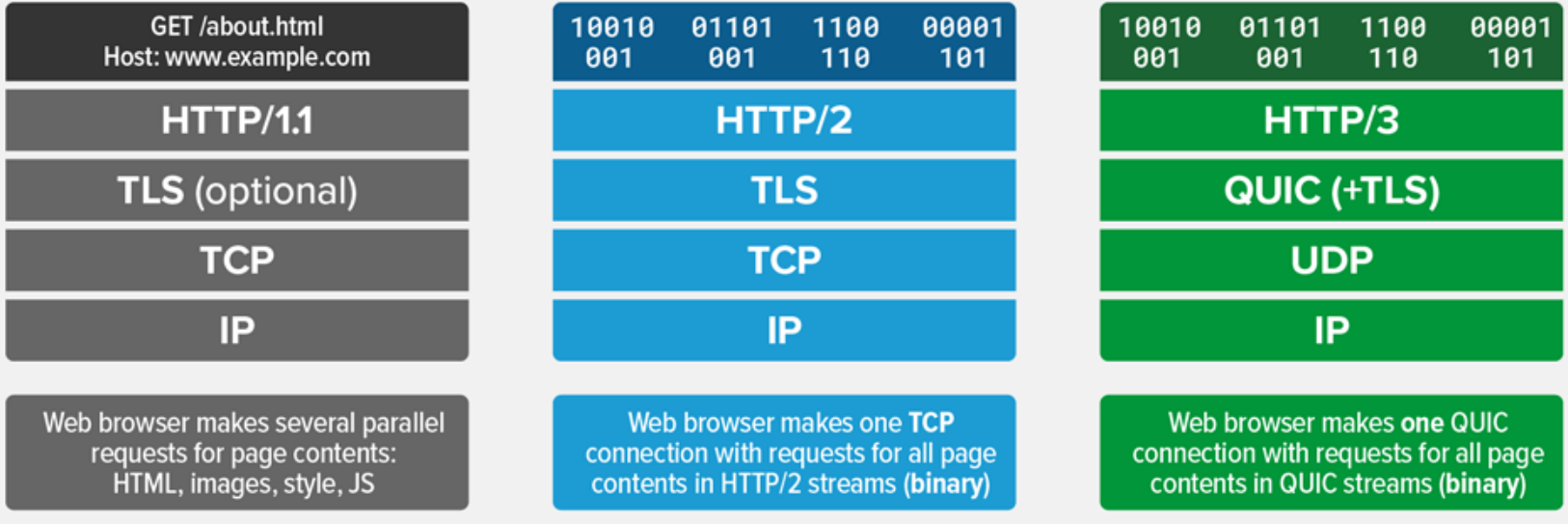




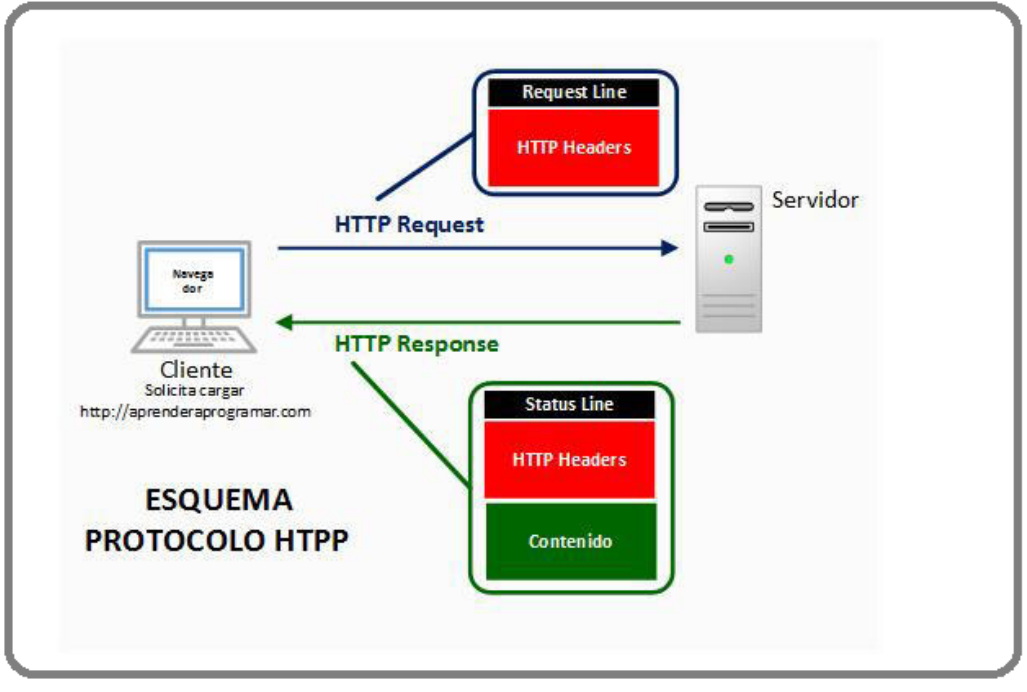
El Protocolo de Transferencia de Hipertexto, o HTTP, es la base de cualquier intercambio de datos en la web y se ha desarrollado a lo largo de las décadas para adaptarse a las crecientes demandas de velocidad y eficiencia. Comenzando con su invención en el CERN en 1989, HTTP ha evolucionado desde su versión 0.9, que solo podía manejar texto, hasta la versión actual, HTTP/2, que puede manejar múltiples solicitudes a la vez, e incluso hasta una futura versión, HTTP/3, que promete mayores velocidades y eficiencia en las redes móviles y de alta latencia. Hoy en día, HTTP es un componente crítico de la web y es usado en prácticamente todas las interacciones en línea, desde cargar páginas web hasta transferir datos a través de APIs.

- **Historia de HTTP:**Desarrollado por Tim Berners-Lee en el CERN en 1989.
- HTTP 0.9 (1991) - Permitía la transferencia de archivos de texto HTML a través de Internet.
- HTTP 1.0 (1996) - Introdujo la noción de encabezados HTTP y soporte para una gama más amplia de métodos de solicitud (GET, POST, HEAD).
- HTTP 1.1 (1997) - Mejoró la eficiencia al permitir conexiones persistentes, lo que significa que una única conexión TCP podría ser utilizada para varias solicitudes/resoluciones HTTP. También introdujo nuevas características como opciones de caché y codificación de contenido.
- HTTP/2 (2015) - Proporcionó mejoras significativas en eficiencia y velocidad, incluyendo solicitudes múltiples en una sola conexión TCP (multiplexación), compresión de encabezados y priorización de solicitudes.

- **Actualidad de HTTP:**HTTP/3 (en progreso) - Propone cambiar de TCP a QUIC, un protocolo de transporte basado en UDP, para mejorar la eficiencia y la velocidad, especialmente en redes móviles y de alta latencia.
- Ampliamente utilizado en todas las interacciones basadas en la web, desde la navegación por páginas hasta las comunicaciones de la API.



Elementos de la Comunicación HTTP



- HTTP opera en un ciclo de petición-respuesta, con el cliente enviando una petición a un servidor, y el servidor enviando una respuesta de vuelta al cliente.

Elementos de la Comunicación HTTP

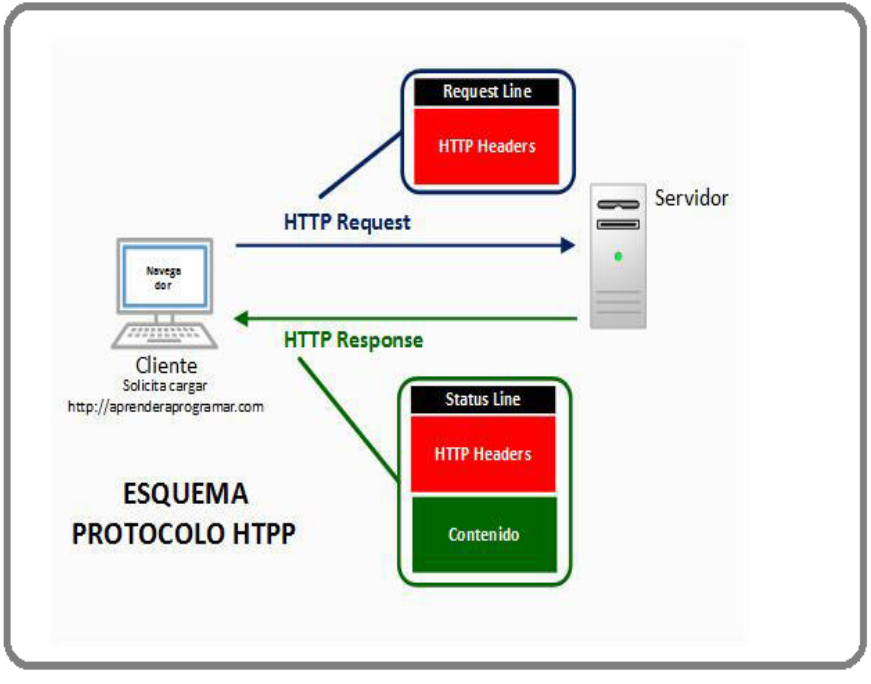
Las peticiones y las respuestas constan de varios componentes:

1. Los **métodos** HTTP definen qué tipo de operación se está solicitando.

2. Los **códigos de estado** HTTP informan sobre el resultado de la petición.

3. Los **encabezados** HTTP proporcionan información adicional sobre la petición o la respuesta, como el tipo de contenido, la codificación del contenido, la fecha, etc.

4. Finalmente, **el cuerpo** de la petición o la respuesta contiene los datos que se están enviando o recibiendo, como la información de un formulario en una petición o una página web en una respuesta.



- **Ciclo de Petición-Respuesta:**
  HTTP opera en un ciclo de petición-respuesta entre el cliente y el servidor.
- **Métodos HTTP:**
  Los métodos HTTP definen qué tipo de operación se está solicitando. Los más comunes son GET, POST, PUT, DELETE, entre otros.
- **Códigos de Estado HTTP:**
  Los códigos de estado HTTP informan el resultado de la petición. Los más conocidos son 200 (OK), 404 (Not Found), 500 (Internal Server Error), entre otros.
- **Encabezados HTTP:**
  Los encabezados HTTP proporcionan información adicional sobre la petición o respuesta.
- **Cuerpo de la Petición/Respuesta:**
  Contiene los datos que se están enviando o recibiendo. En una petición, esto podría ser información de formulario; en una respuesta, podría ser una página web, una imagen, etc.

El desarrollo Web

La capacidad de crear y mantener sitios web y aplicaciones web se ha convertido en una habilidad muy valorada. Sin embargo, el desarrollo web no es solo una cuestión de aprender un lenguaje de programación. Involucra una variedad de tecnologías y conceptos que se debe comprender para construir un sitio web o una aplicación web completamente funcional.
Aquí están los principales temas que se necesita aprender básicamente.

1. **Fundamentos de Internet y la Web**: Comprender cómo funciona Internet a nivel básico, qué es la World Wide Web y cómo se estructuran y se entregan las páginas web.
2. **HTTP y HTTPS**: Entender cómo funcionan estos protocolos de transferencia de hipertexto. Esto incluye comprender las solicitudes y respuestas, los métodos HTTP (GET, POST, PUT, DELETE, etc.) y los códigos de estado HTTP.

3. **HTML, CSS y JavaScript**: Estos son los tres pilares fundamentales del desarrollo web en el frontend. Necesitarás aprender cómo usar HTML para estructurar el contenido, CSS para estilizarlo, y JavaScript para agregar interactividad y hacer solicitudes HTTP al servidor.
4. **Frontend Frameworks y Bibliotecas**: Aprender cómo usar al menos uno de los principales frameworks o bibliotecas frontend de JavaScript, como React, Angular o Vue.js. Estas herramientas te permiten construir interfaces de usuario más complejas e interactivas.
5. **Python**: Necesitarás una sólida comprensión de Python para trabajar en el backend. Esto incluirá conceptos generales de programación, así como características específicas de Python.
6. **Flask o Django**: Aprender a usar uno de estos frameworks de desarrollo web en Python te permitirá construir el servidor que manejará las solicitudes HTTP.

7. **Bases de datos**: Entender cómo trabajar con bases de datos es crucial para la mayoría de las aplicaciones web. Podrías empezar con SQLite (que es simple y se integra bien con Python), y luego pasar a otras bases de datos como PostgreSQL, MySQL o MongoDB.
8. **API REST y JSON**: Aprender a construir y usar APIs REST te permitirá estructurar la comunicación entre el frontend y el backend de tu aplicación. JSON es el formato de datos más comúnmente usado para este propósito.
9. **Autenticación y seguridad**: Es importante aprender cómo manejar la autenticación de los usuarios y cómo asegurar tu aplicación contra amenazas comunes.
10. **Despliegue y hosting**: Finalmente, necesitarás aprender cómo desplegar tu aplicación en la web. Esto podría implicar el uso de servicios de hosting como Heroku, AWS, Google Cloud, etc.

Cada uno de estos temas es bastante amplio por sí mismo. Se puede empezar con los fundamentos, y luego ir profundizando en cada área a medida que sea necesario para el proyecto.

Bootstrap es un marco de trabajo muy importante para el diseño frontend, y proporciona una amplia gama de componentes de estilo predefinidos que pueden facilitar la creación de interfaces de usuario elegantes y coherentes.
Respecto a los **stacks** de tecnología, estos son conjuntos combinados de lenguajes de programación, frameworks y herramientas que se utilizan juntas para desarrollar una aplicación web. Algunos stacks populares incluyen:

1. **MEAN/MERN Stack**: MongoDB, Express.js, Angular/React, Node.js.
2. **LAMP Stack**: Linux, Apache, MySQL, PHP.
3. **Django Stack**: Django, Python, MySQL (con variantes incluyendo diferentes bases de datos y servidores web).
4. **Flask Stack**: Flask, Python, SQLite (con variantes similares a Django).
5. **JAMStack**: JavaScript, APIs, Markup (para desarrollo frontend y sitios web estáticos).

Cada uno de estos stacks tiene sus propias ventajas, desventajas y casos de uso ideales, y es útil entender qué stack (o stacks) se ajusta mejor a las necesidades de tu proyecto.

Practicas con HTTP

La especificación HTTP define varios otros métodos que podrías encontrar en ciertas circunstancias. Aquí está la lista completa de métodos de solicitud HTTP según la especificación HTTP/1.1 y HTTP/2:

GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE, CONNECT, PATCH

Es importante tener en cuenta que no todos los servidores soportarán todos estos métodos. Algunos pueden limitarse a los métodos más comunes (GET, POST, PUT, DELETE, HEAD), mientras que otros pueden soportar todos los métodos enumerados aquí. Además, HTTP/3, la próxima versión del protocolo, puede añadir más métodos a la lista.

Las solicitudes HTTP son una parte fundamental de cualquier comunicación en la web. Cuando hablamos de "solicitudes" en este contexto, nos referimos a los mensajes que un cliente (como un navegador web o un script de Python) envía a un servidor para solicitar una acción específica.

- Como vimos hay varios tipos de solicitudes HTTP, cada una de las cuales se utiliza para una acción diferente. Aquí están las más comunes:

1. **GET**: Este es el tipo de solicitud más común. Se utiliza para solicitar un recurso del servidor. Cuando escribes una URL en tu navegador y presionas enter, tu navegador envía una solicitud GET al servidor asociado con esa URL para pedirle la página web.
2. **POST**: Este tipo de solicitud se utiliza para enviar datos al servidor. Por ejemplo, cuando envías un formulario en una página web, los datos del formulario generalmente se envían al servidor como parte de una solicitud POST.
3. **PUT**: Este tipo de solicitud se utiliza para actualizar un recurso existente en el servidor. Por ejemplo, podrías usar una solicitud PUT para actualizar los detalles de un usuario en una base de datos.
4. **DELETE**: Como probablemente puedas adivinar, este tipo de solicitud se utiliza para eliminar un recurso del servidor.

5. **HEAD**: Este tipo de solicitud es similar a una solicitud GET, pero el servidor sólo devuelve los encabezados de la respuesta, no el cuerpo. Esto puede ser útil si sólo quieres comprobar si un recurso existe o no, sin descargar el recurso en sí.
6. **OPTIONS**: Este tipo de solicitud se utiliza para descubrir qué tipos de solicitudes HTTP son permitidas por el servidor para un recurso específico.
7. **PATCH**: Este tipo de solicitud se utiliza para hacer actualizaciones parciales a un recurso en el servidor.

- Cada tipo de solicitud se utiliza para un propósito diferente, y es importante entender qué tipo de solicitud se debe usar para cada tarea cuando trabajas con HTTP.

- Para las prácticas de solicitudes HTTP, usaremos el módulo requests en Python. Este módulo nos permite enviar solicitudes HTTP de manera muy sencilla.
- **Requests** es una biblioteca de Python que simplifica la tarea de hacer solicitudes HTTP. Con una API sencilla y fácil de usar, requests permite enviar todo tipo de solicitudes HTTP sin la necesidad de realizar tareas de bajo nivel, como la manipulación de parámetros GET/POST o la gestión de cookies.

Si no tienes la biblioteca requests en tu entorno de Python, puedes instalarla usando pip:

```
$ pip install requests
```

Una vez instalado, puedes importar el módulo requests en tu script de Python:

```
$ import requests
```

Practica: Aplicación web con flask

Veamos un ejemplo sencillo de una aplicación web con Flask (un micro framework de Python) que permitirá realizar prácticas básicas con HTTP.
Flask es ligero y fácil de aprender, por lo que es una excelente opción para empezar. La aplicación tendrá una ruta que devuelve una respuesta HTTP simple, lo que permitirá practicar las solicitudes HTTP desde el cliente.

Backend con Flask (app.py):
Primero, vamos a crear una aplicación Flask con una sola ruta. Para este ejemplo, necesitarás tener instalado Flask.

Si aún no lo tienes, puedes instalarlo con: **$ pip install flask**

app.py

```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/')
def home():
    return jsonify(message='¡Hola Mundo!')

if __name__ == '__main__':
    app.run(debug=True)
```

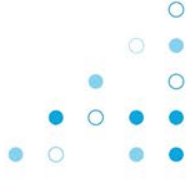
Este script de Python crea una nueva aplicación Flask y define una sola ruta ("/") que devuelve un mensaje de '¡Hola Mundo!' en formato JSON.

Una vez que hayas guardado este script en un archivo llamado app.py, puedes ejecutar tu servidor Flask con el comando python app.py. Después de ejecutar este comando, tu servidor debería estar corriendo en localhost en el puerto 5000.

Frontend con HTML y JavaScript (index.html):

Ahora, vamos a crear una página web muy simple que hará una solicitud GET a nuestro servidor Flask cuando se cargue la página, y luego mostrará la respuesta en la página.

Crear un archivo index.html con el siguiente contenido:



```
<!DOCTYPE html>
<html>
<head>
    <title>Prueba de HTTP con Python y Flask</title>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js
"></script>
</head>
<body>
    <h1>Respuesta del servidor:</h1>
    <p id="message"></p>

    <script>
        $(document).ready(function() {
            $.get('http://localhost:5000', function(data) {
                $('#message').text(data.message);
            });
        });
    </script>
</body>
</html>
```

Este HTML incluye un poco de JavaScript (utilizando jQuery, una popular biblioteca de JavaScript) para hacer la solicitud GET a nuestro servidor Flask cuando se carga la página. La respuesta del servidor se inserta en el párrafo con el id "message".

Ahora, si abres este archivo HTML en un navegador web mientras tu servidor Flask está corriendo, deberías ver el mensaje '¡Hola Mundo!' que viene del servidor.

Este es un ejemplo muy simple, pero debería darte una buena idea de cómo puedes empezar a trabajar con solicitudes y respuestas HTTP en el contexto de una aplicación web. A partir de aquí, puedes empezar a experimentar con rutas más complejas, solicitudes POST, y más.

¡Muchas gracias!