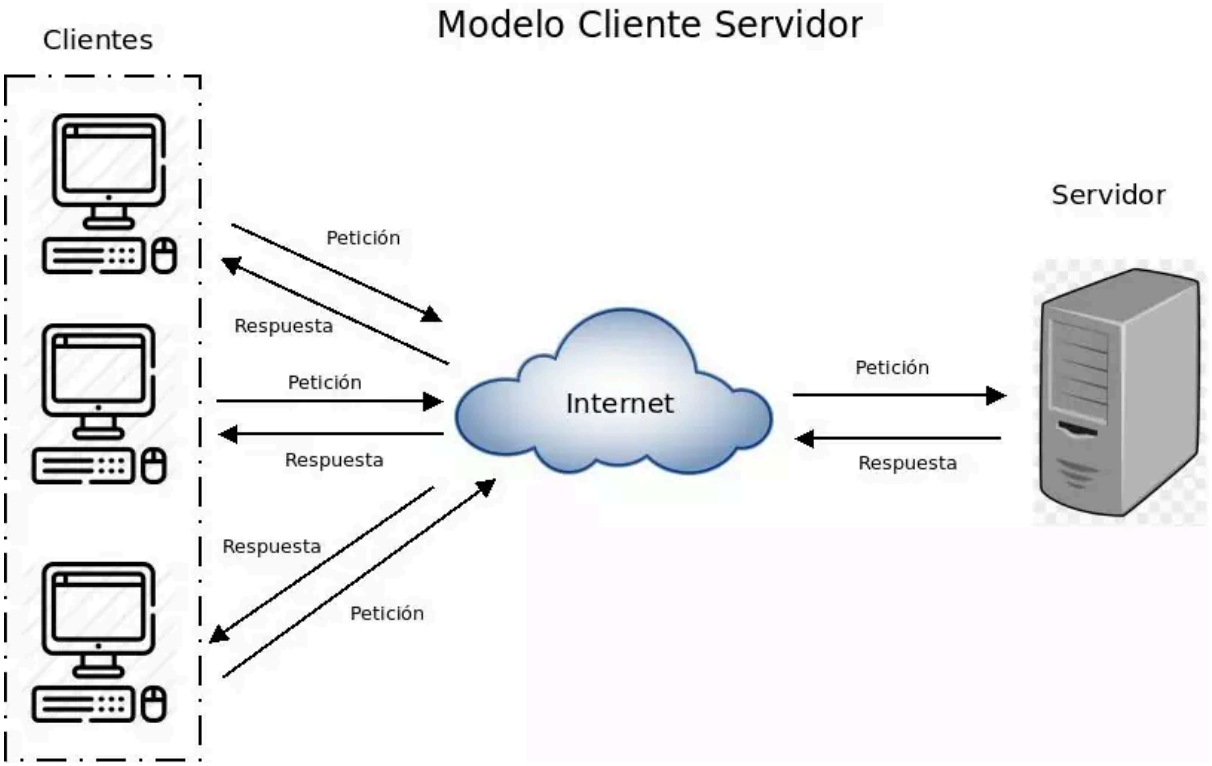
Unidad 4

Servicios Web

Arquitectura Cliente-Servidor para Aplicaciones Web

La arquitectura cliente-servidor es un modelo de diseño utilizado en el desarrollo de aplicaciones web, donde las tareas y responsabilidades están divididas entre dos entidades principales: el cliente y el servidor. Este modelo es fundamental para la mayoría de las aplicaciones web modernas y permite una interacción eficiente y estructurada entre los usuarios y los servicios web.

Componentes de la Arquitectura Cliente-Servidor



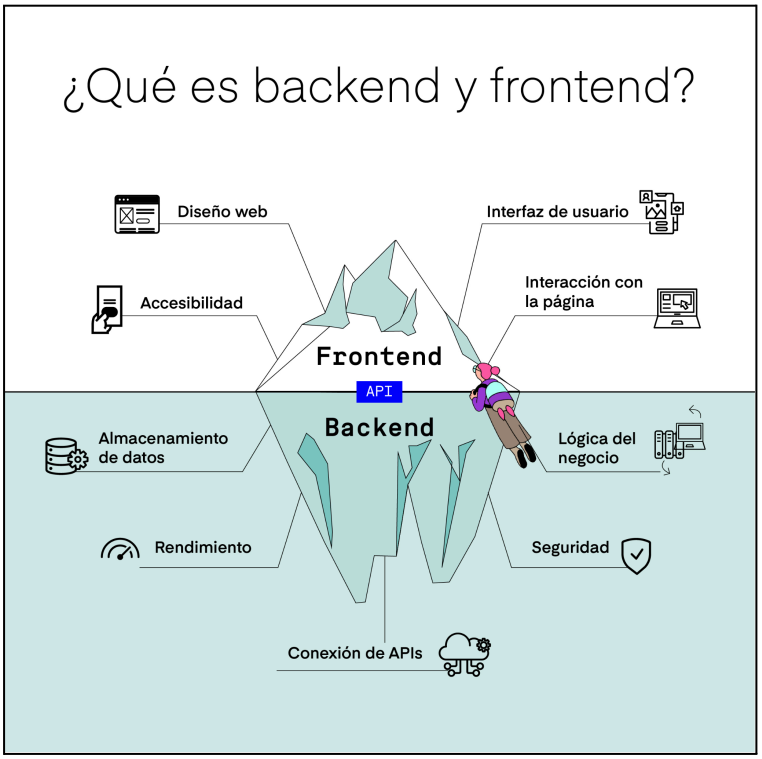
Cliente: El cliente es el lado del usuario, típicamente un navegador web o una aplicación que solicita recursos o servicios desde el servidor. Los clientes pueden ser computadoras, teléfonos inteligentes, tabletas, o cualquier dispositivo capaz de conectarse a la web.

- **Interfaz de Usuario (UI)**: La parte de la aplicación que interactúa directamente con el usuario, como formularios, botones, y gráficos.
- **Lógica de Presentación**: Gestiona cómo se muestra la información al usuario y cómo se manejan las interacciones del usuario.

Servidor: El servidor es el lado que proporciona recursos o servicios solicitados por el cliente. Puede ser un servidor web, un servidor de aplicaciones, o una base de datos.

- **Servidor Web**: Gestiona las solicitudes HTTP/HTTPS y responde con el contenido adecuado, como páginas HTML, archivos, o datos JSON.
- **Servidor de Aplicaciones**: Procesa la lógica de la aplicación, como la autenticación de usuarios, la gestión de sesiones y la ejecución de operaciones comerciales.
- **Base de Datos**: Almacena y gestiona los datos utilizados por la aplicación, respondiendo a consultas y actualizaciones de datos.

Frontend y Backend en el Desarrollo de Aplicaciones Web



En el desarrollo de aplicaciones web, se distingue claramente entre dos áreas principales: el frontend y el backend. Cada una de estas áreas se encarga de aspectos específicos del desarrollo y funcionamiento de una aplicación web, trabajando conjuntamente para proporcionar una experiencia de usuario fluida y eficiente.

Frontend

El frontend es la parte de la aplicación web con la que los usuarios interactúan directamente. Está compuesto por la interfaz de usuario y la experiencia de usuario, y su desarrollo implica principalmente las siguientes tecnologías:

- **HTML (HyperText Markup Language)**: Es el lenguaje estándar para crear y estructurar el contenido en la web. Define la estructura básica de las páginas web utilizando elementos como encabezados, párrafos, listas, enlaces, imágenes, y más. (Unidad 2)

- **CSS (Cascading Style Sheets)**: Es el lenguaje utilizado para describir la presentación de un documento HTML. Permite estilizar la página web, definiendo

aspectos como colores, fuentes, márgenes, disposición de elementos y adaptabilidad a diferentes dispositivos (responsive design). (Unidad 3)

- **JavaScript**: Es un lenguaje de programación que permite implementar funcionalidades dinámicas en una página web. Con JavaScript, los desarrolladores pueden crear interacciones complejas, manipular el DOM (Document Object Model), validar formularios, manejar eventos, y comunicar con el backend mediante AJAX. (Unidad 4)

Frameworks y Bibliotecas: Para facilitar y optimizar el desarrollo frontend, se utilizan frameworks y bibliotecas como React, Angular, Vue.js, Bootstrap, y otros. Estos proporcionan herramientas y componentes preconstruidos que agilizan el proceso de desarrollo y mejoran la experiencia del usuario.

Ejemplo de Tareas del Frontend:

- Crear formularios interactivos.
- Estilizar la página para hacerla visualmente atractiva.
- Implementar la navegación entre diferentes secciones de la aplicación.
- Validar los datos ingresados por los usuarios.
- Realizar peticiones asíncronas al backend y actualizar la interfaz en consecuencia.

Backend
El backend es la parte de la aplicación web que se ejecuta en el servidor y es responsable de gestionar la lógica de negocio, la interacción con bases de datos, la autenticación de usuarios y otras tareas críticas. Las tecnologías comúnmente utilizadas en el desarrollo backend incluyen:

- **Lenguajes de Programación**: Los lenguajes más utilizados en el desarrollo backend son JavaScript (con Node.js), Python, Ruby, PHP, Java, y C#. Cada lenguaje tiene sus propias características y ventajas, y la elección del lenguaje depende de los requisitos específicos del proyecto.

- **Frameworks**: Para simplificar el desarrollo backend, se utilizan frameworks como Express.js (para Node.js), Django (para Python), Ruby on Rails (para Ruby), Laravel (para PHP), Spring (para Java), y ASP.NET (para C#). Estos frameworks proporcionan estructuras y herramientas predefinidas para gestionar rutas, manejar solicitudes HTTP, y acceder a bases de datos.

- **Bases de Datos**: Las aplicaciones web necesitan almacenar y recuperar datos, para lo cual se utilizan bases de datos como MySQL, PostgreSQL, MongoDB, y SQLite. Los desarrolladores backend son responsables de diseñar esquemas de bases de datos, escribir consultas SQL o trabajar con ORM (Object-Relational Mapping) para interactuar con las bases de datos.

- **APIs (Application Programming Interfaces)**: El backend expone servicios a través de APIs que el frontend puede consumir. Las APIs pueden ser **RESTful** o basadas

en GraphQL, y permiten que los diferentes componentes de la aplicación se comuniquen entre sí.

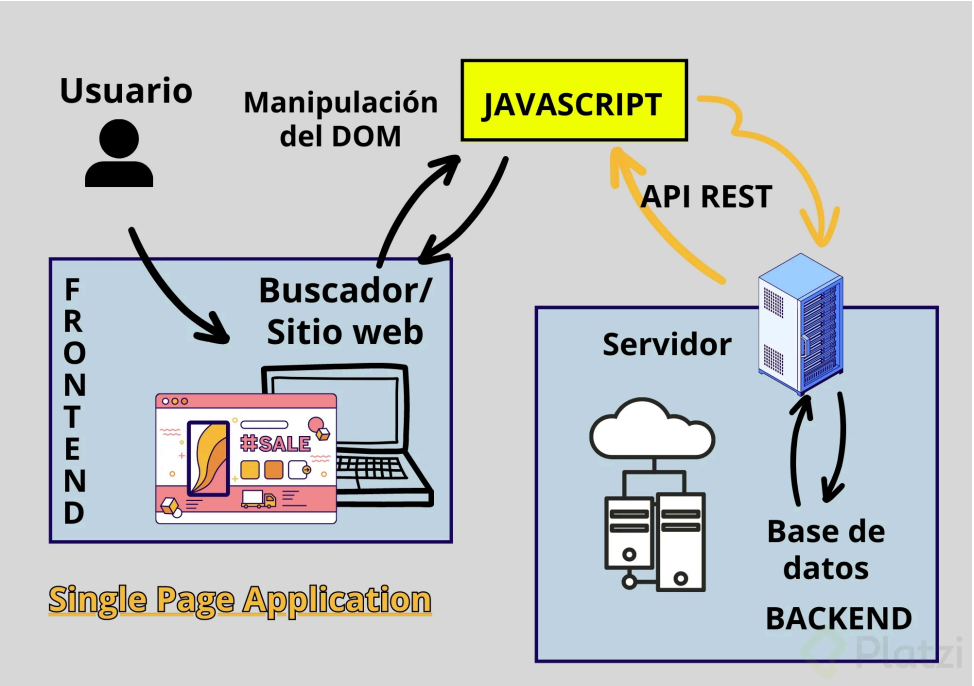
Ejemplo de Tareas del Backend:

- Gestionar la autenticación y autorización de usuarios.
- Procesar y validar datos enviados desde el frontend.
- Interactuar con la base de datos para almacenar y recuperar información.
- Implementar la lógica de negocio de la aplicación.
- Manejar la comunicación entre diferentes servicios y componentes de la aplicación.

Interacción entre Frontend y Backend
El frontend y el backend se comunican a través de peticiones **HTTP**. El frontend envía solicitudes al backend para obtener datos o ejecutar acciones, y el backend responde con la información necesaria. Esta comunicación es esencial para el funcionamiento de una aplicación web, y se realiza generalmente mediante APIs.

Ejemplo de Flujo de Trabajo:

- Un usuario ingresa datos en un formulario en la interfaz (frontend).
- El frontend valida los datos y los envía al servidor a través de una solicitud HTTP (POST).
- El backend recibe la solicitud, procesa los datos, realiza operaciones necesarias (como almacenar datos en la base de datos) y responde con un resultado.
- El frontend recibe la respuesta del backend y actualiza la interfaz de usuario en consecuencia.



Introducción a los Servicios Web y API RESTful

En la era digital, la comunicación entre diferentes aplicaciones y servicios es fundamental para crear sistemas integrados y funcionales. Los servicios web y las API RESTful son herramientas clave que facilitan esta comunicación, permitiendo que las aplicaciones intercambien datos y funciones de manera eficiente.

¿Qué son los Servicios Web?

Los servicios web son aplicaciones que utilizan protocolos estándar como **HTTP/HTTPS** para intercambiar datos entre sistemas diferentes a través de una red, generalmente Internet. Estos servicios permiten que las aplicaciones interactúen entre sí, independientemente de las plataformas o lenguajes de programación en los que estén desarrolladas.

Tipos de Servicios Web:

- **SOAP (Simple Object Access Protocol):** Es un protocolo basado en XML que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos **XML**. SOAP es conocido por su robustez y extensibilidad, pero también por ser más complejo y con mayor sobrecarga que otros métodos.

- **REST (Representational State Transfer)**: Es un estilo arquitectónico que utiliza HTTP para realizar operaciones CRUD (Create, Read, Update, Delete) en recursos web representados en formatos como **JSON** o XML. *REST es más simple y ligero comparado con SOAP, lo que lo hace popular para el desarrollo de servicios web*.

¿Qué es una API?

Una API (Application Programming Interface) es un conjunto de reglas y definiciones que permiten que las aplicaciones se comuniquen entre sí. Las APIs actúan como intermediarios que permiten que diferentes software interactúen, proporcionándoles métodos y datos para realizar diversas tareas.

Tipos de APIs:

- APIs Públicas: Están disponibles para cualquier desarrollador y son utilizadas para integrar servicios externos en una aplicación. Ejemplos incluyen las APIs de Google Maps, Twitter, y Facebook.

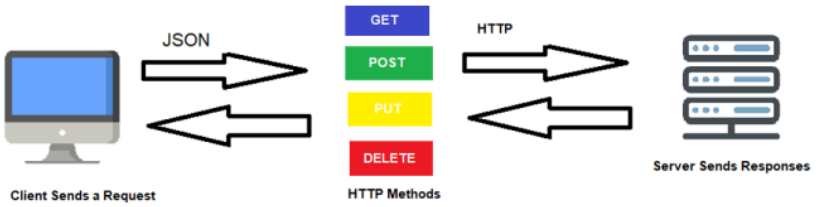
- APIs Privadas: Se utilizan dentro de una organización para mejorar la interoperabilidad entre sistemas internos.

- APIs Asociadas: Se utilizan para compartir datos y servicios entre socios comerciales o desarrolladores externos específicos.

¿Qué es una API RESTful?

Una API RESTful es una implementación de una API que sigue los principios de REST. REST (Representational State Transfer) es un estilo arquitectónico para diseñar servicios de red escalables. Las APIs RESTful son ampliamente utilizadas debido a su simplicidad y eficiencia.

Ejemplo de Operaciones CRUD en una API RESTful:



GET /usuarios: Recupera una lista de usuarios.
GET /usuarios/1: Recupera los detalles del usuario con ID 1.
POST /usuarios: Crea un nuevo usuario.
PUT /usuarios/1: Actualiza los detalles del usuario con ID 1.
DELETE /usuarios/1: Elimina el usuario con ID 1.

Beneficios de las APIs RESTful:

- **Simplicidad**: Utilizan los verbos HTTP estándar, lo que las hace fáciles de entender y utilizar.
- **Escalabilidad**: Diseñadas para trabajar en una arquitectura distribuida y escalar horizontalmente.
- **Flexibilidad**: Pueden ser utilizadas por diferentes lenguajes de programación y plataformas.
- **Ligereza**: Generalmente utilizan JSON, un formato ligero para el intercambio de datos.

Ejemplo de una APIs RESTful:

En este ejemplo simple se configura un servidor HTTP básico con **Node.js**. Cuando recibe una solicitud **GET** a la ruta **/users**, responde con una lista de usuarios en formato JSON. Si la ruta no coincide, responde con un mensaje de error 404.

```
const http = require('http');

let users = [
  { id: 1, name: 'Alice' },
  { id: 2, name: 'Bob' },
  { id: 3, name: 'Charlie' }
];

const server = http.createServer((req, res) => {
  const { method, url } = req;

  if (method === 'GET' && url === '/users') {
    // Configurar el encabezado de la respuesta HTTP
    res.writeHead(200, { 'Content-Type': 'application/json' });
    // Enviar la lista de usuarios en formato JSON
    res.end(JSON.stringify(users));
  } else {
    // Si la ruta no coincide, enviar un mensaje de error 404
    res.writeHead(404, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify({ message: 'Not Found' }));
  }
});

const PORT = 3000;
server.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Probar la API

Puedes probar la API utilizando herramientas como curl o Postman. Para llamar al API usamos un ejemplo como el siguiente. La respuesta será un JSON, que veremos más adelante.

```
curl -X GET http://localhost:3000/users
```

Formatos de Intercambio de Datos: JSON

¿Qué es JSON?

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y fácil de leer y escribir tanto para humanos como para las máquinas. JSON se utiliza ampliamente para transmitir datos en aplicaciones web, especialmente entre un servidor y un cliente web. Su sintaxis se deriva del lenguaje de programación JavaScript, aunque es independiente de cualquier lenguaje de programación específico.

Estructura de JSON

JSON está compuesto por dos estructuras principales:

Objetos: Los objetos en JSON se representan como una colección de pares clave-valor delimitados por llaves {}. Cada clave es una cadena y está seguida por un valor, separados por dos puntos :. Los pares clave-valor están separados por comas ,.

```
{
  "nombre": "Juan",
  "edad": 30,
  "ciudad": "Madrid"
}
```

Arreglos: Los arreglos en JSON son una colección de valores ordenados delimitados por corchetes []. Los valores pueden ser de cualquier tipo de dato soportado por JSON (números, cadenas, booleanos, objetos, arreglos, nulos).

```
[
  "Manzana",
  "Banana",
  "Cereza"
]
```

Tipos de Datos en JSON

JSON admite los siguientes tipos de datos:

Cadenas (Strings): Se representan con comillas dobles "

"nombre": "Juan"

Números (Numbers): Pueden ser enteros o decimales.

"edad": 30

Booleanos (Booleans): Pueden ser true o false

"es_estudiante": false

Nulos (Nulls): Representa un valor nulo.

"apellido": null

Objetos (Objects): Colección de pares clave-valor.

```
"direccion": {
  "calle": "Calle Mayor",
  "ciudad": "Madrid"
}
```

Arreglos (Arrays): Lista ordenada de valores

"hobbies": ["Leer", "Correr", "Programar"]

Desventajas de JSON

- Sin soporte para comentarios: JSON no admite comentarios, lo que puede dificultar la inclusión de anotaciones o explicaciones en los datos.
- Menos robusto que XML: JSON es menos expresivo que XML y puede no ser adecuado para ciertos tipos de datos complejos o jerárquicos.

Consumo de servicios web desde JavaScript.

El consumo de servicios web es una tarea esencial en el desarrollo moderno de aplicaciones web. JavaScript, con su capacidad de ejecutar tanto en el navegador como en el servidor, ofrece varias formas de interactuar con servicios web y APIs. A continuación, exploramos los métodos más comunes para consumir servicios web desde JavaScript.

Métodos para Consumir Servicios Web en JavaScript

- XMLHttpRequest (XHR)
- Fetch API
- Axios

XMLHttpRequest (XHR)
XMLHttpRequest es una API integrada en los navegadores web que permite realizar solicitudes HTTP. Es el método más antiguo y ampliamente compatible para consumir servicios web en JavaScript.

Ejemplo de uso de XMLHttpRequest

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://api.example.com/data', true);

xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
        var data = JSON.parse(xhr.responseText);
        console.log(data);
    }
};

xhr.send();
```

Fetch API
La Fetch API es una interfaz moderna que proporciona una forma más sencilla y flexible de realizar solicitudes HTTP. Es una alternativa más poderosa y legible que XMLHttpRequest.

Ejemplo de uso de Fetch API

```
fetch('https://api.example.com/data')
    .then(response => {
        if (!response.ok) {
            throw new Error('Network response was not ok ' +
response.statusText);
        }
        return response.json();
    })
    .then(data => {
        console.log(data);
    })
    .catch(error => {
        console.error('There has been a problem with your fetch
operation:', error);
    });
```

Axios
Axios es una biblioteca basada en promesas para realizar solicitudes HTTP. Es popular debido a su simplicidad y capacidad para manejar solicitudes y respuestas de manera más eficiente.

```
const axios = require('axios');

axios.get('https://api.example.com/data')
    .then(response => {
        console.log(response.data);
    })
    .catch(error => {
        console.error('There was an error!', error);
    });
```

Pasos Generales para Consumir Servicios Web

- **Hacer una solicitud HTTP**: Utilizando uno de los métodos mencionados (XHR, Fetch API, Axios), se envía una solicitud al servicio web.

- **Manejar la Respuesta**: Las respuestas del servicio web se manejan típicamente en formato JSON. Estas respuestas se procesan para usarlas en la aplicación.

- **Manejo de Errores**: Es importante manejar errores de red y respuestas incorrectas para asegurar la robustez de la aplicación.

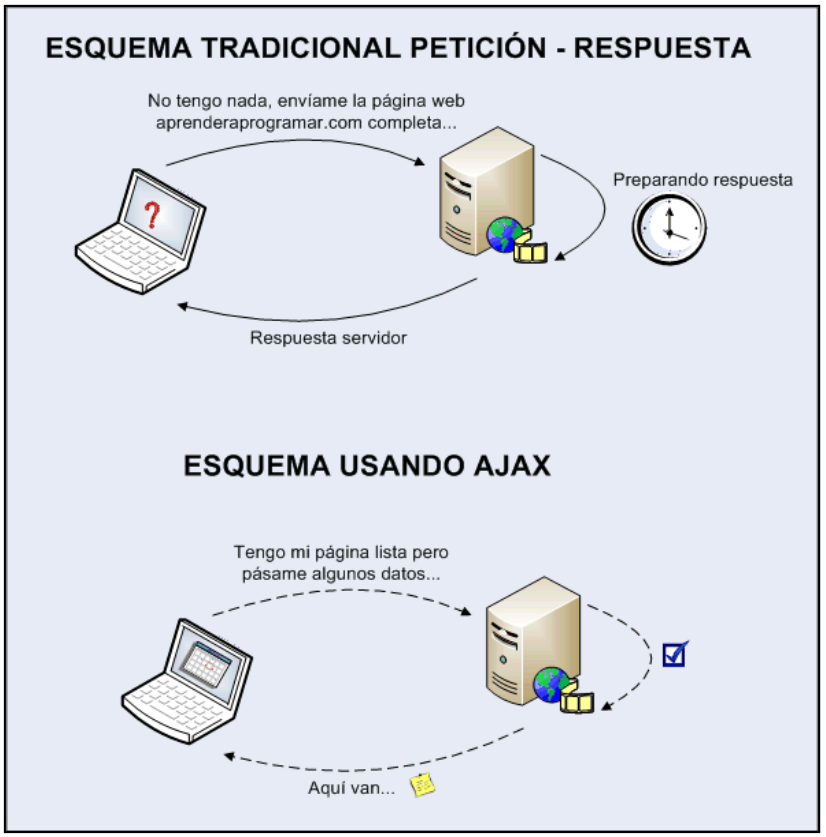
Introducción a AJAX

AJAX, que significa "Asynchronous JavaScript and XML" (JavaScript y XML asíncronos), es una técnica utilizada en el desarrollo web para crear aplicaciones web interactivas. Con AJAX, las aplicaciones web pueden enviar y recibir datos del servidor de manera asíncrona sin recargar la página completa. Esto proporciona una experiencia de usuario más fluida y rápida.

¿Cómo funciona AJAX?

El funcionamiento de AJAX se basa en la combinación de varias tecnologías:

- **HTML/CSS**: Para la estructura y estilo de la página web.
- **JavaScript**: Para manipular y controlar el contenido dinámico.
- **XMLHttpRequest**: Para realizar solicitudes HTTP asíncronas al servidor.
- **JSON/XML**: Formatos de datos utilizados para intercambiar información entre el servidor y el cliente.



Ejemplo de Uso de AJAX

A continuación, se muestra un ejemplo simple de cómo usar AJAX para consumir un servicio web:

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AJAX Example</title>
</head>
<body>
    <h1>AJAX Example</h1>
    <button onclick="loadData()">Load Data</button>
    <div id="result"></div>

    <script src="script.js"></script>
</body>
</html>
```

JavaScript (script.js)

```
function loadData() {
    var xhr = new XMLHttpRequest();
    var url = 'https://api.example.com/data';

    xhr.onreadystatechange = function() {
        if (xhr.readyState === 4 && xhr.status === 200) {
            var data = JSON.parse(xhr.responseText);
            document.getElementById('result').innerHTML = JSON.stringify(data, null, 2);
        }
    };

    xhr.open('GET', url, true);
    xhr.send();
}
```

Ventajas de Usar AJAX

- Interactividad: Permite actualizar partes de la página web sin recargar por completo.
- Velocidad: Mejora la velocidad de la aplicación web, ya que solo se actualizan los datos necesarios.
- Mejor Experiencia de Usuario: Al reducir los tiempos de carga y proporcionar respuestas rápidas, mejora significativamente la experiencia del usuario.