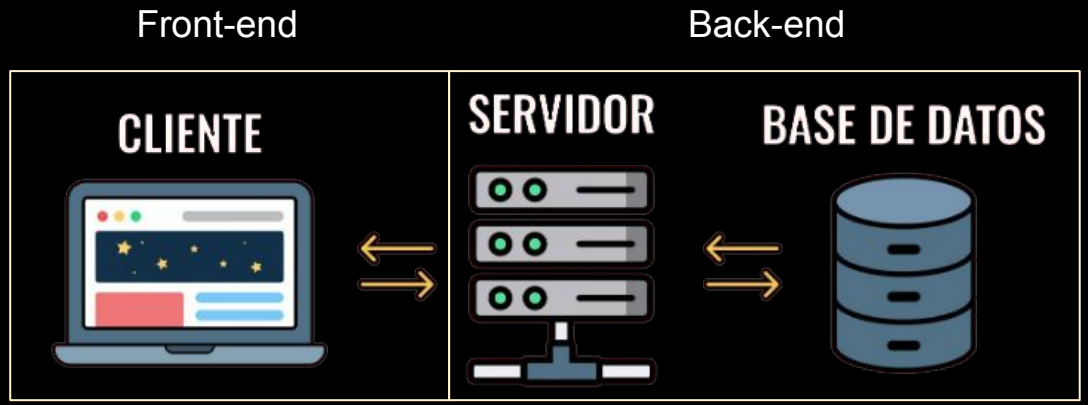
Clase Nº 2: Componentes de software

Crombie

Componentes de software

Son unidades autónomas y reutilizables que forman parte de un sistema de software más grande. Estos componentes se pueden combinar y ensamblar para construir aplicaciones más complejas

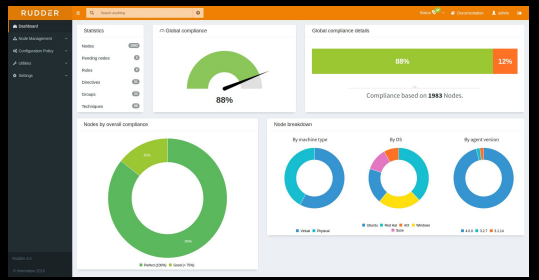


Crombie

crombie.dev

Cliente o Frontend

- Es la parte de la aplicación que interactúa con los **usuarios**.
- Todo aquello que vemos al ingresar a un sitio web: imágenes, colores, letras, etc.

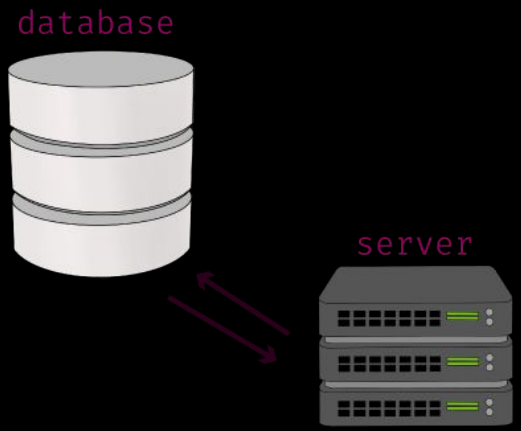


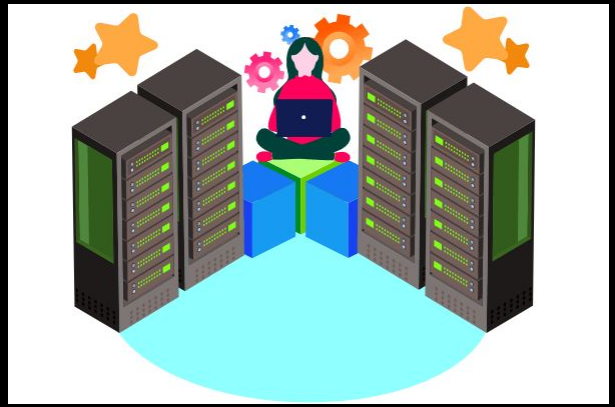
Crombie

crombie.dev

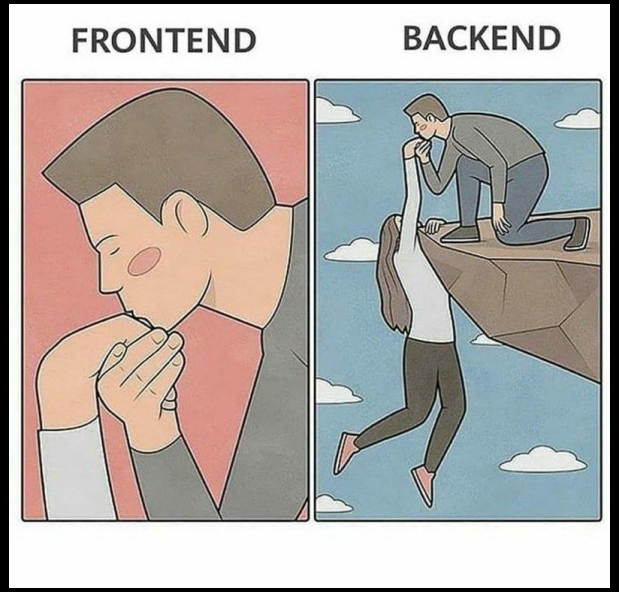
Servidor o Backend

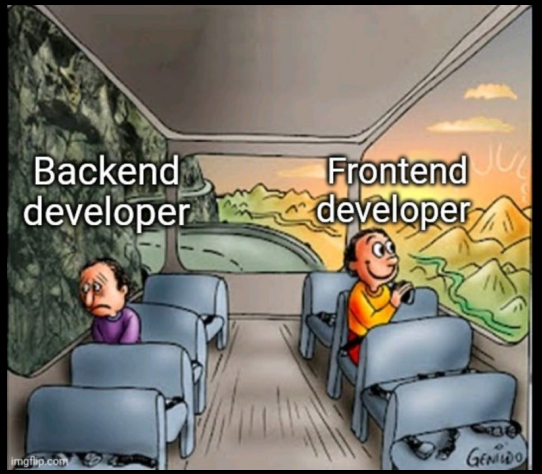
- Parte de un sistema de software que se encarga de procesar la lógica, el almacenamiento y la manipulación de los datos.
- Es responsable de gestionar la funcionalidad detrás de escena que permite que las aplicaciones funcionen correctamente.
- Lo que el cliente no ve
- Guarda toda la información de la aplicación





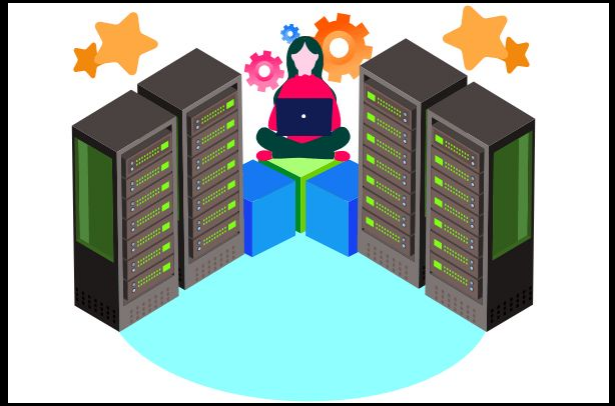
Otra forma de verlo





API REST

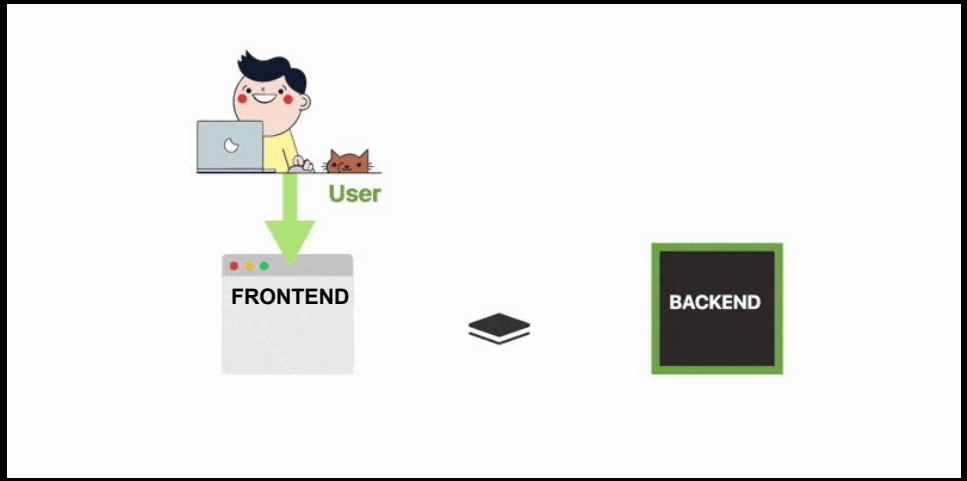
API REST (Application Programming Interface Representational State Transfer) es un estilo arquitectónico utilizado en el desarrollo de aplicaciones web, que permite la comunicación y transferencia de datos entre diferentes sistemas a través de protocolos HTTP



¿Cómo se comunican?

El frontend (cliente) envía solicitudes al backend (servidor) con información y parámetros necesarios, y el backend procesa la solicitud y envía una respuesta de vuelta al frontend, generalmente con los datos solicitados.

Esto permite que el frontend y el backend intercambien información y se mantengan sincronizados.

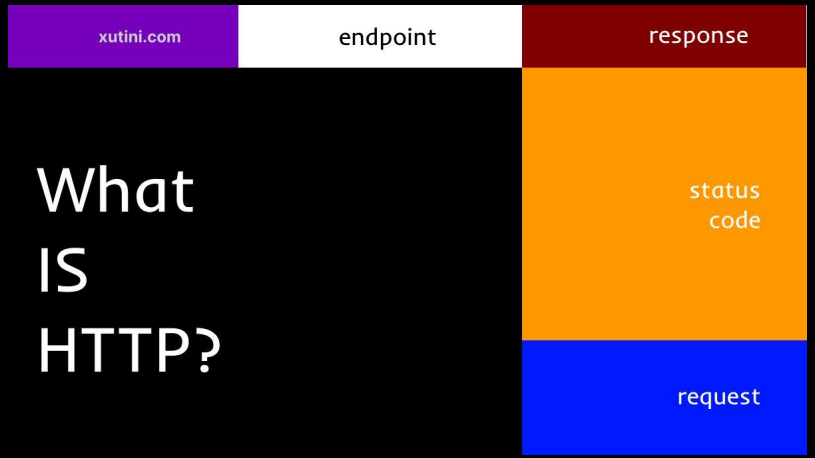


Protocolo de comunicación HTTP

Funciona como un conjunto de reglas que permite a los navegadores y servidores intercambiar información. El cliente (navegador) envía una solicitud HTTP al servidor para obtener datos, y el servidor responde con una respuesta HTTP que incluye el código de estado y los datos solicitados

Todas las solicitudes se componen de los siguientes datos:

- Host
- Endpoint
- Request
- Response
- Status Code



Otros protocolos existentes

- HTTP (Hypertext Transfer Protocol): Protocolo utilizado para la comunicación entre clientes y servidores web, permitiendo la transferencia de hipertexto y otros recursos.
- FTP (File Transfer Protocol): Protocolo utilizado para la transferencia de archivos entre un cliente y un servidor en una red, con funcionalidades básicas de listar, descargar, subir y eliminar archivos.
- SFTP (SSH File Transfer Protocol): Protocolo seguro basado en SSH que permite la transferencia segura de archivos y ejecución de comandos remotos en un servidor.
- SMTP (Simple Mail Transfer Protocol): Protocolo utilizado para el envío de correo electrónico entre servidores, especificando cómo deben entregarse los mensajes de correo electrónico a través de Internet.
- SSH (Secure Shell): Protocolo de red que proporciona una conexión segura y cifrada para administrar y acceder a servidores remotos, permitiendo la ejecución de comandos y transferencia de archivos de forma segura.

Probando HTTP

Realicemos una solicitud HTTP a una API (backend) pública y veamos cómo funciona.

Para este ejercicio descargaremos Postman: https://www.postman.com/

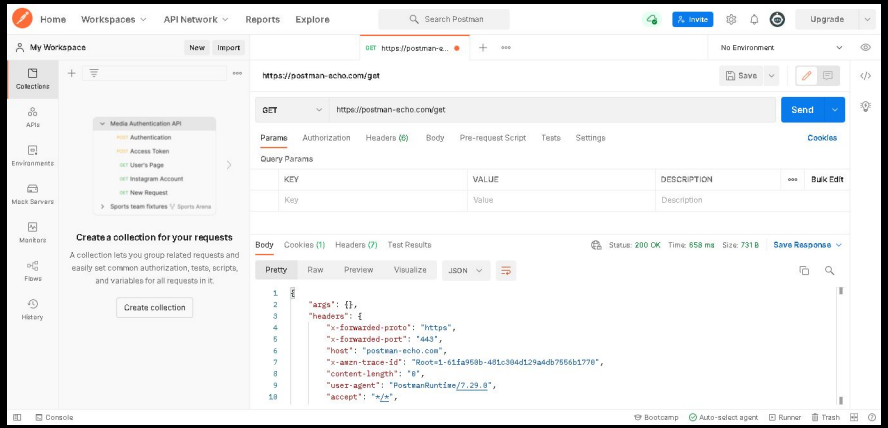
Host: https://jsonplaceholder.typicode.com

Endpoint: /comments

Request: ?postId=1

Response:

Status Code:
`



Continuando con React

Hooks y Fetch de datos

Objetivo de la clase

- ➔ Comprender las diferencias entre React vs HTML+CSS+JS
- ➔ Poner en practica los hooks de react: useState, useEffect y useContext para crear 3 aplicaciones

Crombie

Comando a correr

```
Usuario@LENOVO08 MINGW64 ~/Desktop
$ npm create vite@latest
√ Project name: ... hola
√ Select a framework: » React
√ Select a variant: » JavaScript

Scaffolding project in C:\Users\Usuario\Desktop\hola...

Done. Now run:

  cd hola
  npm install
  npm run dev
```

Crombie

Qué vamos a hacer?

- ➔ Listado de tareas (useState)
- ➔ Contador (useReducer/useState)
- ➔ Obtención de un nombre random a través de una api (useEffect)
- ➔ Crear dark mode para las aplicaciones (useContext)

Crombie

useState

- ➔ Permite agregar/gestionar un “estado” en un componente funcional.
- ➔ Se inicializa con un valor x. Ya sea string,number,boolean,etc.
- ➔ Retorna un arreglo de **dos** elementos: state y setState.
- ➔ Cuando se llama a la función setState, se actualiza la variable.

```
const [state, setstate] = useState();
```

Crombie

useEffect

- ➔ Permite realizar “efectos secundarios” en un componente funcional. Ej: llamadas a API.
- ➔ Se ejecuta después del primer renderizado o cuando cambian sus dependencias.
- ➔ Toma una función que contiene la lógica del efecto (por ejemplo, una llamada a una API).

```
    useEffect(() => {

fetch("https://randomuser.me/api/")
            .then(response =>
response.json())
            .then(data =>
setName(data.results[0].name.first))
            .catch(error =>
console.error(error));
    }, []);
```

Crombie

useEffect

- Se puede pasar un array como segundo argumento. Este array especifica qué variables deben cambiar para que el código vuelva a ejecutarse.
  - Sin dependencias ([]): El código se ejecuta solo una vez (al montar el componente).
  - Con dependencias ([dep1, dep2]): El código se ejecuta cada vez que cualquiera de las dependencias cambia.

useContext

- ➔ Permite generar un contexto global.
- ➔ Pasar props a componentes sin necesidad de que estos pasen de padre a hijo hasta llegar al nivel deseado.
- ➔ Útil para manejo de temas (Oscuro - Normal), autenticaciones o configuraciones globales.

useReducer

- ➔ Similar a useState pero para casos más complejos / múltiples opciones.
- ➔ Útil cuando el estado depende de varios factores.
- ➔ También devuelve una arreglo con dos variables, un state y una función de actualización “dispatch”.
- ➔ Se inicia con una función (**reducer**) y un estado inicial.

Vamos a la práctica.

- ➔ Listado de tareas (useState)
- ➔ Contador (useReducer/useState)
- ➔ Obtención de un nombre random a través de una api (useEffect)
- ➔ Crear dark mode para las aplicaciones (useContext)

Crombie