DEV.F::

Clase 3

Full Stack

Comenzamos en 10 min

6:40pm (hora CDMX)

DEV.F::

Full Stack

Full Stack: Backend y Frontend

Introducción

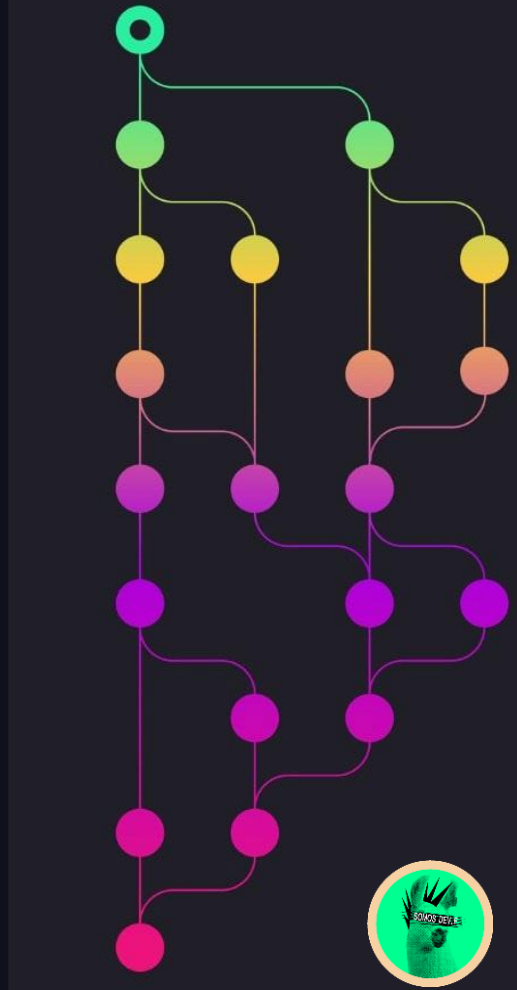
La arquitectura de aplicaciones Full Stack involucra tanto el backend como el frontend. El backend maneja la lógica de negocio, la base de datos y la autenticación, mientras que el frontend se encarga de la interfaz de usuario y la experiencia del usuario.

Backend en una Arquitectura Full Stack

El backend es la parte de la aplicación que gestiona los datos, la lógica del negocio y la comunicación con el frontend a través de APIs.

Sus principales componentes incluyen:

1. Servidor
2. Base de Datos
3. APIs (REST y GraphQL)



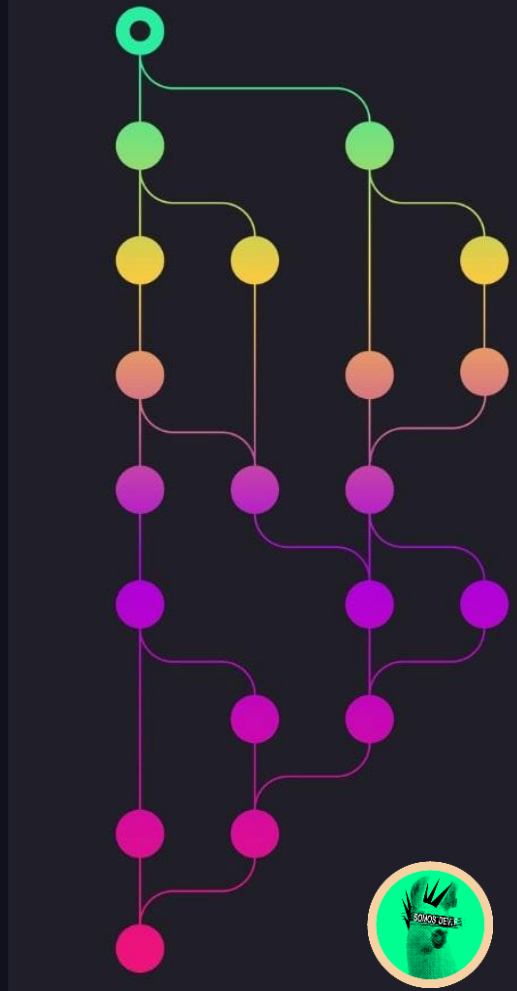
DEV.F::

1. Servidor

El servidor es el encargado de recibir peticiones y devolver respuestas. Se puede construir con tecnologías como:

Node.js con Express, Django con Python, Spring Boot con Java

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

const app = express();

app.get('/', (req, res) => {

  res.send('Hola, mundo!');

});

app.listen(3000, () => console.log('Servidor corriendo en puerto 3000'));
```



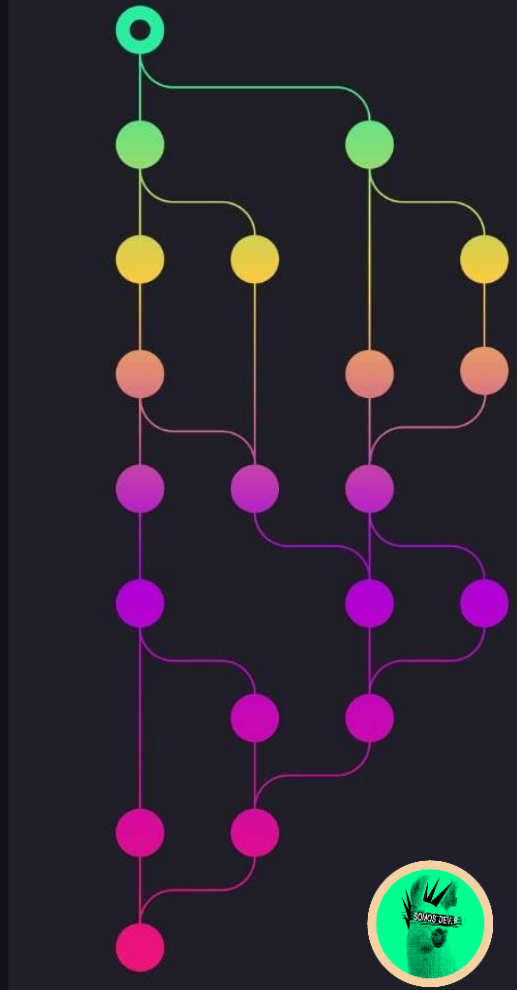
DEV.F::

2. Base de Datos

El backend interactúa con bases de datos SQL (PostgreSQL, MySQL) o NoSQL (MongoDB, Firebase) para almacenar información.

Ejemplo de conexión a MongoDB con Mongoose:

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

mongoose.connect('mongodb://localhost:27017/miapp');
```

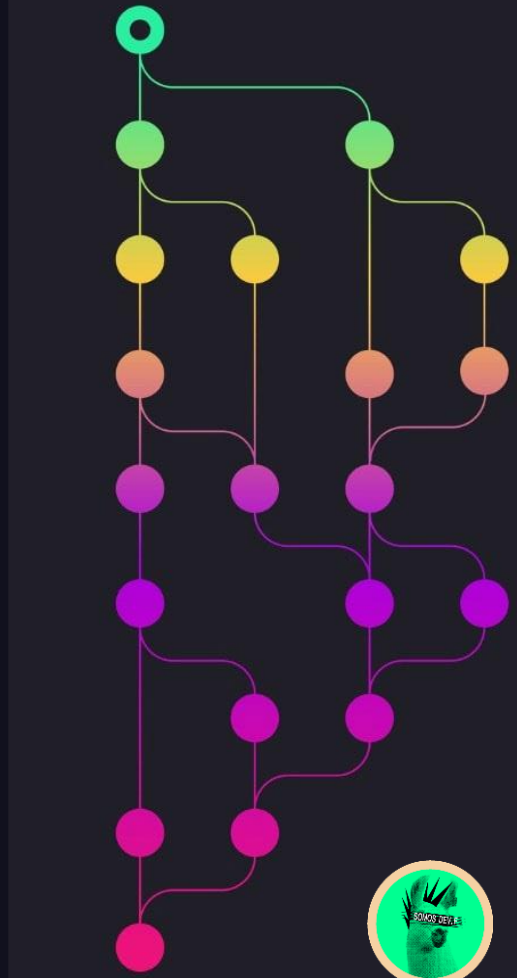


3. APIs (REST y GraphQL)

Las APIs permiten que el frontend interactúe con el backend.

Ejemplo de API REST en Express:

```
app.get('/usuarios', async (req, res) => {
  const usuarios = await Usuario.find();
  res.json(usuarios);
});
```

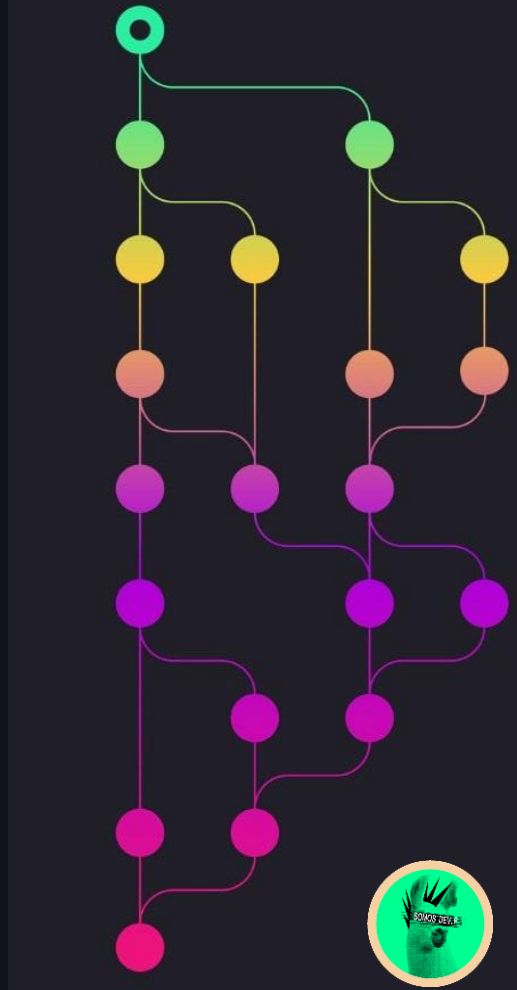


DEV.F::

Frontend en una Arquitectura Full Stack

El frontend es la capa visual y de interacción de la aplicación. Se construye con HTML, CSS y JavaScript, utilizando bibliotecas y frameworks como React, Vue o Angular.

React Vue.js Angular



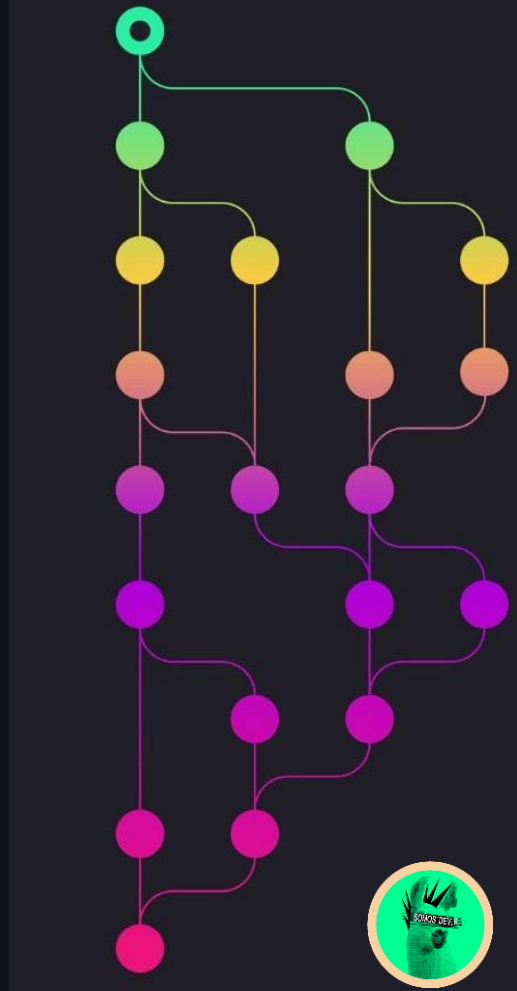
DEV.F::

Creación de Componentes Reutilizables

Los componentes en React facilitan la reutilización y modularización del código.

Ejemplo de un componente simple en React:

```
const Boton = ({ texto, onClick }) => {

  return <button onClick={onClick} className="btn">{texto}</button>;

};

export default Boton;
```



DEV.F::

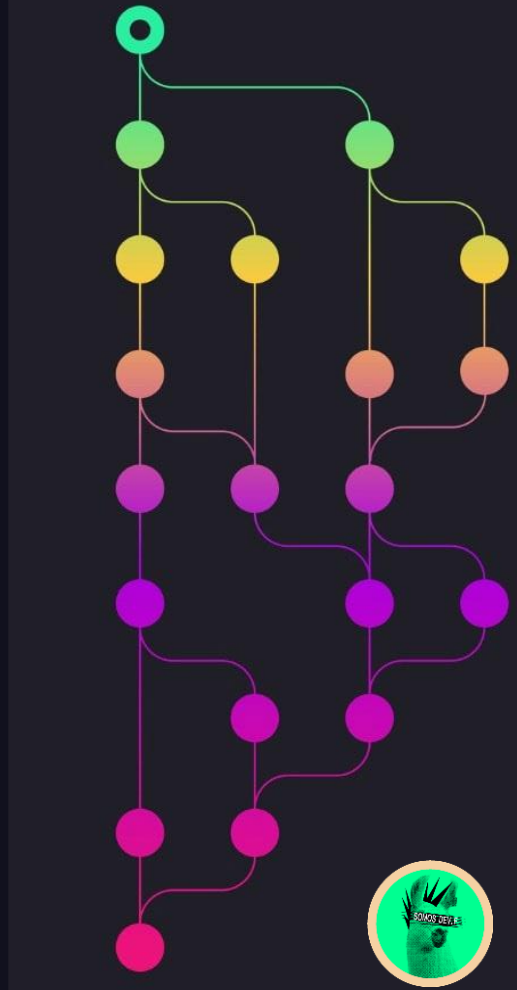
Manejo del Estado en el Frontend

Para gestionar datos en el frontend se pueden utilizar:

useState para estados locales.

useContext o **Redux** para estados globales.

```
import { useState } from 'react';

function Contador() {
  const [contador, setContador] = useState(0);

  return (
    <div>
      <p>Contador: {contador}</p>
      <button onClick={() => setContador(contador + 1)}>Incrementar</button>
    </div>
  );
}
```



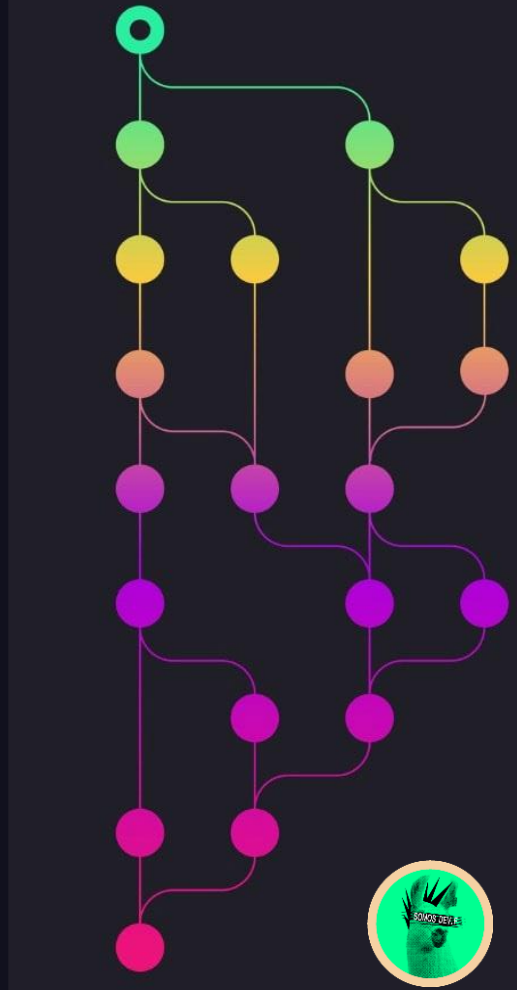
DEV.F::

Consumo de APIs en el Frontend

El frontend se comunica con el backend usando fetch o bibliotecas como Axios.

Ejemplo de consumo de API con fetch:

```
useEffect(() => {
  fetch('https://jsonplaceholder.typicode.com/users')
    .then(res => res.json())
    .then(data => setUsuarios(data));
}, []);
```

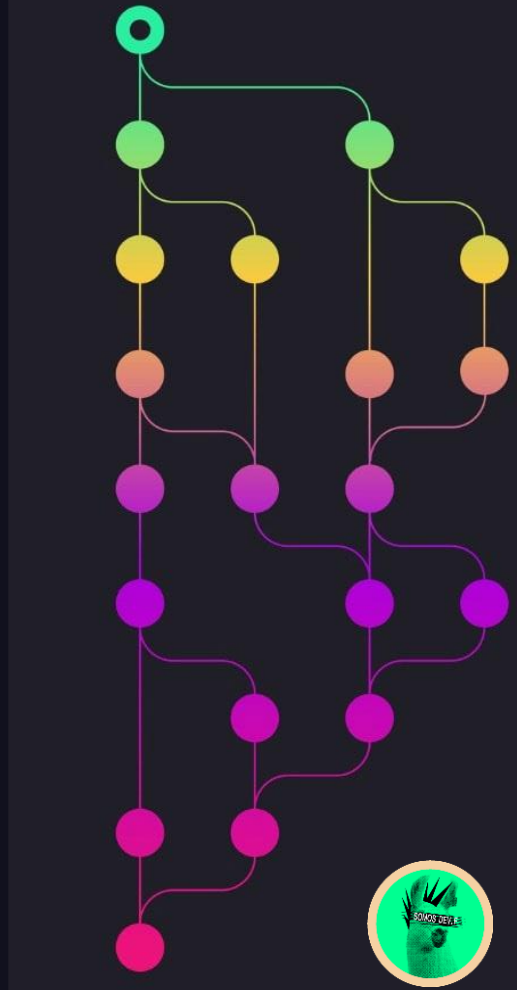


DEV.F::

Enrutamiento en el Frontend

React Router permite manejar rutas en aplicaciones de una sola página (SPA).

```jsx
import { BrowserRouter, Route, Routes } from 'react-router-dom';

import Home from './pages/Home';

import About from './pages/About';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```



Práctica

Ilustrar una pequeña aplicación Full Stack

La siguiente clase: Vamos a crear un repo y lo vamos a manipular desde nuestra computadora

Descripción del Problema

Necesitamos una aplicación que permita gestionar alumnos y sus asignaciones a grupos de clase. Los usuarios deben poder:

- Agregar nuevos alumnos.
- Ver la lista de alumnos y sus grupos.
- Editar la información de un alumno.
- Eliminar alumnos si es necesario.