

JOSÉ MANUEL MORA Z

MEDIATOR
MEDIATOR
MEDIATOR

DESIGN PATTERN: MEDIATOR

LAB 3 - SOFTWARE DESIGN



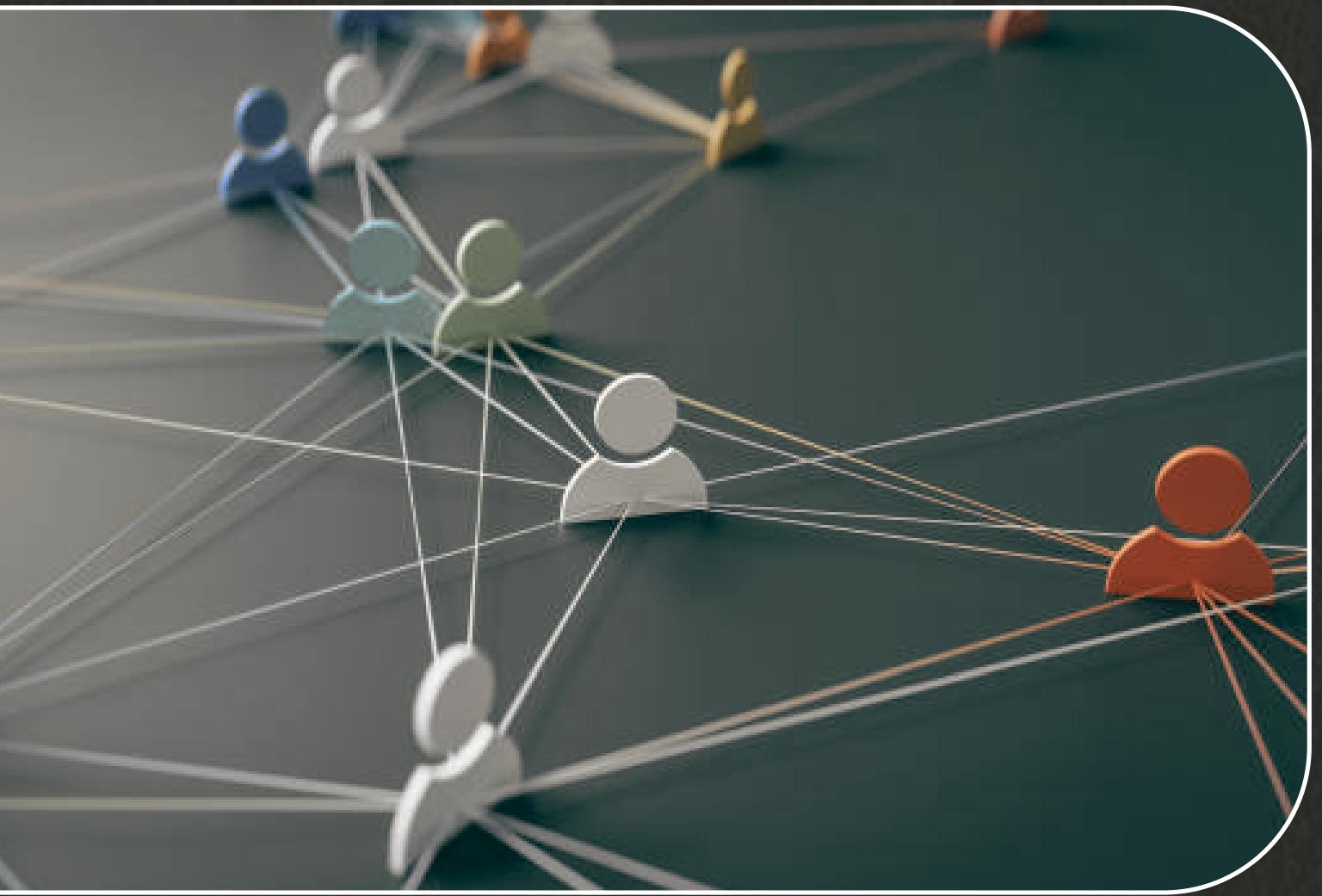
TABLA DE CONTENIDO

JOSÉ MANUEL MORA Z

03	MEDIADOR	08	REL. CON OTROS PATRONES
04	MANT. Y ESCALABILIDAD	09	PASOS PARA IMPLEMENTAR
05	ESTRUCTURA	10	CASOS DE USO
06	APLICABILIDAD	11	TESTING
07	VENTAJAS Y DESVENTAJAS	12	EJEMPLO DE CÓDIGO

MEDIADOR

JOSÉ MANUEL MORA Z



Es un patrón de comportamiento que busca resolver el fuerte acoplamiento que surge en sistemas con múltiples componentes que interactúan entre sí, cosa que dificulta el mantenimiento, escalabilidad y reutilización del software.

Para esto impone un objeto mediador para restringir la comunicación entre los componentes.

MANTENIMIENTO Y ESCALABILIDAD

JOSÉ MANUEL MORA Z

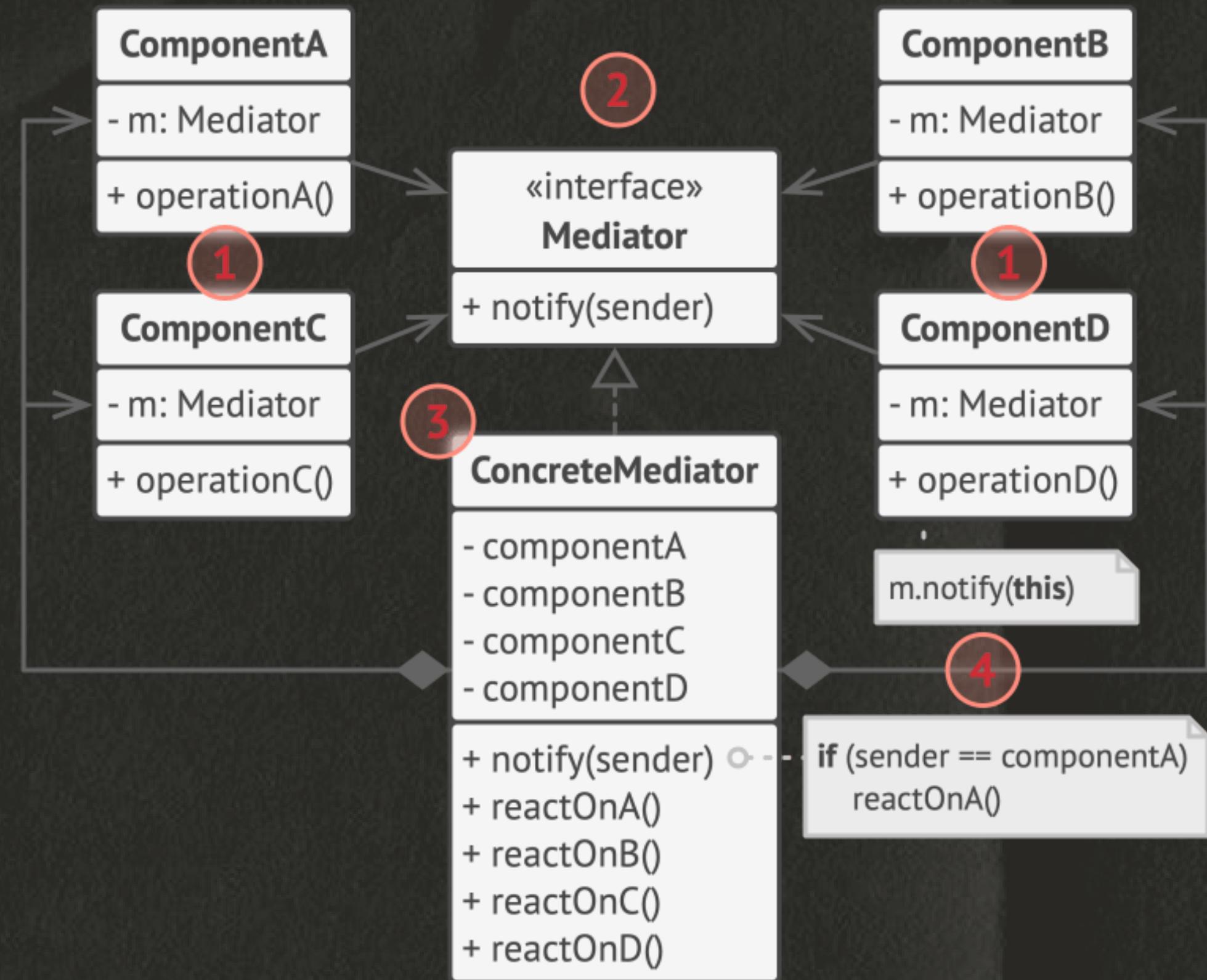
Al reducir las dependencias de cada componente, se mejoran significativamente el mantenimiento y la escalabilidad. Al estar aislados, los cambios a un componente no afectan a los otros.

Permite, además, cambiar a los objetos mediadores sin que se den cuenta, mientras la interfaz sea la misma.



ESTRUCTURA

JOSÉ MANUEL MORA Z



APLICABILIDAD

JOSÉ MANUEL MORA Z

- 1.Cuando las clases están acopladas y son difíciles de modificar
- 2.Cuando no se puede reutilizar un componente debido a que depende de otros componentes
- 3.Cuando se están creando muchas subclases para reutilizar comportamientos básicos.



- 1.El patrón permite extraer las relaciones entre las clases, aislando los cambios realizados a cada componente.
- 2.Al aplicar el patrón, las dependencias se reducen en gran medida.
- 3.Como el mediador es el que define las relaciones entre componentes, es fácil definir nuevos comportamientos sin crear nuevas subclases.

VENTAJAS Y DESVENTAJAS

JOSÉ MANUEL MORA Z

VENTAJAS

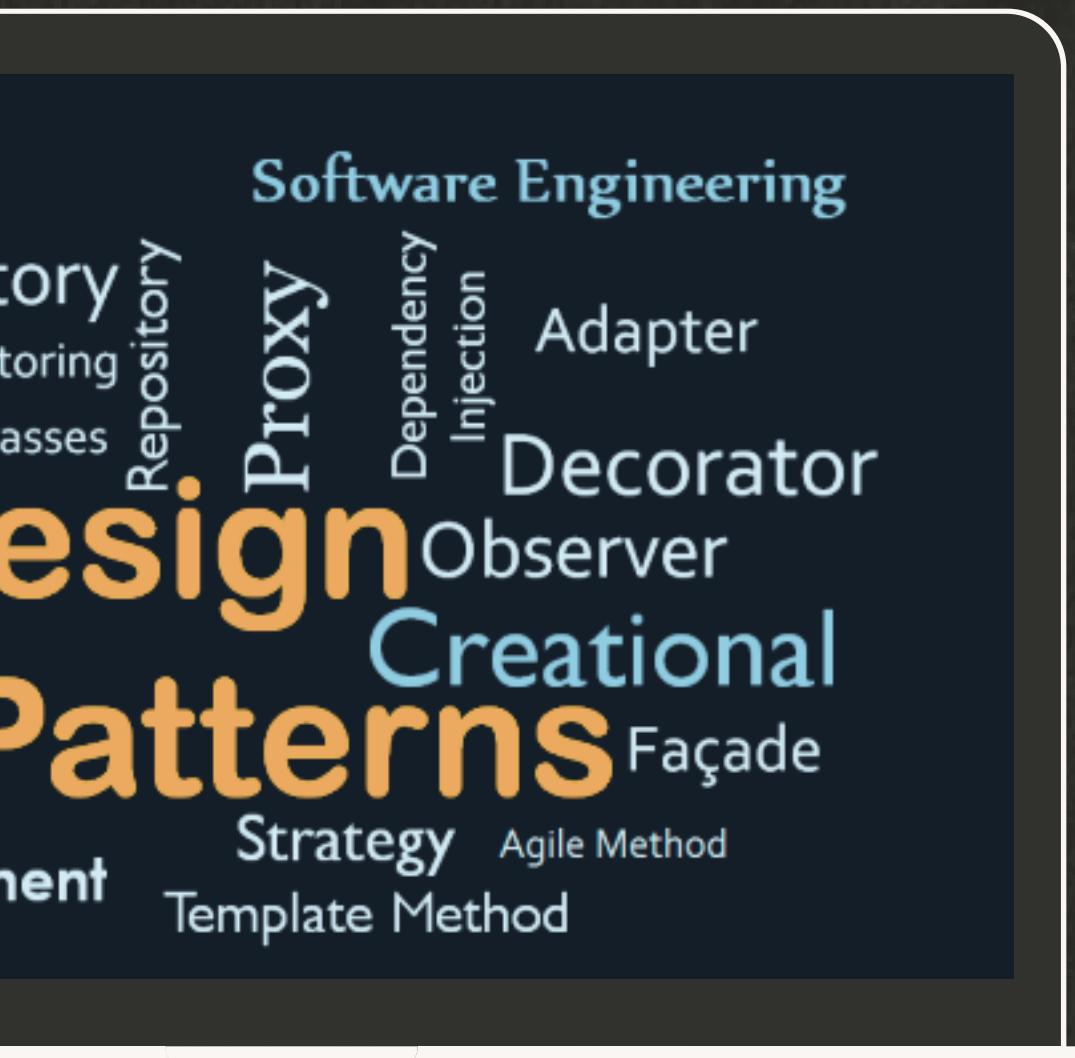
- + Responsabiliza al mediador de toda la comunicación (SRP)
- + Permite introducir nuevos mediadores sin cambiar los componentes (OCP)
- + Reduce acoplamiento entre componentes
- + Facilita reutilización de componentes

DESVENTAJAS

- Puede volverse un God Object si conoce demasiado sobre los componentes
- Crea overhead, que puede afectar el rendimiento del programa
- Overkill para programas simples
- Al centralizar todo el flujo, se vuelve un único punto de fallo.

RELACIÓN CON OTROS PATRONES

JOSÉ MANUEL MORA Z



MEDIATOR Y:

I

Chain of Resp., Command y Observer: Los 4 patrones abordan la comunicación entre objetos, aunque de forma distinta.

II

Facade: Ambos tienen un trabajo similar: organizar la colaboración entre objetos estrechamente acoplados.

PASOS PARA IMPLEMENTAR

JOSÉ MANUEL MORA Z

- 1.Identificar el grupo de componentes que puedan ocupar independencia
- 2.Declarar la interfaz del mediador y describir los protocolos de comunicación.
- 3.Implementar el mediador concreto.
- 4.(Extra) Responsabilizar al mediador de crear y destruir los componentes, para que los tenga guardados de una vez.
- 5.Darle a los componentes una referencia al mediador.
- 6.Cambiar la lógica de los componentes para que usen al mediador correctamente.

CASOS DE USO

JOSÉ MANUEL MORA Z

Casos cuándo usar el patrón:

- Comunicación compleja entre componentes
- Reducción del acoplamiento
- Centralización del control
- Modificar los componentes sin afectar otros
- Reutilización de componentes

Casos cuándo NO usar el patrón:

- Las interacciones son simples
- El mediador se vuelve demasiado complejo
- El rendimiento es crítico

TESTING

JOSÉ MANUEL MORA Z

TESTING DE COMPONENTES

En cuanto a componentes, estos se pueden testear más fácil, pues sus dependencias se ven reducidas ampliamente

TESTING DEL MEDIADOR

Para el mediador, ¿qué se testea?, pues la lógica de comunicación entre componentes en sí.

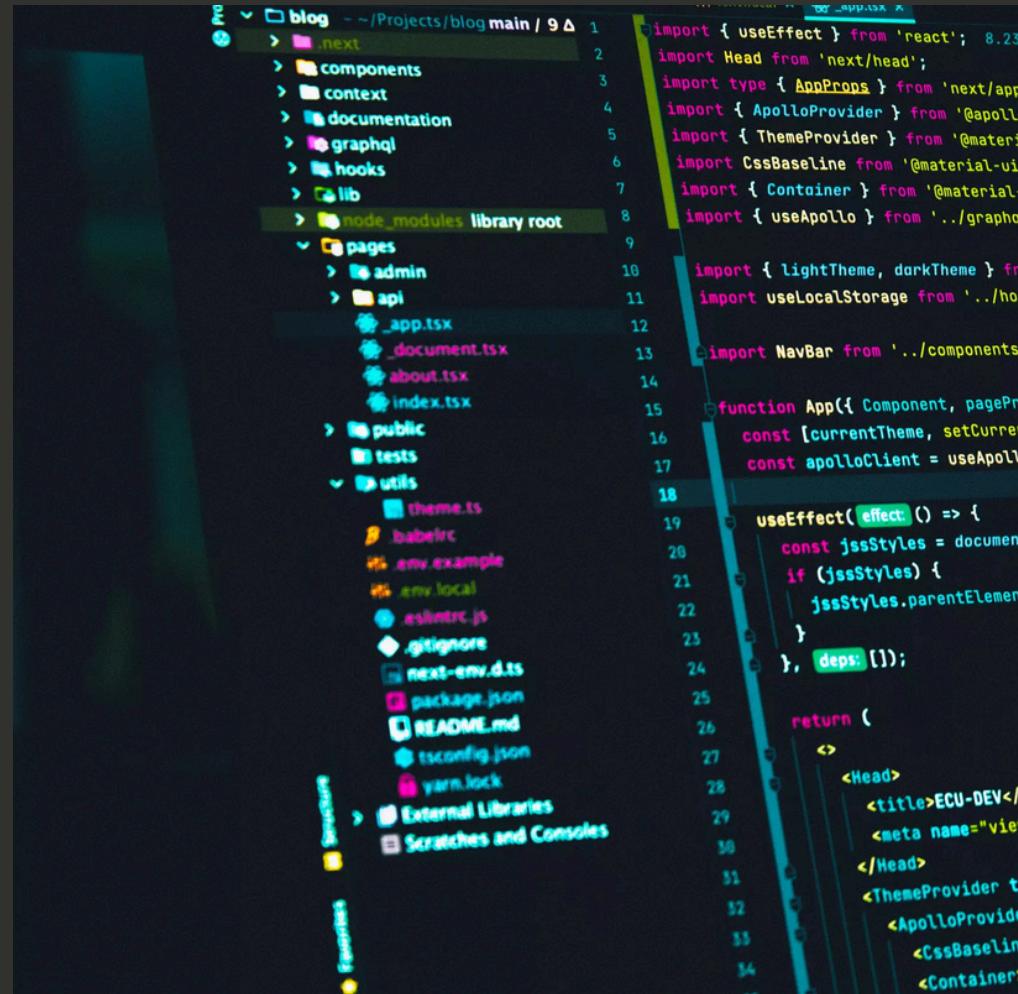
EJEMPLO DE CÓDIGO

JOSÉ MANUEL MORA Z

Ejemplo basado en el proyecto, simula la creación de un menú para dos usuarios, uno con restricciones alimenticias y el otro sin ninguna.

El mediador coordina las interacciones entre los servicios de menú, recetas y perfiles de usuario.

Cada componente maneja su propia lógica, mientras que el mediador solo coordina la comunicación



The screenshot shows a code editor with a dark theme. On the left is a file tree for a 'blog' project. The tree includes 'next', 'components', 'context', 'documentation', 'graphql', 'hooks', 'lib', 'node_modules' (marked as library root), 'pages' (containing 'admin', 'api', '_app.tsx', '_document.tsx', 'about.tsx', 'index.tsx'), 'public', 'tests', and 'utils' (containing 'themes.ts', 'babelrc', 'env.example', 'env.local', 'eslintrc.js', 'gitignore', 'next-env.d.ts', 'package.json', 'README.md', 'tsconfig.json', 'yarn.lock', 'External Libraries', and 'Scratches and Consoles'). On the right is a code editor window showing a snippet of 'App.tsx'. The code imports various components and uses hooks like 'useEffect' and 'useLocalStorage'. It defines a function 'App' that returns a component structure with 'Head', 'title', 'meta', 'ThemeProvider', 'ApolloProvider', 'CssBaseline', and 'Container'.

```
import { useEffect } from 'react';
import Head from 'next/head';
import type { AppProps } from 'next/app';
import { ApolloProvider } from '@apollo/client';
import { ThemeProvider } from '@material-ui/core';
import CssBaseline from '@material-ui/core/CssBaseline';
import { Container } from '@material-ui/core/Container';
import { useApollo } from '../graphql';

function App({ Component, pageProps }) {
  const [currentTheme, setCurrentTheme] = useState('light');
  const apolloClient = useApollo();

  useEffect(() => {
    const jssStyles = document.createElement('link');
    if (jssStyles) {
      jssStyles.parentElement = document.head;
    }
  }, [deps]);

  return (
    <Head>
      <title>ECU-DEV</title>
      <meta name="viewport" content="width=device-width, initial-scale=1" />
    </Head>
    <ThemeProvider theme={currentTheme}>
      <ApolloProvider client={apolloClient}>
        <CssBaseline />
        <Container>
          <Component {...pageProps} />
        </Container>
      </ApolloProvider>
    </ThemeProvider>
  );
}

export default App;
```

MEDIATOR
MEDIATOR

THANK YOU

MEDIATOR

JOSÉ MANUEL MORA Z