PARTE I — ESTILOS EN REACT

El problema de los estilos en aplicaciones grandes

En HTML tradicional se acostumbra a utilizar uno o pocos archivos CSS globales:

```
<link rel="stylesheet" href="styles.css">
```

A medida que una aplicación crece, comienzan a aparecer problemas:

- clases repetidas;
- nombres iguales;
- conflictos entre componentes;
- dificultad para mantener estilos;
- estilos que afectan lugares no deseados.

Por ejemplo, un desarrollador podría escribir algo como esto:

```
.boton{
   background: blue;
}
```

Mientras que otro podría escribir:

```
.boton{
   background: red;
}
```

¿Qué botón se verá rojo?

No resulta evidente. Esto recibe el nombre de **contaminación o colisión de estilos**.

Para evitar esto React propone que se pase de una página enorme con estilos a:

- Dato que tengo componentes independientes.
- Cada componente puede tener sus propios estilos.

Las estrategias de estilización en React

En React existen múltiples formas de aplicar estilos. Cada estrategia resuelve un problema distinto: organización, escalabilidad, aislamiento, reutilización o integración con componentes. Entenderlas permite elegir la más adecuada según el tamaño del proyecto y el equipo. A continuación, un mapa conceptual útil sugerido por la documentación oficial (aunque puede haber ligeros cambios definidos por el equipo de desarrollo) es:

```
Estilos en React
├── CSS tradicional
├── CSS modularizado (CSS Modules)
├── CSS-in-JS
│       ├── Styled Components
│       ├── Emotion
├── Frameworks CSS
│       ├── Bootstrap
│       ├── Tailwind
│       └── Material UI (MUI)
```

CSS Tradicional

La forma más simple y conocida. Se importa un archivo .css y se aplican clases como en HTML.

Por ejemplo:

```
import "./App.css";

.titulo {
  color: blue;
}

<h1 className="titulo">
  Hola React
</h1>
```

Ventajas

- Sencillo: no requiere aprender nada nuevo.
- Familiar: funciona igual que en proyectos web tradicionales.

Desventajas

- Riesgo de colisiones: dos componentes pueden usar la misma clase sin querer.
- Poca escalabilidad: difícil de mantener en proyectos grandes.
- Sin aislamiento: los estilos son globales por defecto.

CSS Modularizado (CSS modules)

React permite importar archivos `.module.css` que generan clases únicas automáticamente.

Esto evita colisiones entre nombres de clases y mantiene los estilos acotados al componente.

Por ejemplo, puede definir el archivo `Button.module.css`

```
.boton {
  background-color: green;
  color: white;
}
```

Que será utilizado por el componente de manera similar a lo siguiente, suponiendo que tiene un archivo `Button.jsx`

```
import styles from "./Button.module.css";

<button className={styles.boton}>
  Aceptar
</button>
```

Aquí hay algo que se debe destacar. React genera internamente:

```
.boton_xa47fd3
```

En lugar de

```
.boton
```

Es por este motivo que cada componente posee sus estilos aislados. CSS Modules encapsula estilos igual que los componentes encapsulan comportamiento.

Ventajas

- Aislamiento real: cada clase se vuelve única.
- Escalable: ideal para proyectos medianos y grandes.

- Sigue siendo CSS puro.

Desventajas

- Más verboso: hay que importar estilos en cada componente.
- Menos flexible que CSS-in-JS para estilos dinámicos.

CSS-in-JS

Los estilos se escriben directamente en JavaScript, como parte del componente.

Las librerías más usadas son:

- Styled Components
- Emotion

A continuación, se presente un ejemplo con Styled Components

```
import styled from "styled-components";

const Titulo = styled.h1`
  color: blue;
  font-size: 24px;
`;

<Titulo>Hola React</Titulo>
```

Ventajas

- Estilos dinámicos basados en props o estado.
- Aislamiento total: cada componente genera sus propias clases.
- Excelente para diseño de sistemas de componentes.

Desventajas

- Sobrecarga en runtime (aunque hoy es mínima).
- Aprendizaje adicional.
- Puede generar demasiada lógica dentro del componente si no se organiza bien.

Frameworks CSS

Los estilos se escriben directamente en JavaScript, como parte del componente. Por ejemplo:

Bootstrap

- ✓ Rápido para prototipos
- ✓ Sistema de grillas
- ✗ HTML muy cargado de clases
- ✗ No está pensado para React específicamente

Tailwind CSS

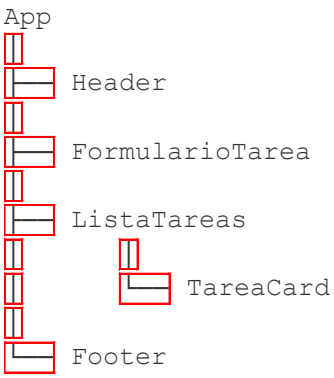
- ✓ Utilitario: clases pequeñas y combinables
- ✓ Muy usado en proyectos modernos
- ✓ Excelente para consistencia visual
- ✗ Curva de aprendizaje inicial
- ✗ HTML muy "ruidoso" si no se organiza

Material UI (MUI)

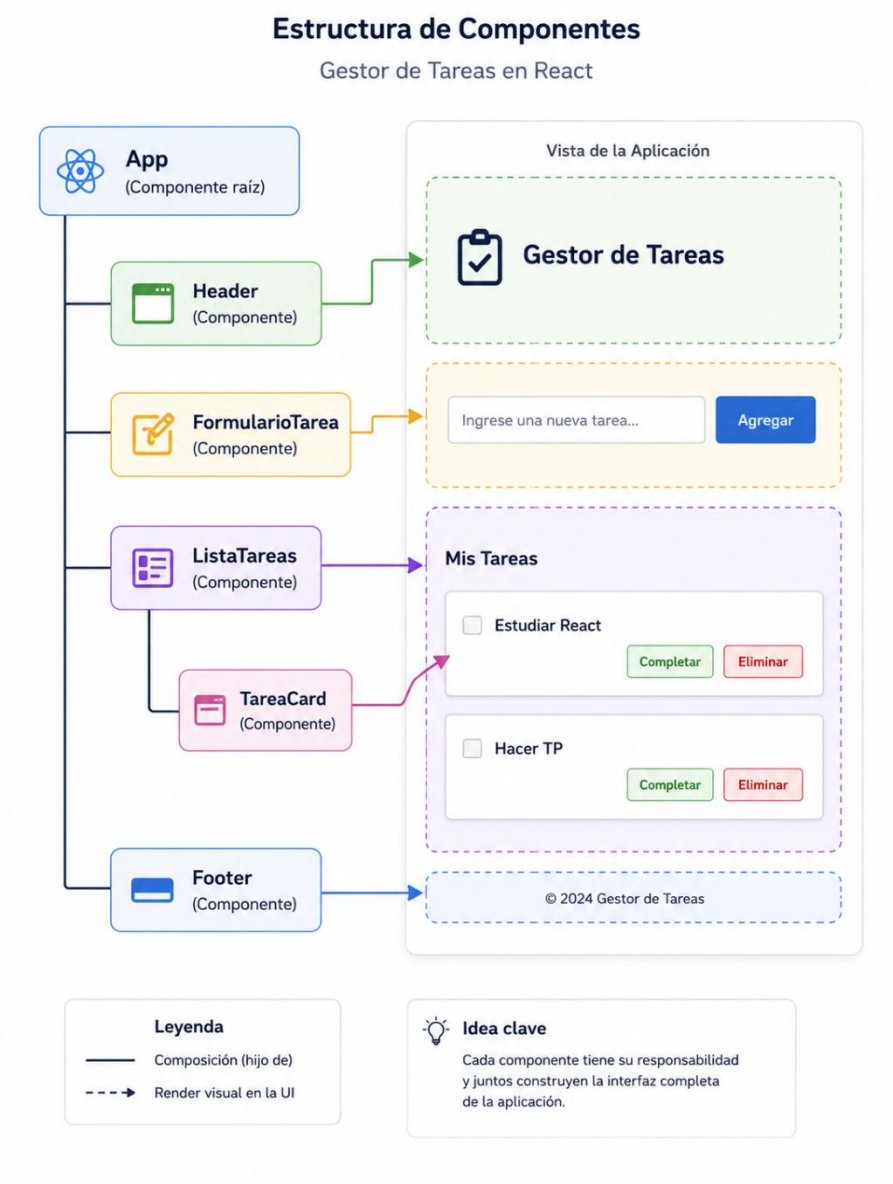
- ✓ Componentes listos, accesibles y modernos
- ✓ Muy usado en entornos profesionales
- ✓ Integración profunda con React
- ✗ Estética muy marcada
- ✗ La personalización avanzada puede ser compleja

Veamos el proceso de aplicación de estas diferentes estrategias de estilos en un mismo ejemplo.

El proyecto se trata de un Gestor de Tareas. La idea central es generar esta jerarquía de componentes:



La interfaz por generar debe tener la siguiente estructura:



La idea será construir exactamente esto tres veces usando diferentes estrategias de estilización.

Caso 1: Aplicando CSS Modules

Paso 1: Crear el proyecto con vite

```
npm create vite@latest gestor-tareas-cssmodules

cd gestor-tareas-cssmodules

npm install

npm run dev
```

Se obvia el proceso de selección de las plantillas adecuadas. No hace falta instalar bibliotecas adicionales.

Paso 2: Generar la siguiente estructura propuesta

```
src/
├── components/
│   ├── Header.jsx
│   ├── FormularioTarea.jsx
│   ├── ListaTareas.jsx
│   ├── TareaCard.jsx
│   └── Footer.jsx
├── styles/
│   ├── App.module.css
│   ├── Header.module.css
│   ├── FormularioTarea.module.css
│   ├── ListaTareas.module.css
│   ├── TareaCard.module.css
│   └── Footer.module.css
├── App.jsx
└── main.jsx
```

Paso 3: Establecer el contenido de cada componente y llamar a su estilo correspondiente

App.module.css

```
.app {
  max-width: 760px;
  margin: 40px auto;
  padding: 32px;
  font-family: Arial, Helvetica, sans-serif;
  background: #ffffff;
  border-radius: 18px;
  box-shadow: 0 8px 24px rgba(0, 0, 0, 0.08);
}
```

Header.module.css

```
.header {
  padding: 32px;
  margin-bottom: 24px;
  text-align: center;
  background: #eef8ef;
  border: 2px dashed #53a653;
  border-radius: 16px;
}

.header h1 {
  margin: 0;
  font-size: 2rem;
  color: #1f3b2c;
}

.header p {
  margin-top: 8px;
  color: #4b6b57;
}
```

FormularioTarea.module.css

```
.formulario {
  display: flex;
  gap: 12px;
  padding: 24px;
  margin-bottom: 24px;
  background: #fff7e6;
  border: 2px dashed #f0ad37;
  border-radius: 16px;
}

.formulario input {
  flex: 1;
  padding: 14px;
  font-size: 1rem;
  border: 1px solid #d8d8d8;
  border-radius: 8px;
}

.formulario button {
  padding: 14px 22px;
  font-weight: bold;
  color: white;
  background: #2563eb;
  border: none;
  border-radius: 8px;
  cursor: pointer;
}

.formulario button:hover {
  background: #1d4ed8;
}
```

ListaTareas.module.css

```
.lista {
  padding: 24px;
  margin-bottom: 24px;
  background: #faf5ff;
  border: 2px dashed #9b5de5;
  border-radius: 16px;
}

.lista h2 {
  margin-top: 0;
  color: #302044;
}

.vacio {
  padding: 16px;
  color: #777;
  background: white;
  border-radius: 8px;
}
```

TareaCard.module.css

```
.acciones {
  display: flex;
  gap: 8px;
}

.acciones button {
  padding: 9px 14px;
  font-weight: bold;
  border-radius: 8px;
  cursor: pointer;
}

.botonCompletar {
  color: #1f7a34;
  background: #effaf1;
  border: 1px solid #3fa65a;
}

.botonCompletar:hover {
  background: #dff3e4;
}

.botonEliminar {
  color: #b42318;
  background: #fff1f0;
  border: 1px solid #e5484d;
}

.botonEliminar:hover {
  background: #ffe0df;
}
```

Footer.module.css

```
.footer {
  padding: 18px;
  text-align: center;
  color: #31527a;
  background: #eef5ff;
  border: 2px dashed #4f8dd3;
  border-radius: 14px;
}
```

Header.jsx

```
import styles from "../styles/Header.module.css";

const Header = () => {
  return (
    <header className={styles.header}>
      <h1>Gestor de Tareas</h1>
      <p>Aplicación React con Vite y CSS Modules</p>
    </header>
  );
};

export default Header;
```

FormularioTarea.jsx

```
import { useState } from "react";
import styles from "../styles/FormularioTarea.module.css";

const FormularioTarea = ({ agregarTarea }) => {
  const [texto, setTexto] = useState("");

  const manejarSubmit = (e) => {
    e.preventDefault();

    if (texto.trim() === "") return;

    agregarTarea(texto);
    setTexto("");
  };

  return (
    <form className={styles.formulario} onSubmit={manejarSubmit}>
      <input
        type="text"
        placeholder="Ingrese una nueva tarea..."
        value={texto}
        onChange={(e) => setTexto(e.target.value)}
      />

      <button type="submit">Agregar</button>
    </form>
  );
};

export default FormularioTarea;
```

ListaTareas.jsx

```
import TareaCard from "./TareaCard";
import styles from "../styles/ListaTareas.module.css";

const ListaTareas = ({ tareas, cambiarEstado, eliminarTarea }) => {
  return (
    <section className={styles.lista}>
      <h2>Mis tareas</h2>

      {tareas.length === 0 ? (
        <p className={styles.vacio}>No hay tareas cargadas.</p>
      ) : (
        tareas.map((tarea) => (
          <TareaCard
            key={tarea.id}
            tarea={tarea}
            cambiarEstado={cambiarEstado}
            eliminarTarea={eliminarTarea}
          />
        ))
      )}
    </section>
  );
};

export default ListaTareas;
```

TareaCard.jsx

```
import styles from "../styles/TareaCard.module.css";

const TareaCard = ({ tarea, cambiarEstado, eliminarTarea }) => {
  return (
    <article className={styles.card}>
      <div className={styles.contenido}>
        <input
          type="checkbox"
          checked={tarea.completada}
          onChange={() => cambiarEstado(tarea.id)}
        />

        <span className={tarea.completada ? styles.completada : ""}>
          {tarea.texto}
        </span>
      </div>

      <div className={styles.acciones}>
        <button
          className={styles.botonCompletar}
          onClick={() => cambiarEstado(tarea.id)}
        >
          Completar
        </button>

        <button
          className={styles.botonEliminar}
          onClick={() => eliminarTarea(tarea.id)}
        >
          Eliminar
        </button>
      </div>
    </article>
  );
};

export default TareaCard;
```

Footer.jsx

```
import styles from "../styles/Footer.module.css";

const Footer = () => {
  return (
    <footer className={styles.footer}>
      <p>© 2026 Gestor de Tareas - Programación Visual</p>
    </footer>
  );
};

export default Footer;
```

App.jsx

```jsx
import { useState } from "react";
import Header from "./components/Header";
import FormularioTarea from "./components/FormularioTarea";
import ListaTareas from "./components/ListaTareas";
import Footer from "./components/Footer";
import styles from "./styles/App.module.css";

const App = () => {
  const [tareas, setTareas] = useState([
    { id: 1, texto: "Estudiar React", completada: false },
    { id: 2, texto: "Hacer TP", completada: false }
  ]);

  const agregarTarea = (texto) => {
    const nuevaTarea = {
      id: Date.now(),
      texto,
      completada: false
    };

    setTareas([...tareas, nuevaTarea]);
  };

  const cambiarEstado = (id) => {
    const nuevasTareas = tareas.map((tarea) =>
      tarea.id === id
        ? { ...tarea, completada: !tarea.completada }
        : tarea
    );

    setTareas(nuevasTareas);
  };

  const eliminarTarea = (id) => {
    const nuevasTareas = tareas.filter((tarea) => tarea.id !== id);
    setTareas(nuevasTareas);
  };

  return (
    <main className={styles.app}>
      <Header />

      <FormularioTarea agregarTarea={agregarTarea} />

      <ListaTareas
        tareas={tareas}
        cambiarEstado={cambiarEstado}
        eliminarTarea={eliminarTarea}
      />

      <Footer />
    </main>
  );
};

export default App;
```

Al ejecutar esta aplicación verás este resultado:

Caso 2: Aplicando CSS-in-JS con styled-components

Paso 1: Crear el proyecto con vite

```
npm create vite@latest gestor-tareas-cssinjs
cd gestor-tareas-cssinjs
npm install
npm run dev
```

Se obvia el proceso de selección de las plantillas adecuadas.

Paso 2: Instalar styled-components

Si estás ejecutando el servidor lo detienes con Crtl + C. Luego necesitas instalar las bibliotecas que permitirán que los estilos vivan dentro del propio archivo del componente, como componentes estilizados.

```
npm install styled-components
```

Luego, podemos volver a ejecutar el servidor.

Paso 3: Generar la siguiente estructura propuesta

```
src/
├── components/
│   ├── Header.jsx
│   ├── FormularioTarea.jsx
│   ├── ListaTareas.jsx
│   ├── TareaCard.jsx
│   └── Footer.jsx
├── App.jsx
└── main.jsx
```

En esta versión no tendremos carpeta `styles/`, porque los estilos se escribirán dentro de cada componente usando `styled-components`.

Paso 4: Establecer el contenido de cada componente

main.jsx

```jsx
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import App from "./App.jsx";

createRoot(document.getElementById("root")).render(
  <StrictMode>
    <App />
  </StrictMode>
);
```

App.jsx

```jsx
import { useState } from "react";
import styled from "styled-components";
import Header from "./components/Header";
import FormularioTarea from "./components/FormularioTarea";
import ListaTareas from "./components/ListaTareas";
import Footer from "./components/Footer";

const ContenedorApp = styled.main`
  max-width: 760px;
  margin: 40px auto;
  padding: 32px;
  font-family: Arial, Helvetica, sans-serif;
  background: #ffffff;
  border-radius: 18px;
  box-shadow: 0 8px 24px rgba(0, 0, 0, 0.08);
`;

const App = () => {
  const [tareas, setTareas] = useState([
    { id: 1, texto: "Estudiar React", completada: false },
    { id: 2, texto: "Hacer TP", completada: false }
  ]);

  const agregarTarea = (texto) => {
    const nuevaTarea = {
      id: Date.now(),
      texto,
      completada: false
    };

    setTareas([...tareas, nuevaTarea]);
  };
```

```
  const cambiarEstado = (id) => {
    const nuevasTareas = tareas.map((tarea) =>
      tarea.id === id
        ? { ...tarea, completada: !tarea.completada }
        : tarea
    );

    setTareas(nuevasTareas);
  };

  const eliminarTarea = (id) => {
    const nuevasTareas = tareas.filter((tarea) => tarea.id !== id);
    setTareas(nuevasTareas);
  };

  return (
    <ContenedorApp>
      <Header />

      <FormularioTarea agregarTarea={agregarTarea} />

      <ListaTareas
        tareas={tareas}
        cambiarEstado={cambiarEstado}
        eliminarTarea={eliminarTarea}
      />

      <Footer />
    </ContenedorApp>
  );
};

export default App;
```

Header.jsx

```
import styled from "styled-components";

const Encabezado = styled.header`
  padding: 32px;
  margin-bottom: 24px;
  text-align: center;
  background: linear-gradient(135deg, #eef8ef, #f7fff8);
  border: 2px dashed #53a653;
  border-radius: 16px;

  h1 {
    margin: 0;
    font-size: 2rem;
    color: #1f3b2c;
  }

  p {
    margin-top: 8px;
    color: #4b6b57;
  }
`;

const Header = () => {
  return (
    <Encabezado>
      <h1>Gestor de Tareas</h1>
      <p>Aplicación React con Vite y CSS-in-JS</p>
    </Encabezado>
  );
};

export default Header;
```

Desarrollados estos dos componentes, cabe responder la pregunta: ¿Cómo se da cuenta React como debe aplicar los estilos?

En `App.jsx` observemos esta línea

```
const ContenedorApp = styled.main`
  max-width:760px;
  margin:40px auto;
  padding:32px;
`;
```

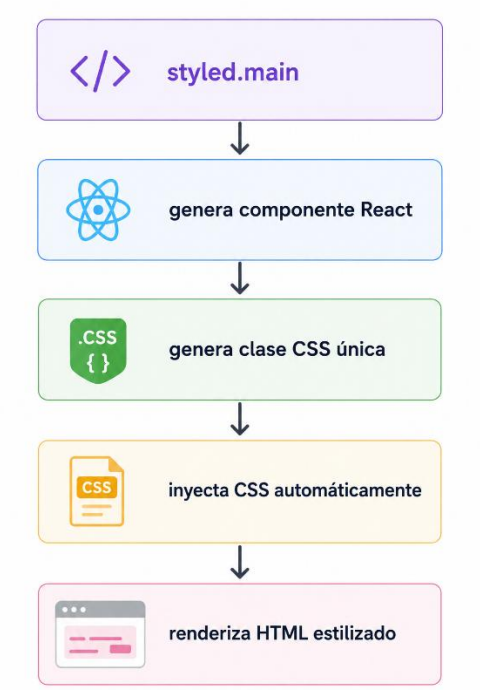
A primera vista podría parecer que estamos creando una variable común. Sin embargo, eso no es exactamente lo que ocurre.

La función:

```
styled.main
```

le indica a la biblioteca styled-components que cree un nuevo componente React basado en la etiqueta HTML <main> y que le asocie los estilos indicados.

Internamente sucede algo similar a:



Por eso posteriormente utilizamos:

```
<ContenedorApp>
```

en lugar de:

```
<main className="contenedor">
```

La diferencia conceptual es importante. En CSS tradicional tenemos una etiqueta HTML como esto:

```
<main class="contenedor">
```

Donde el estilo referenciado está probablemente en un archivo de estilos con algo similar a esto:

```
.contenedor{

   padding:32px;

}
```

Pero gracias a la biblioteca de CSS-in-js, internamente lo que realiza React es esto:

```
const ContenedorApp=styled.main`
   padding:32px;
`;

<ContenedorApp>
```

El estilo deja de ser un archivo separado y pasa a formar parte del propio componente.

De manera similar, en Header.jsx tenemos:

```
const Encabezado = styled.header`

  padding:32px;
  margin-bottom:24px;

  h1{

     margin:0;
     color:#1f3b2c;

  }

`;
```

Aquí ocurren varias cosas simultáneamente. Primero:

```
styled.header
```

crea un componente basado en la etiqueta `<header>`

Luego:

```
h1{

margin:0;

}
```

establece reglas internas que afectan únicamente a los elementos contenidos dentro de ese componente. Y React genera algo parecido a:

```
.header_x8fd42 h1{
    margin:0;
}
```

En lugar de

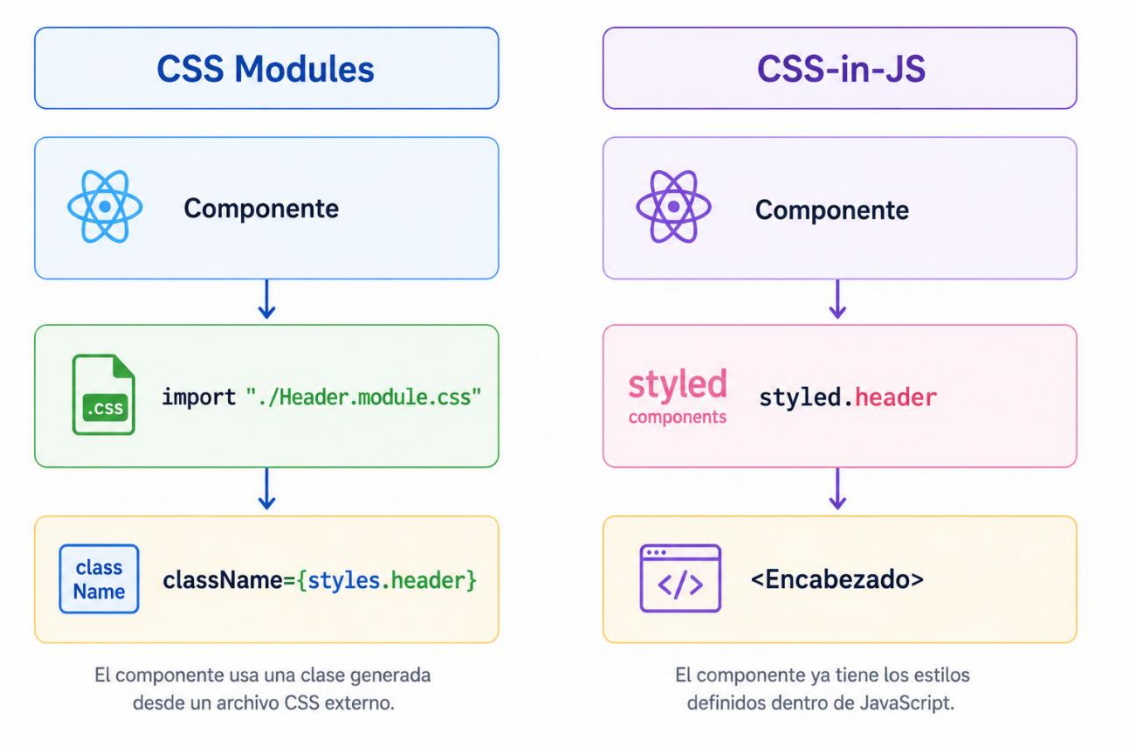
```
h1{
    margin:0;
}
```

La ventaja es enorme:

- evita colisiones;
- encapsula estilos;
- mantiene coherencia;
- facilita mantenimiento.

El componente y el estilo pasan a comportarse como una sola unidad lógica.

El siguiente grafico compara los procesos conceptuales que ocurren detrás de cada una de estas estrategias



Ahora que entendemos el funcionamiento conceptual de la estrategia CSS-in-js podemos mostrar como están conformados los otros componentes

FormularioTarea.jsx

```
import { useState } from "react";
import styled from "styled-components";
```

```
const Formulario = styled.form`
  display: flex;
  gap: 12px;
  padding: 24px;
  margin-bottom: 24px;
  background: linear-gradient(135deg, #fff7e6, #fffdf5);
  border: 2px dashed #f0ad37;
  border-radius: 16px;

  input {
    flex: 1;
    padding: 14px;
    font-size: 1rem;
    border: 1px solid #d8d8d8;
    border-radius: 8px;
  }

  button {
    padding: 14px 22px;
    font-weight: bold;
    color: white;
    background: #2563eb;
    border: none;
    border-radius: 8px;
    cursor: pointer;
  }

  button:hover {
    background: #1d4ed8;
  }
`;

const FormularioTarea = ({ agregarTarea }) => {
  const [texto, setTexto] = useState("");

  const manejarSubmit = (e) => {
    e.preventDefault();

    if (texto.trim() === "") return;

    agregarTarea(texto);
    setTexto("");
  };

  return (
    <Formulario onSubmit={manejarSubmit}>
      <input
        type="text"
        placeholder="Ingrese una nueva tarea..."
        value={texto}
        onChange={(e) => setTexto(e.target.value)}
      />

      <button type="submit">Agregar</button>
    </Formulario>
  );
}; export default FormularioTarea;
```

ListaTareas.jsx

```
import styled from "styled-components";
import TareaCard from "./TareaCard";

const SeccionLista = styled.section`
  padding: 24px;
  margin-bottom: 24px;
  background: linear-gradient(135deg, #faf5ff, #ffffff);
  border: 2px dashed #9b5de5;
  border-radius: 16px;

  h2 {
    margin-top: 0;
    color: #302044;
  }
`;

const MensajeVacio = styled.p`
  padding: 16px;
  color: #777;
  background: white;
  border-radius: 8px;
`;

const ListaTareas = ({ tareas, cambiarEstado, eliminarTarea }) => {
  return (
    <SeccionLista>
      <h2>Mis tareas</h2>

      {tareas.length === 0 ? (
        <MensajeVacio>No hay tareas cargadas.</MensajeVacio>
      ) : (
        tareas.map((tarea) => (
          <TareaCard
            key={tarea.id}
            tarea={tarea}
            cambiarEstado={cambiarEstado}
            eliminarTarea={eliminarTarea}
          />
        ))
      )}
    </SeccionLista>
  );
};

export default ListaTareas;
```

TareaCard.jsx

```
import styled from "styled-components";

const Card = styled.article`
  display: flex;
  justify-content: space-between;
  align-items: center;
  gap: 12px;
  padding: 18px;
  margin-bottom: 14px;
  background: white;
  border: 1px solid #e6e6e6;
  border-left: 6px solid ${(props) =>
    props.$completada ? "#3fa65a" : "#9b5de5"};
  border-radius: 12px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.06);
`;

const Contenido = styled.div`
  display: flex;
  align-items: center;
  gap: 12px;

  span {
    font-size: 1rem;
    font-weight: 500;
    color: ${(props) => (props.$completada ? "#777" : "#222")};
    text-decoration: ${(props) =>
      props.$completada ? "line-through" : "none"};
  }
`;

const Acciones = styled.div`
  display: flex;
  gap: 8px;
`;

const Boton = styled.button`
  padding: 9px 14px;
  font-weight: bold;
  border-radius: 8px;
  cursor: pointer;
  transition: 0.2s;
`;

const BotonCompletar = styled(Boton)`
  color: #1f7a34;
  background: #effaf1;
  border: 1px solid #3fa65a;

  &:hover {
    background: #dff3e4;
  }
`;
```

```
const BotonEliminar = styled(Boton)`
  color: #b42318;
  background: #fff1f0;
  border: 1px solid #e5484d;

  &:hover {
    background: #ffe0df;
  }
`;

const TareaCard = ({ tarea, cambiarEstado, eliminarTarea }) => {
  return (
    <Card $completada={tarea.completada}>
      <Contenido $completada={tarea.completada}>
        <input
          type="checkbox"
          checked={tarea.completada}
          onChange={() => cambiarEstado(tarea.id)}
        />

        <span>{tarea.texto}</span>
      </Contenido>

      <Acciones>
        <BotonCompletar onClick={() => cambiarEstado(tarea.id)}>
          Completar
        </BotonCompletar>

        <BotonEliminar onClick={() => eliminarTarea(tarea.id)}>
          Eliminar
        </BotonEliminar>
      </Acciones>
    </Card>
  );
};

export default TareaCard;
```

Footer.jsx

```
import styled from "styled-components";

const Pie = styled.footer`
  padding: 18px;
  text-align: center;
  color: #31527a;
  background: #eef5ff;
  border: 2px dashed #4f8dd3;
  border-radius: 14px;

  p {
    margin: 0;
  }
`;

const Footer = () => {
  return (
    <Pie>
      <p>© 2026 Gestor de Tareas - Programación Visual</p>
    </Pie>
  );
};

export default Footer;
```

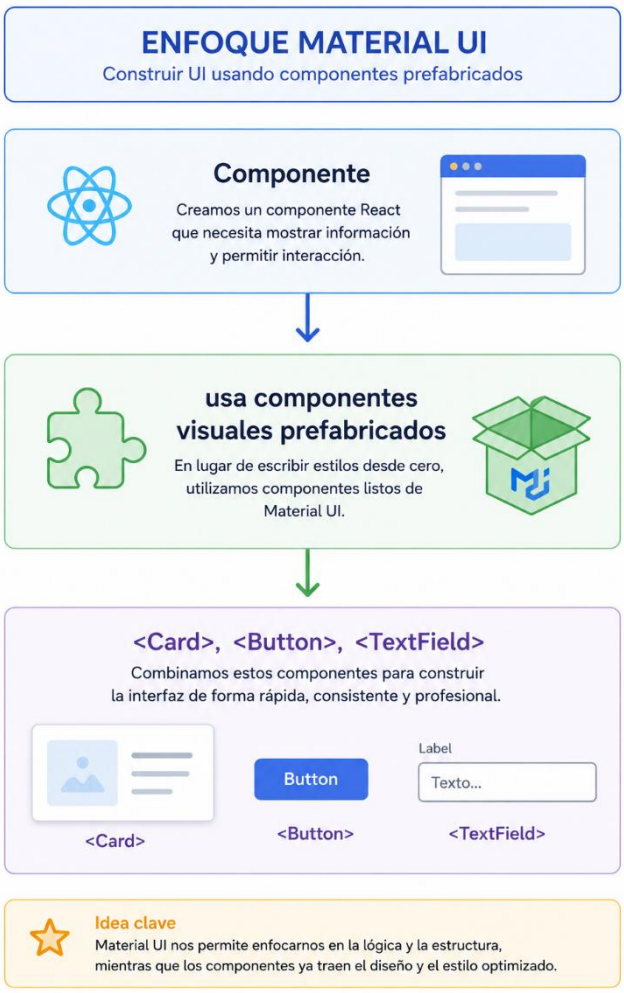
Notarás que al ejecutar el proyecto React obtendrás el mismo estilo aplicado a los componentes del caso 1.

Caso 3: Material UI con VITE

Queremos generar la misma aplicación, pero el diseño visual debe ser generado a partir de componentes predefinidos por la biblioteca Material UI. En este caso usaremos:

- Container
- Box
- Typography
- TextField
- Button
- Card
- CardContent
- Stack
- Checkbox
- Paper

Material UI plantea un diseño conceptual que difiere a los vistos en los dos casos anteriores. Este enfoque está representado por la siguiente imagen



Material UI permite construir interfaces profesionales rápidamente porque ya ofrece:

- componentes predefinidos;
- diseño consistente;
- buenas prácticas visuales;
- sistema de espaciado;
- variantes de botones;
- formularios;
- tarjetas;
- tematización;
- accesibilidad básica.

Paso 1: Crear el proyecto con vite

```
npm create vite@latest gestor-tareas-mui
cd gestor-tareas-mui
npm install
```

Se obvia el proceso de selección de las plantillas adecuadas.

Paso 2: Instalar material-ui

```
npm install @mui/material @emotion/react @emotion/styled
```

Opcionalmente podemos instalar los íconos de la biblioteca mediante esto

```
npm install @mui/icons-material
```

Luego, podemos volver a ejecutar el servidor.

Paso 3: Generar la siguiente estructura propuesta

```
src/
├── components/
│   ├── Header.jsx
│   ├── FormularioTarea.jsx
│   ├── ListaTareas.jsx
│   ├── TareaCard.jsx
│   └── Footer.jsx
├── App.jsx
└── main.jsx
```

En esta versión no vamos a crear archivos CSS propios. La interfaz se arma con componentes de Material UI dentro de los archivos definidos en la carpeta, en este caso estos:

```
<Container>
<Box>
<Paper>
<Typography>
<TextField>
<Button>
<Card>
<Stack>
<Checkbox>
```

Paso 4: Establecer el contenido de cada componente

main.jsx

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import App from "./App.jsx";

createRoot(document.getElementById("root")).render(
  <StrictMode>
    <App />
  </StrictMode>
);
```

App.jsx

```
import { useState } from "react";
import { Container, Paper, Stack } from "@mui/material";
import Header from "./components/Header";
import FormularioTarea from "./components/FormularioTarea";
import ListaTareas from "./components/ListaTareas";
import Footer from "./components/Footer";

const App = () => {
  const [tareas, setTareas] = useState([
    { id: 1, texto: "Estudiar React", completada: false },
    { id: 2, texto: "Hacer TP", completada: false }
  ]);

  const agregarTarea = (texto) => {
    const nuevaTarea = {
      id: Date.now(),
      texto,
      completada: false
    };

    setTareas([...tareas, nuevaTarea]);
  };

  const cambiarEstado = (id) => {
    const nuevasTareas = tareas.map((tarea) =>
      tarea.id === id
        ? { ...tarea, completada: !tarea.completada }
        : tarea
    );

    setTareas(nuevasTareas);
  };

  const eliminarTarea = (id) => {
    const nuevasTareas = tareas.filter((tarea) => tarea.id !== id);
    setTareas(nuevasTareas);
  };
```

```
  return (
    <Container maxWidth="md" sx={{ py: 5 }}>
      <Paper
        elevation={4}
        sx={{
          p: 4,
          borderRadius: 4,
          backgroundColor: "#ffffff"
        }}
      >
        <Stack spacing={3}>
          <Header />

          <FormularioTarea agregarTarea={agregarTarea} />

          <ListaTareas
            tareas={tareas}
            cambiarEstado={cambiarEstado}
            eliminarTarea={eliminarTarea}
          />

          <Footer />
        </Stack>
      </Paper>
    </Container>
  );
};

export default App;
```

En Material UI usamos componentes visuales prediseñados. Para el caso de App.jsx estamos usando estos:

- `<Container>`: Centra y limita el ancho del contenido
- `<Paper>`: Crea una superficie visual con sombra
- `<Stack>`: Organiza elementos con separación

Como puedes notar cada uno de ellos tiene una función visual establecida.

Además, usamos la propiedad sx:

```
sx={{
  p: 4,
  borderRadius: 4,
  backgroundColor: "#ffffff"
}}
```

sx permite aplicar estilos directamente sobre componentes de Material UI. Es una forma rápida de personalizar componentes sin crear archivos CSS.

Con estas aclaraciones sobre el mecanismo para aplicar Material-UI podemos generar los demás componentes.

Header.jsx

```
import { Box, Typography } from "@mui/material";

const Header = () => {
  return (
    <Box
      sx={{
        p: 4,
        textAlign: "center",
        borderRadius: 3,
        background: "linear-gradient(135deg, #e3f2fd, #f5fbff)",
        border: "1px solid #bbdefb"
      }}
    >
      <Typography variant="h4" component="h1" fontWeight="bold" co-
lor="primary">
        Gestor de Tareas
      </Typography>

      <Typography variant="body1" color="text.secondary" sx={{ mt: 1
}}>
        Aplicación React con Vite y Material UI
      </Typography>
    </Box>
  );
};

export default Header;
```

Aquí podemos distinguir algunos elementos conceptuales sobre el uso de Material-UI. Material UI usa componentes visuales en lugar de etiquetas HTML puras.

Aquí usamos `<Box>`. Es un componente contenedor flexible. Puede comportarse como un `div`, pero tiene acceso directo al sistema de estilos de Material UI mediante sx.

También usamos `<Typography>`. Sirve para textos, títulos y párrafos con estilos consistentes. Por ejemplo

```
<Typography variant="h4" component="h1">
```

Significa que visualmente se ve como h4 de Material UI, pero semánticamente se renderiza como h1. Esto es útil porque permite separar la apariencia visual de la semántica HTML.

FormularioTarea.jsx

```
import { useState } from "react";
import { Paper, Stack, TextField, Button } from "@mui/material";

const FormularioTarea = ({ agregarTarea }) => {
  const [texto, setTexto] = useState("");

  const manejarSubmit = (e) => {
    e.preventDefault();

    if (texto.trim() === "") return;

    agregarTarea(texto);
    setTexto("");
  };
```

```
  return (
    <Paper
      component="form"
      elevation={0}
      onSubmit={manejarSubmit}
      sx={{
        p: 3,
        borderRadius: 3,
        backgroundColor: "#fff8e1",
        border: "1px solid #ffe082"
      }}
    >
      <Stack direction={{ xs: "column", sm: "row" }} spacing={2}>
        <TextField
          fullWidth
          label="Nueva tarea"
          placeholder="Ingrese una nueva tarea..."
          value={texto}
          onChange={(e) => setTexto(e.target.value)}
        />

        <Button type="submit" variant="contained" size="large">
          Agregar
        </Button>
      </Stack>
    </Paper>
  );
};

export default FormularioTarea;
```

Este componente combina lógica de React con componentes visuales de Material UI:

- `<TextField>`: reemplaza al input HTML básico.
- `<Button>`: reemplaza al botón tradicional
- `<Paper>`: crea una superficie visual
- `<Stack>`: ordena los elementos con espaciado

Analicemos lo siguiente:

```
<Stack direction={{ xs: "column", sm: "row" }} spacing={2}>
```

Indica que en pantallas pequeñas los elementos se apilan verticalmente, mientras que en pantallas medianas o mayores se muestran en fila. Es decir, Material UI ayuda también al diseño responsivo.

ListaTareas.jsx

```
import { Paper, Typography, Stack, Alert } from "@mui/material";
import TareaCard from "./TareaCard";

const ListaTareas = ({ tareas, cambiarEstado, eliminarTarea }) => {
  return (
    <Paper
      elevation={0}
      sx={{
        p: 3,
        borderRadius: 3,
        backgroundColor: "#faf5ff",
        border: "1px solid #e1bee7"
      }}
    >
      <Typography variant="h5" fontWeight="bold" sx={{ mb: 2 }}>
        Mis tareas
      </Typography>

      {tareas.length === 0 ? (
        <Alert severity="info">No hay tareas cargadas.</Alert>
      ) : (
        <Stack spacing={2}>
          {tareas.map((tarea) => (
            <TareaCard
              key={tarea.id}
              tarea={tarea}
              cambiarEstado={cambiarEstado}
              eliminarTarea={eliminarTarea}
            />
          ))}
        </Stack>
      )}
    </Paper>
  );
};

export default ListaTareas;
```

TareaCard.jsx

```
import {
  Card,
  CardContent,
  Stack,
  Checkbox,
  Typography,
  Button
} from "@mui/material";

const TareaCard = ({ tarea, cambiarEstado, eliminarTarea }) => {
  return (
    <Card
      variant="outlined"
      sx={{
        borderRadius: 3,
        borderLeft: tarea.completada
          ? "6px solid #2e7d32"
          : "6px solid #7b1fa2",
        backgroundColor: tarea.completada ? "#f1f8e9" : "#ffffff"
      }}
    >
      <CardContent>
        <Stack
          direction={{ xs: "column", sm: "row" }}
          justifyContent="space-between"
          alignItems={{ xs: "flex-start", sm: "center" }}
          spacing={2}
        >
          <Stack direction="row" alignItems="center" spacing={1}>
            <Checkbox
              checked={tarea.completada}
              onChange={() => cambiarEstado(tarea.id)}
            />

            <Typography
              variant="body1"
              sx={{
                fontWeight: 500,
                textDecoration: tarea.completada ? "line-through" :
"none",
                color: tarea.completada ? "text.secondary" :
"text.primary"
              }}
            >
              {tarea.texto}
            </Typography>
          </Stack>

          <Stack direction="row" spacing={1}>
            <Button
              variant="outlined"
              color="success"
              onClick={() => cambiarEstado(tarea.id)}
            >
              Completar
            </Button>
```

```
            <Button
              variant="outlined"
              color="error"
              onClick={() => eliminarTarea(tarea.id)}
            >
              Eliminar
            </Button>
          </Stack>
        </Stack>
      </CardContent>
    </Card>
  );
};

export default TareaCard;
```

Aquí se ve muy bien el enfoque de Material UI. En lugar de escribir:

```
<article className="card">
```

usamos:

```
<Card>
```

El componente `Card` ya trae una estructura visual preparada.

Luego usamos `<CardContent>` para definir el contenido interno de la tarjeta.

El estilo se personaliza con sx:

```
sx={{
  borderRadius: 3,
  borderLeft: tarea.completada
    ? "6px solid #2e7d32"
    : "6px solid #7b1fa2"
}}
```

Esto permite que el estilo dependa del estado de la tarea. Si está completada:

- borde verde
- fondo verde claro
- texto tachado

Si está pendiente

- borde violeta
- fondo blanco
- texto normal

Footer.jsx

```
import { Box, Typography } from "@mui/material";

const Footer = () => {
  return (
    <Box
      component="footer"
      sx={{
        p: 2,
        textAlign: "center",
        borderRadius: 3,
        backgroundColor: "#e3f2fd",
        border: "1px solid #bbdefb"
      }}
    >
      <Typography variant="body2" color="text.secondary">
        © 2026 Gestor de Tareas - Programación Visual
      </Typography>
    </Box>
  );
};

export default Footer;
```

Notarás que al ejecutar el proyecto React obtendrás el mismo estilo similar a los componentes del caso 1 y 2, pero con detalles sutiles que la diferencian, además de que este diseño es responsivo