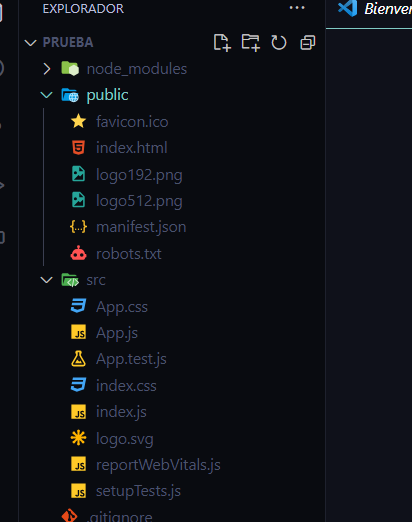
**Diseño ecommerce con React – Pasos Previos**

1. Se puede optar por isntalar ‘create-react-app’ de manera global para este y proyectos futuros. No obstante, si al intentar crear un proyecto con el paso 2. Se detecta que no está, será instalado automáticamente.
2. Usar el comando npx créate-react-app ecommerceapp (o cualquiera que sea el nombre dado. No usar mayúsculas)
3. Al crear un proyecto nuevo (en limpio), vemos una estructura de directorios similares a:



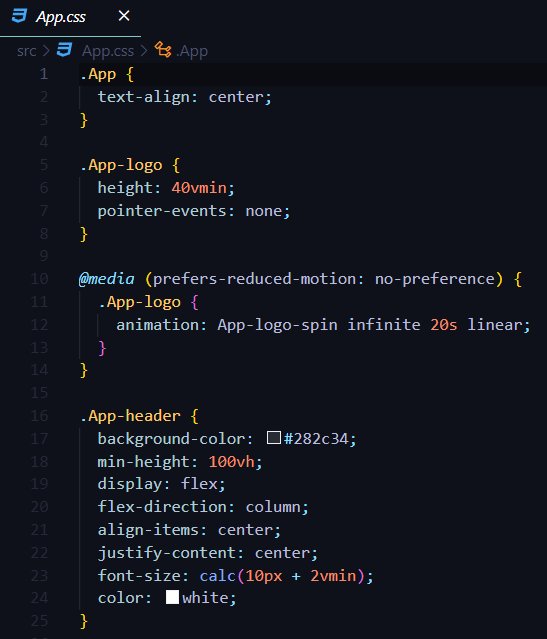
En la carpeta public vemos un archivo index.html de manera predeterminada, que tiene el siguiente código:

Texto

Descripción generada automáticamente

Por otra parte, en el directorio src encontramos los archivos predeterminados: App.css, App.js, index.css, index.js, entre otros.

En App.css encontramos el siguiente código:



Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

En App.js encontramos de manera predeterminada el código:

Texto

Descripción generada automáticamente

En index.css encontramos:

Captura de pantalla de computadora

Descripción generada automáticamente

Y en index.js tenemos:

Texto

Descripción generada automáticamente

ChatGPT da la siguiente explicación sobre estas estructuras de directorios predeterminadas:

**¿Qué secuencia sigue el proceso de renderización en el proyecto React?**

Es decir, que cuando ejecuto mi proyecto en el navegador, el proceso no empieza por public/index.html sino por index.js

ChatGPT

**ChatGPT**

Exacto, cuando ejecutas tu proyecto de React en el navegador, el proceso de inicio no comienza por **public/index.html**, sino por **src/index.js**. El archivo **public/index.html** actúa más como un archivo de plantilla base que proporciona la estructura HTML inicial para tu aplicación, pero la ejecución y la lógica de inicio están definidas en el archivo **src/index.js**.

Cuando abres tu aplicación en el navegador, el navegador carga **public/index.html** como la página principal. En este archivo, encontrarás un contenedor **<div>** con un **id** de "root":

htmlCopy code

<div id="root"></div>

Luego, cuando React inicia tu aplicación, busca este elemento con el **id** "root" en el DOM y utiliza el archivo **src/index.js** para renderizar dinámicamente la aplicación en ese contenedor específico. El código en **src/index.js** utiliza **ReactDOM.render()** para realizar esta operación.

Entonces, el flujo de ejecución es:

1. El navegador carga **public/index.html** como la página principal.
2. El navegador encuentra el **<div id="root"></div>** en **public/index.html**.
3. React inicia la aplicación desde **src/index.js** utilizando **ReactDOM.render()** y renderiza la aplicación en el elemento con el **id** "root" en **public/index.html**.

En resumen, aunque **public/index.html** proporciona la estructura HTML inicial, la lógica de inicio y la renderización de la aplicación comienzan desde **src/index.js**. Este enfoque facilita la creación de aplicaciones dinámicas de una sola página (SPA) mediante React.

¿Qué ventajas tiene que el proceso empiece por src/index.js y no por public/index.html?

ChatGPT

**ChatGPT**

Que el proceso comience por **src/index.js** en lugar de **public/index.html** en una aplicación de React tiene varias ventajas:

1. **Programación Declarativa:** React se basa en un enfoque declarativo para construir interfaces de usuario. Al comenzar desde **src/index.js**, donde se define la estructura de la aplicación en código JavaScript, se favorece un estilo de programación declarativa. Esto facilita la comprensión y mantenimiento del código, ya que te centras en qué debería hacer la aplicación en lugar de en cómo debería hacerlo.
2. **Componentización:** React fomenta la construcción de aplicaciones mediante componentes reutilizables. Al iniciar desde **src/index.js**, puedes organizar y componer tus componentes en una jerarquía clara y modular. Esto facilita la reutilización de componentes en diferentes partes de la aplicación y mejora la mantenibilidad del código.
3. **Enlace Dinámico:** React permite enlazar dinámicamente los datos del estado de la aplicación con la interfaz de usuario. Comenzar desde **src/index.js** te brinda la capacidad de establecer conexiones dinámicas entre los datos y la representación visual de la aplicación. Esto es esencial para construir aplicaciones interactivas y reactivas.
4. **Recarga en Caliente (Hot-Reloading):** El proceso de inicio desde **src/index.js** facilita la implementación de características como la recarga en caliente, que permite ver los cambios en tiempo real mientras desarrollas sin tener que recargar toda la página. Esto es especialmente útil para aumentar la productividad y acelerar el proceso de desarrollo.
5. **Configuración y Personalización:** Iniciar desde **src/index.js** te brinda la oportunidad de configurar y personalizar el entorno de desarrollo. Puedes establecer configuraciones específicas, agregar middleware, y configurar herramientas como Babel y Webpack directamente desde el punto de entrada.
6. **Facilita la Integración de Herramientas y Bibliotecas:** Iniciar desde **src/index.js** también facilita la integración de herramientas y bibliotecas externas. Puedes incorporar fácilmente soluciones de gestión de estado, enrutamiento, y otras bibliotecas, y ajustar su integración desde el punto de entrada principal.

En resumen, comenzar desde **src/index.js** en una aplicación de React proporciona un punto de entrada centralizado y programático que se alinea con los principios de diseño y las características de React, facilitando la construcción de aplicaciones modulares, declarativas y dinámicas.

ACLARACIÓN

**Conclusión:**

En resumen, el proceso inicia con la carga de **public/index.html** en el navegador, pero la lógica y el control de la aplicación React comienzan con **src/index.js**. La inserción dinámica en el DOM real y la gestión eficiente de las actualizaciones se llevan a cabo en este proceso, que implica la utilización del Virtual DOM para lograr un rendimiento optimizado. Ambas partes están relacionadas y trabajan juntas para proporcionar una experiencia de usuario eficiente y receptiva.

Cuando se inicia el proceso de renderizar los diferentes componentes en el index.js, primero se actualizan o se llaman todos los estilos css asociados a cada componentes todo dentro del DOM virtual para después sí pasar al archivo index.html, creería yo. O todo se realiza en simultáneo con el archivo public/index.html?

ChatGPT

**ChatGPT**

Cuando React inicia el proceso de renderizado, la actualización de los estilos CSS asociados a cada componente generalmente sigue un flujo específico, que involucra el Virtual DOM. Aquí hay un resumen del proceso:

1. **Inicio del Proceso:** Cuando se inicia la aplicación React, el navegador carga el archivo **public/index.html** que contiene la estructura HTML básica, incluido el contenedor con un **id**, como **<div id="root"></div>**, donde se montará dinámicamente la aplicación React.
2. **Inicio desde src/index.js:** El archivo **src/index.js** es el punto de entrada principal. Aquí se utiliza **ReactDOM.render()** para renderizar la aplicación React en el contenedor especificado en **public/index.html**. La creación y manipulación del Virtual DOM comienzan en este punto.
3. **Renderizado de Componentes:** Durante el proceso de renderizado, React renderiza cada componente de manera secuencial según la jerarquía del árbol de componentes. Cuando un componente se renderiza, React también se encarga de gestionar los estilos asociados a ese componente.
4. **Actualización del Virtual DOM:** La actualización de estilos en el Virtual DOM es parte del proceso de renderizado. Si un componente tiene estilos asociados, React los aplica en el Virtual DOM.
5. **Reconciliación y Actualización Eficiente:** React compara el Virtual DOM actualizado con el Virtual DOM anterior y determina las diferencias. Se realizan operaciones mínimas en el DOM real para reflejar estas diferencias, incluyendo la actualización de estilos.
6. **Inserción en el DOM Real:** Después de la reconciliación y la determinación de las actualizaciones necesarias, React realiza la inserción dinámica

Es decir, que en vez de llamar todos los componentes desde index.js como el ejemplo que me diste más arriba, puedo o más debo llamar los componentes adicionales desde App.js

ChatGPT

**ChatGPT**

Exacto, en una aplicación React, especialmente cuando se vuelve más compleja, es común organizar y estructurar tus componentes de manera jerárquica. El componente principal, generalmente llamado **App** (pero podría tener otro nombre según tus preferencias), actúa como un contenedor que organiza y renderiza otros componentes.

En lugar de tener un solo archivo **App.js** grande que contiene toda la lógica y la interfaz de usuario, se recomienda dividir tu aplicación en componentes más pequeños y especializados. Cada uno de estos componentes puede tener su propio archivo y código dedicado a una tarea o responsabilidad específica.

Por ejemplo, puedes tener una estructura de archivos como esta:

plaintextCopy code

src/ |-- index.js |-- App.js |-- components/ | |-- Header.js | |-- Sidebar.js | |-- Content.js | |-- Footer.js

Y en **App.js**, puedes importar y utilizar estos componentes adicionales:

jsxCopy code

import React from 'react'; import './App.css'; import Header from './components/Header'; import Sidebar from './components/Sidebar'; import Content from './components/Content'; import Footer from './components/Footer'; function App() { return ( <div className="App"> <Header /> <Sidebar /> <Content /> <Footer /> </div> ); } export default App;

Esta estructura facilita la gestión y el mantenimiento de tu aplicación, ya que cada componente se encarga de una parte específica de la interfaz de usuario o de la lógica. Además, mejora la reutilización de componentes y la legibilidad del código.

Recuerda que la organización de archivos y componentes puede variar según la preferencia del equipo y la naturaleza de la aplicación, pero la idea principal es mantener una estructura modular y fácil de entender.

**Pregunta: Al crear un proyecto nuevo en React, veo que lo siguiente en el archivo index.js, donde no entiendo por qué aparece createRoot en lugar de ReactDOM.render. import React from 'react';**

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

Respuesta: ¡Gracias por preguntar! createRoot es una nueva API de React que se introdujo en la versión 17. Es una alternativa a ReactDOM.render que está diseñada para mejorar el rendimiento y la capacidad de respuesta de las aplicaciones de React.

La principal diferencia entre createRoot y ReactDOM.render es que createRoot crea un nuevo contenedor raíz para renderizar componentes, mientras que ReactDOM.render renderiza componentes directamente en un elemento DOM existente. createRoot también proporciona una API más flexible para trabajar con componentes de React en modo concurrente.

En el código que compartiste, ReactDOM.createRoot se utiliza para crear un nuevo contenedor raíz para la aplicación de React. Luego, root.render() se utiliza para renderizar el componente App en el contenedor raíz. Esto es similar a cómo se utiliza ReactDOM.render, pero con la ventaja adicional de que createRoot puede mejorar el rendimiento y la capacidad de respuesta de la aplicación.

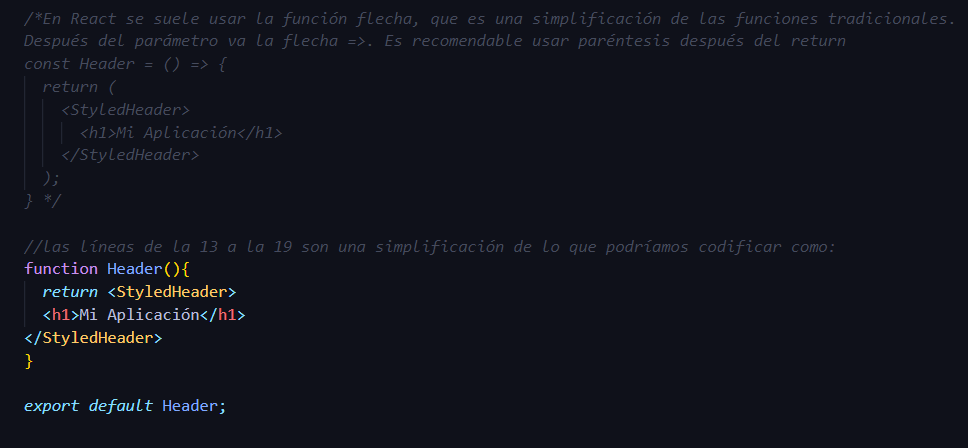
**Pregunta: Y eso de <React.StrictMode> ¿Qué es?**

React.StrictMode es una herramienta de desarrollo que ayuda a encontrar problemas potenciales en una aplicación de React. Se utiliza para envolver componentes y habilitar comprobaciones adicionales y advertencias en el modo de desarrollo. Estas comprobaciones y advertencias pueden ayudar a encontrar errores comunes en los componentes de React y solucionarlos antes de que se publique la aplicación.

En el código que compartiste, <React.StrictMode> se utiliza para envolver el componente App y habilitar estas comprobaciones adicionales y advertencias. Esto puede ayudar a encontrar errores comunes en la aplicación de React y solucionarlos antes de que se publique la aplicación.

**Funciones Flecha (Arrow Function)**

En React, se aconseja la nueva manera de trabajar funciones, es decir, usar la nomenclatura funciones flecha. En este caso, después del parámetro se coloca una flecha =>. En vez de darle como tal un nombre a la función, se usa una constante o variable para asignar el valor obtenido con la función flecha. Ver el siguiente ejemplo:



En este caso, ambas formas serían válidas (incluyendo lo que está comentado).