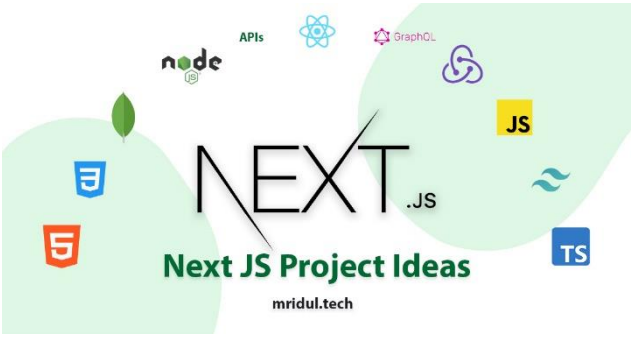
CUADERNO PERSONAL

DESARROLLO DE APLICACIONES WEB



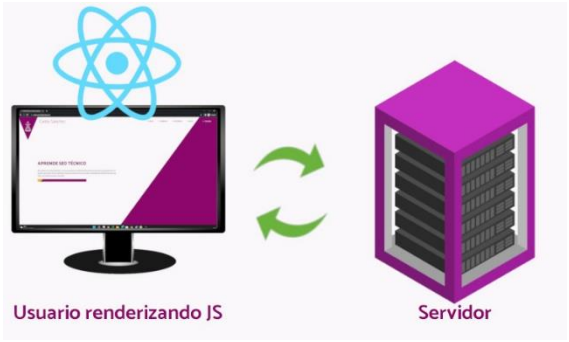
SEMANA 9 – Next JS – CSR/SSR

Introducción a Next.js

Next.js es un framework de React para la construcción de aplicaciones web modernas. Proporciona características avanzadas como renderizado del lado del servidor (SSR), generación estática de páginas (SSG), y soporte para la creación de API. Next.js es ampliamente utilizado por su capacidad para optimizar el rendimiento de las aplicaciones y mejorar la experiencia del desarrollador.

Características Principales de Next.js

1. **Renderizado del lado del servidor (SSR):** Next.js permite renderizar páginas en el servidor antes de enviarlas al cliente. Esto mejora el rendimiento inicial y la optimización para motores de búsqueda (SEO).
2. **Renderizado del lado del cliente (CSR):** Similar a una aplicación React estándar, donde el contenido se renderiza completamente en el cliente después de que se carga el JavaScript.



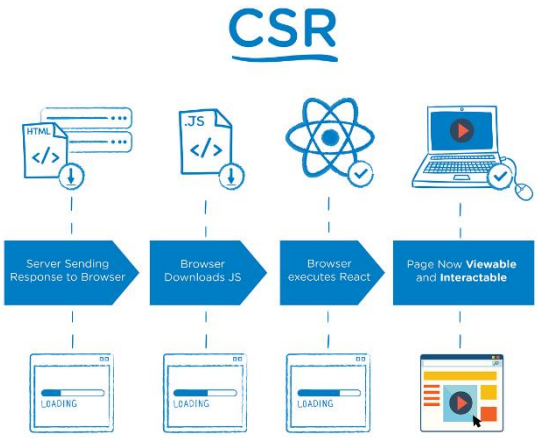
3. **Generación estática de páginas (SSG):** Permite pre-renderizar páginas a HTML estático en el momento de la construcción, lo cual es ideal para páginas que no cambian con frecuencia.
4. **Soporte para API:** Next.js permite crear endpoints API dentro de la misma estructura del proyecto, facilitando la creación de aplicaciones full-stack.

5. **Enrutamiento automático:** Los archivos en la carpeta pages se convierten automáticamente en rutas, eliminando la necesidad de configurar enrutadores manualmente.
6. **Soporte integrado para CSS y Sass:** Next.js soporta estilos CSS, CSS Modules, y preprocesadores como Sass, facilitando el manejo de estilos.

CSR (Client-Side Rendering)

Client-Side Rendering (CSR) es una técnica de renderizado en la que el contenido de la página se genera completamente en el cliente, es decir, en el navegador del usuario. En aplicaciones de React tradicionales, CSR es el enfoque por defecto, donde la aplicación se inicializa con un archivo HTML vacío y el contenido se renderiza utilizando JavaScript.

- **Ventajas de CSR:**



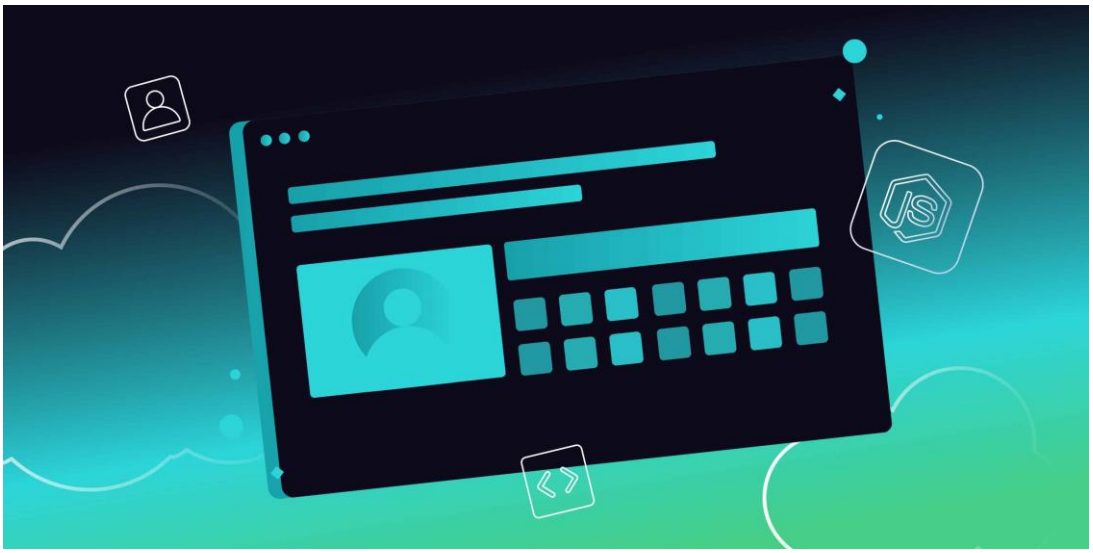
  - **Interactividad:** La aplicación puede ser muy interactiva y dinámica ya que el renderizado ocurre en el navegador.
  - **Menor carga en el servidor:** El servidor solo necesita servir archivos estáticos y manejar la lógica de la API.

- **Desventajas de CSR:**
  - **Tiempo de carga inicial:** El tiempo de carga puede ser mayor ya que el navegador debe descargar y ejecutar el JavaScript antes de que se muestre el contenido.
  - **SEO limitado:** Dado que el contenido se carga dinámicamente, los motores de búsqueda pueden tener dificultades para indexar la página.

SSR (Server-Side Rendering)

Server-Side Rendering (SSR) es una técnica donde las páginas son renderizadas en el servidor antes de ser enviadas al cliente. Next.js facilita el SSR, permitiendo que las aplicaciones carguen rápidamente y sean mejor indexadas por los motores de búsqueda.

- **Ventajas de SSR:**
    - **Mejor SEO:** Dado que el contenido se entrega completamente renderizado, los motores de búsqueda pueden indexar la página fácilmente.
    - **Tiempo de carga inicial más rápido:** El contenido está listo para mostrarse inmediatamente, mejorando la experiencia del usuario.
- **Desventajas de SSR:**
    - **Mayor carga en el servidor:** El servidor debe renderizar cada solicitud, lo que puede incrementar el tiempo de respuesta si no se gestiona adecuadamente.
    - **Menos interactividad inicial:** La interactividad depende de la carga adicional de JavaScript en el cliente.



Creación del Proyecto Next.js

Crear un proyecto con Next.js es sencillo y directo gracias a la herramienta de línea de comandos create-next-app. A continuación, se describe el proceso para crear un nuevo proyecto.

1. **Instalación de Node.js y npm:**
    - Asegúrate de tener **Node.js** y **npm** instalados en tu sistema. Puedes verificar las versiones instaladas ejecutando:

bash

Copy code

node -v

npm -v

2. **Crear un nuevo proyecto Next.js:**
    - Utiliza la herramienta create-next-app para crear un nuevo proyecto con todas las configuraciones iniciales necesarias:

bash

Copy code

npx create-next-app@latest mi-proyecto

Reemplaza mi-proyecto con el nombre que desees para tu proyecto.

Enrutamiento en Next.js

Next.js utiliza un sistema de enrutamiento basado en archivos. Cada archivo dentro de la carpeta pages se convierte automáticamente en una ruta accesible desde la aplicación.

1. **Rutas básicas:**
    - Un archivo llamado index.js dentro de la carpeta pages se mapea a la ruta /.
    - Un archivo llamado about.js dentro de la carpeta pages se mapea a la ruta /about.
2. **Rutas anidadas:**
    - Puedes crear subcarpetas dentro de pages para anidar rutas. Por ejemplo, un archivo blog.js dentro de pages/posts se mapea a la ruta /posts/blog.
3. **Rutas dinámicas:**

- Las rutas dinámicas permiten capturar segmentos de URL como parámetros. Utiliza corchetes para definir rutas dinámicas. Por ejemplo, un archivo [id].js dentro de pages/posts se mapea a la ruta /posts/:id.

jsx

Copy code

```
import { useRouter } from 'next/router';


function Post() {
  const router = useRouter();
  const { id } = router.query;


  return <p>Post ID: {id}</p>;
}


export default Post;
```

En este ejemplo, la página Post captura el parámetro id de la URL, lo que permite acceder al valor dinámico dentro del componente.

4. **Rutas API:**
    - Next.js permite crear rutas API dentro de la carpeta pages/api. Cada archivo se convierte en un endpoint accesible desde el cliente.

jsx

Copy code

```
// pages/api/usuarios.js
export default function handler(req, res) {
  res.status(200).json({ nombre: 'John Doe' });
}
```

En este ejemplo, se crea una ruta API en /api/usuarios que responde con un objeto JSON.

Practica:

Código:

```
// app/layout.tsx
import "./globals.css";

export const metadata = {
  title: "Employee Management",
  description: "A simple employee management system",
};

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="en">
      <body>{children}</body>
    </html>
  );
}

"use client"; // Indica que este es un componente cliente

import React, {
  useState,
  useReducer,
  createContext,
  useContext,
  useEffect,
} from "react";
import EmployeeForm from "../../components/EmployeeForm";
import EmployeeList from "../../components/EmployeeList";
import FilterEmployees from "../../components/FilterEmployees";
import { Employee, Post } from "../types";
import styles from "./page.module.css";

interface EmployeeContextType {
  state: Employee[];
  dispatch: React.Dispatch<any>;
```

```
}

const EmployeeContext = createContext<EmployeeContextType | null>(null);

const initialState: Employee[] = [];

const employeeReducer = (state: Employee[], action: any) => {
  switch (action.type) {
    case "ADD_EMPLOYEE":
      return [...state, action.payload];
    case "EDIT_EMPLOYEE":
      return state.map((employee) =>
        employee.id === action.payload.id ? action.payload : employee
      );
    case "DELETE_EMPLOYEE":
      return state.filter((employee) => employee.id !== action.payload);
    case "SET_EMPLOYEES":
      return action.payload;
    default:
      return state;
  }
};

const Home = () => {
  const [employeeToEdit, setEmployeeToEdit] = useState<Employee |
null>(null);
  const [state, dispatch] = useReducer(employeeReducer, initialState);
  const [filteredEmployees, setFilteredEmployees] =
useState<Employee[]>(state);
  const [posts, setPosts] = useState<Post[]>([]);

  useEffect(() => {
    const fetchPosts = async () => {
      const res = await
fetch("https://jsonplaceholder.typicode.com/posts");
      const data: Post[] = await res.json();
      setPosts(data);
    };

    fetchPosts();

    const storedEmployees = localStorage.getItem("employees");
    if (storedEmployees) {
      dispatch({ type: "SET_EMPLOYEES", payload:
JSON.parse(storedEmployees) });
    }
  }, []);

  useEffect(() => {
```

```
    localStorage.setItem("employees", JSON.stringify(state));
  }, [state]);

  return (
    <EmployeeContext.Provider value={{ state, dispatch }}>
      <div className={styles.container}>
        <h1 className={styles.title}>Employee Management</h1>
        <EmployeeForm
          employeeToEdit={employeeToEdit}
          setEmployeeToEdit={setEmployeeToEdit}
        />
        <FilterEmployees setFilteredEmployees={setFilteredEmployees} />
        <EmployeeList
          employees={filteredEmployees}
          setEmployeeToEdit={setEmployeeToEdit}
        />
        <div className={styles.postsContainer}>
          <h2 className={styles.title}>Posts</h2>
          {posts.map((post) => (
            <div key={post.id} className={styles.post}>
              <h3>{post.title}</h3>
              <p>{post.body}</p>
            </div>
          ))}
        </div>
      </div>
    </EmployeeContext.Provider>
  );
};

export { EmployeeContext };
export type { EmployeeContextType };
export default Home;

'use client';
import React, { useContext, useState, useEffect } from "react";
import { EmployeeContext, EmployeeContextType } from "../src/app/page";
import { Employee } from "../src/types";

interface EmployeeFormProps {
  employeeToEdit: Employee | null;
  setEmployeeToEdit: React.Dispatch<React.SetStateAction<Employee |
null>>;
}

const EmployeeForm = ({
  employeeToEdit,
  setEmployeeToEdit,
}: EmployeeFormProps) => {
```

```
  const { dispatch } = useContext(EmployeeContext) as
EmployeeContextType;
  const [employee, setEmployee] = useState<Employee>({
    id: 0,
    name: "",
    address: "",
    city: "",
  });

  useEffect(() => {
    if (employeeToEdit) {
      setEmployee(employeeToEdit);
    }
  }, [employeeToEdit]);

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (employeeToEdit) {
      dispatch({ type: "EDIT_EMPLOYEE", payload: employee });
      setEmployeeToEdit(null);
    } else {
      dispatch({
        type: "ADD_EMPLOYEE",
        payload: { ...employee, id: Date.now() },
      });
    }
    setEmployee({ id: 0, name: "", address: "", city: "" });
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Name"
        value={employee.name}
        onChange={(e) => setEmployee({ ...employee, name: e.target.value
})}
      />
      <input
        type="text"
        placeholder="Address"
        value={employee.address}
        onChange={(e) => setEmployee({ ...employee, address:
e.target.value })}
      />
      <input
        type="text"
        placeholder="City"
        value={employee.city}
```

```
        onChange={(e) => setEmployee({ ...employee, city: e.target.value
})}
      />
      <button type="submit">
        {employeeToEdit ? "Update" : "Add"} Employee
      </button>
    </form>
  );
};

export default EmployeeForm;

"use client";
import React, { useContext } from "react";
import { EmployeeContext, EmployeeContextType } from "../src/app/page";
import { Employee } from "../src/types";

interface EmployeeListProps {
  employees: Employee[];
  setEmployeeToEdit: React.Dispatch<React.SetStateAction<Employee |
null>>;
}

const EmployeeList = ({ employees, setEmployeeToEdit }:
EmployeeListProps) => {
  const { dispatch } = useContext(EmployeeContext) as
EmployeeContextType;

  return (
    <ul>
      {employees.map((employee) => (
        <li key={employee.id}>
          {employee.name} - {employee.address}, {employee.city}
          <button onClick={() =>
setEmployeeToEdit(employee)}>Edit</button>
          <button
            onClick={() =>
              dispatch({ type: "DELETE_EMPLOYEE", payload: employee.id })
            }
          >
            Delete
          </button>
        </li>
      ))}
    </ul>
  );
};

export default EmployeeList;
```

```
"use client";
import React, { useContext, useState } from "react";
import { EmployeeContext, EmployeeContextType } from "../src/app/page";
import { Employee } from "../src/types";

interface FilterEmployeesProps {
  setFilteredEmployees: React.Dispatch<React.SetStateAction<Employee[]>>;
}

const FilterEmployees = ({ setFilteredEmployees }: FilterEmployeesProps)
=> {
  const { state } = useContext(EmployeeContext) as EmployeeContextType;
  const [query, setQuery] = useState("");

  const handleFilter = () => {
    setFilteredEmployees(
      state.filter((employee) =>
        employee.name.toLowerCase().includes(query.toLowerCase())
      )
    );
  };

  return (
    <div>
      <input
        type="text"
        placeholder="Filter by name"
        value={query}
        onChange={(e) => setQuery(e.target.value)}
      />
      <button onClick={handleFilter}>Filter</button>
    </div>
  );
};

export default FilterEmployees;
```

Vista desde el LocalHost: