CUADERNO PERSONAL

DESARROLLO DE APLICACIONES WEB

SEMANA 10 – Next JS – CSR/SSR

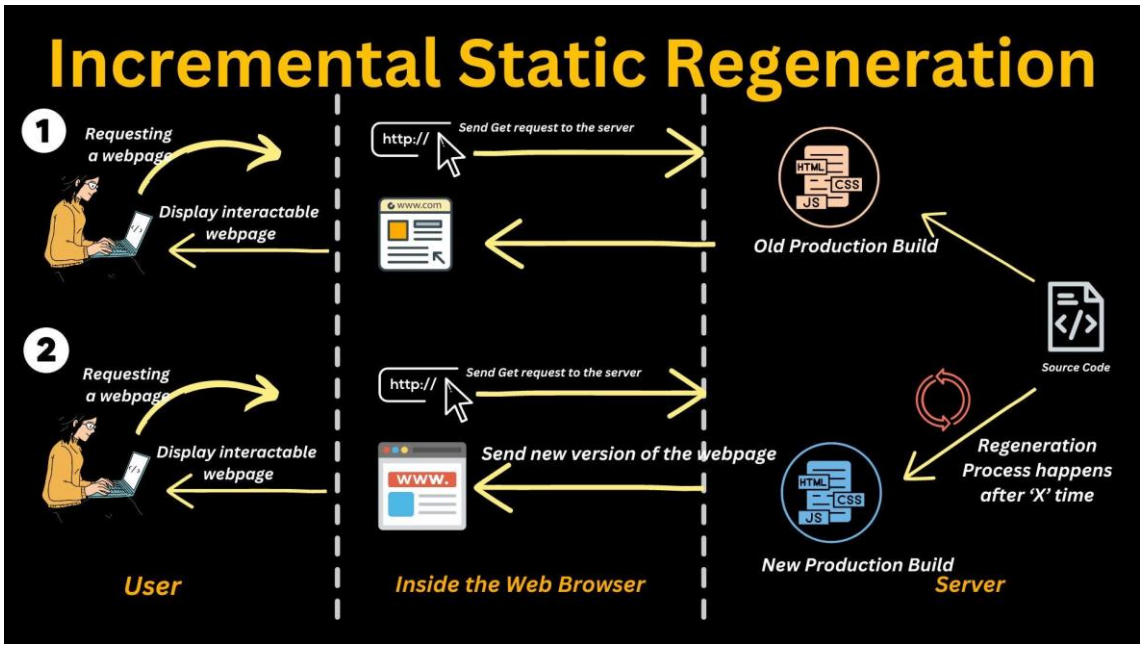
Static Generation (SSG)

Static Generation (SSG) es una técnica de renderizado en la que las páginas son pre-renderizadas a HTML estático durante el proceso de construcción. Esto permite que las páginas se sirvan rápidamente desde un servidor o CDN sin la necesidad de realizar cálculos en cada solicitud. SSG es ideal para páginas que no cambian con frecuencia y necesitan cargar rápidamente.

- **Ventajas de SSG:**
    - **Rendimiento óptimo:** Las páginas estáticas se sirven muy rápidamente desde un servidor o CDN.
    - **SEO amigable:** Las páginas están completamente renderizadas antes de ser enviadas a los usuarios, mejorando la indexación por los motores de búsqueda.
- **Desventajas de SSG:**
    - **Contenido estático:** Las páginas no pueden actualizarse dinámicamente sin una nueva construcción, lo que puede ser un problema para contenido que cambia frecuentemente.
- **Implementación de SSG en Next.js**
- Para utilizar SSG en Next.js, puedes usar la función `getStaticProps`. Esta función se ejecuta en el momento de la construcción y permite obtener datos necesarios para pre-renderizar la página.
- **Ejemplo de SSG:**
- jsx
- Copy code
- // pages/blog.js

```
import fetch from 'node-fetch';

export async function getStaticProps() {
  // Obtener datos desde una API o fuente de datos
  const res = await fetch('https://jsonplaceholder.typicode.com/posts');
  const posts = await res.json();

  // Retornar los datos como props
  return {
    props: {
      posts,
    },
  };
}

function Blog({ posts }) {
  return (
    <div>
      <h1>Blog</h1>
      <ul>
        {posts.map((post) => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </div>
  );
}

export default Blog;
```

- En este ejemplo, la función getStaticProps obtiene una lista de publicaciones desde una API externa y las pasa al componente Blog como propiedades.



Incremental Static Regeneration (ISR)

Incremental Static Regeneration (ISR) es una característica avanzada de Next.js que permite actualizar páginas estáticas después de la construcción inicial, sin necesidad de reconstruir todo el sitio. ISR es útil para páginas que requieren actualizaciones periódicas, pero que aún se benefician de las ventajas de SSG.

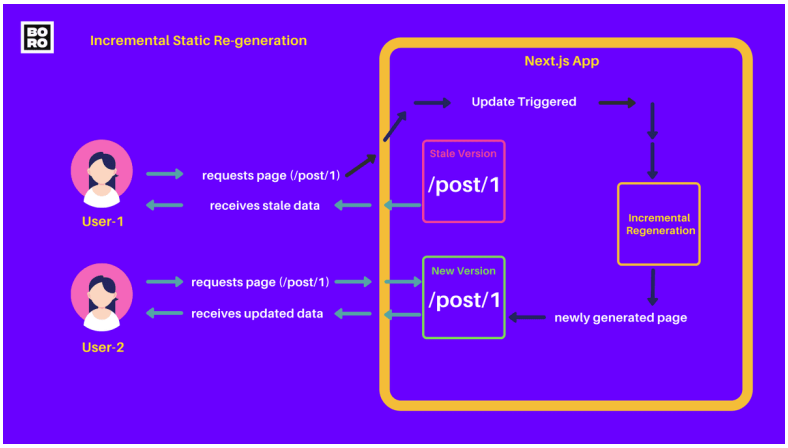
- **Ventajas de ISR:**
    - **Actualización automática:** Permite que las páginas se actualicen en el servidor a intervalos definidos sin necesidad de una reconstrucción completa.
    - **Combina SSG y SSR:** Ofrece la velocidad del SSG con la capacidad de actualización de SSR.
- **Desventajas de ISR:**
    - **Complejidad:** Puede ser más complejo de configurar que SSG, especialmente para páginas que requieren múltiples actualizaciones.

Incremental Static Generation (ISG)

Incremental Static Generation (ISG) es un concepto que a menudo se confunde con ISR. Sin embargo, ISG generalmente se refiere a la capacidad de generar páginas de manera incremental en tiempo de ejecución cuando

son solicitadas por primera vez. Esto permite que las aplicaciones manejen rutas desconocidas o dinámicas con una pre-renderización estática.

- **Ventajas de ISG:**
    - **Escalabilidad:** Maneja rutas dinámicas de manera eficiente sin requerir una reconstrucción completa.
    - **Optimización del rendimiento:** Reduce la carg a inicial al generar páginas bajo demanda.



- **Desventajas de ISG:**
    - **Almacenamiento en caché:** Requiere una buena estrategia de almacenamiento en caché para manejar páginas generadas bajo demanda.

Creación de un Proyecto en Next.js

La creación de un proyecto en Next.js es un proceso sencillo que se puede realizar en pocos pasos. A continuación, se describen las etapas principales:

1. **Instalar Node.js y npm:**
    - Asegúrate de tener **Node.js** y **npm** instalados en tu sistema. Verifica las versiones con:

bash

Copy code

node -v

npm -v

2. **Crear un nuevo proyecto:**

- Utiliza create-next-app para inicializar un nuevo proyecto Next.js con configuración predeterminada:

bash

Copy code

npx create-next-app mi-proyecto

3. **Estructura del proyecto:**
    - La estructura generada incluye carpetas y archivos principales:

java

Copy code

mi-proyecto/

├── pages/

│ ├── api/

│ │ └── hello.js

│ ├── _app.js

│ ├── index.js

├── public/

├── styles/

├── .gitignore

├── package.json

├── README.md

4. **Iniciar el servidor de desarrollo:**
    - Navega al directorio del proyecto y ejecuta:

bash

Copy code

cd mi-proyecto

npm run dev

- Accede al proyecto en http://localhost:3000.

Practica:

Código:

```
// ColorContext.js
import React, { createContext, useState } from "react";

export const ColorContext = createContext();

export const ColorProvider = ({ children }) => {
  const [color1, setColor1] = useState("blue");
  const [color2, setColor2] = useState("green");
  const [color3, setColor3] = useState("lightblue");

  const getRandomColor = () => {
    const letters = "0123456789ABCDEF";
    let color = "#";
    for (let i = 0; i < 6; i++) {
      color += letters[Math.floor(Math.random() * 16)];
    }
    return color;
  };

  const changeColors = () => {
    setColor1(getRandomColor());
    setColor2(getRandomColor());
    setColor3(getRandomColor());
  };

  return (
    <ColorContext.Provider value={{ color1, color2, color3, changeColors
}}>
      {children}
    </ColorContext.Provider>
  );
};



// Componente1.js
import React, { useContext } from "react";
import { ColorContext } from "./ColorContext";

function Componente1() {
  const { color1, changeColors } = useContext(ColorContext);
  return (
```

```
    <div className="componente1" style={{ backgroundColor: color1 }}>
      <button onClick={changeColors}>Cambiar color</button>
    </div>
  );
}

export default Componente1;

// Componente2.js
import React, { useContext } from "react";
import { ColorContext } from "./ColorContext";

function Componente2() {
  const { color2, changeColors } = useContext(ColorContext);
  return (
    <div className="componente1" style={{ backgroundColor: color2 }}>
      <button onClick={changeColors}>Cambiar color</button>
    </div>
  );
}

export default Componente2;

// Componente3.js
import React, { useContext } from "react";
import { ColorContext } from "./ColorContext";

function Componente3() {
  const { color3, changeColors } = useContext(ColorContext);
  return (
    <div className="componente3" style={{ backgroundColor: color3 }}>
      <img src="/myImage.png" alt="My photo" />
      <button onClick={changeColors}>Cambiar color</button>
    </div>
  );
}

export default Componente3;

import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root")
```

```
);

import React, { useState } from "react";
import Componente1 from "./Componente1";
import Componente2 from "./Componente2";
import Componente3 from "./Componente3";

const getRandomColor = () => {
  const letters = "0123456789ABCDEF";
  let color = "#";
  for (let i = 0; i < 6; i++) {
    color += letters[Math.floor(Math.random() * 16)];
  }
  return color;
};

const ParentComponent = () => {
  const [backgroundColor, setBackgroundColor] = useState("#ffffff");

  const handleChangeColor = () => {
    setBackgroundColor(getRandomColor());
  };

  return (
    <div>
      <button onClick={handleChangeColor}>Cambiar Color de Todos</button>
      <Componente1 initialColor={backgroundColor} />
      <Componente2 initialColor={backgroundColor} />
      <Componente3 initialColor={backgroundColor} />
    </div>
  );
};

export default ParentComponent;

import { useState, useEffect } from "react";

const images = [
  "/image1.jpg",
  "/image2.jpg",
  "/image3.jpg",
  "/image4.jpg",
  "/image5.jpg",
];

const getRandomImage = () => {
  const randomIndex = Math.floor(Math.random() * images.length);
  return images[randomIndex];
};
```

```
const useRandomImage = () => {
  const [image, setImage] = useState(getRandomImage());

  useEffect(() => {
    const interval = setInterval(() => {
      setImage(getRandomImage());
    }, 1000); // 30 segundos

    return () => clearInterval(interval); // Limpiar intervalo al
desmontar el componente
  }, []);

  return image;
};

export default useRandomImage;

import React from "react";
import useRandomImage from "./useRandomImage";
import "./ImageWidget.css";

const ImageWidget = () => {
  const image = useRandomImage();

  return (
    <div className="image-widget">
      <img src={image} alt="Random" />
    </div>
  );
};

export default ImageWidget;

import React from "react";

const EmployeeCard = ({ employee, onClose }) => {
  return (
    <div className="employee-card">
      <h2>{employee.name}</h2>
      <p>
        <strong>Dirección:</strong> {employee.address}
      </p>
      <p>
        <strong>Ciudad:</strong> {employee.city}
      </p>
      <button onClick={onClose}>Cerrar</button>
    </div>
  );
```

```
};

export default EmployeeCard;

import React, { useState, useContext, useEffect } from "react";
import { EmployeeContext } from "../context/EmployeeContext";

const EmployeeForm = ({ editableEmployee, setEditableEmployee }) => {
  const { dispatch } = useContext(EmployeeContext);
  const [employee, setEmployee] = useState({
    id: "",
    name: "",
    address: "",
    city: "",
  });

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

  const handleChange = (e) => {
    setEmployee({ ...employee, [e.target.name]: e.target.value });
  };

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

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        name="name"
        placeholder="Nombre"
        value={employee.name}
        onChange={handleChange}
        required
```

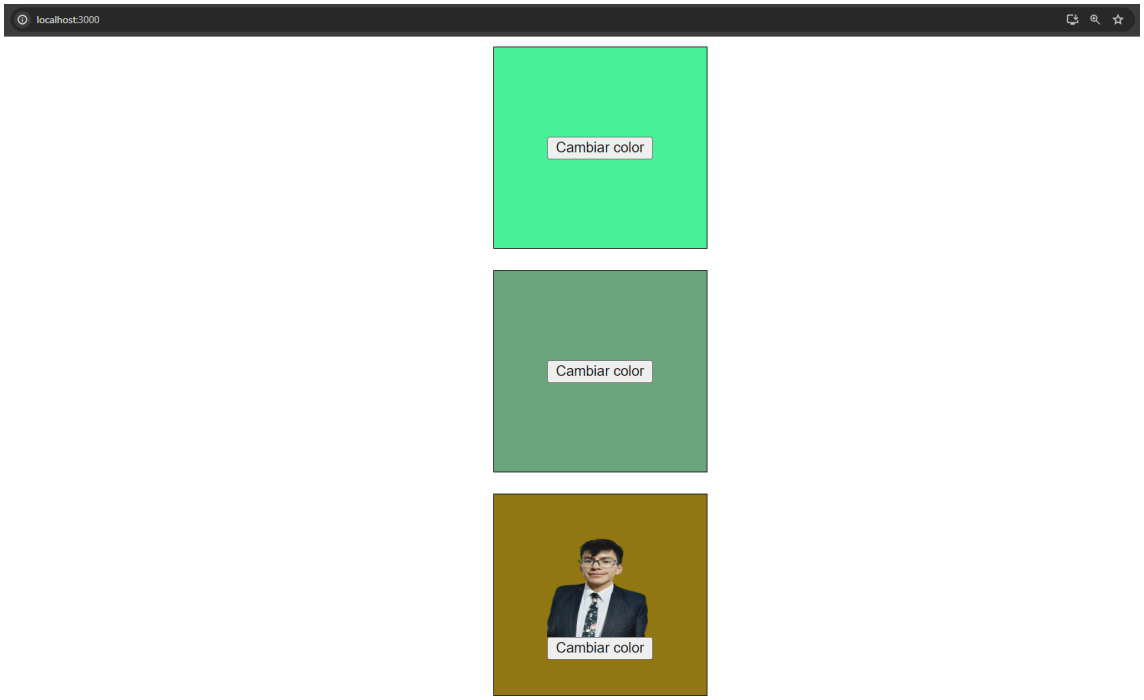
```
      />
      <input
        type="text"
        name="address"
        placeholder="Dirección"
        value={employee.address}
        onChange={handleChange}
        required
      />
      <input
        type="text"
        name="city"
        placeholder="Ciudad"
        value={employee.city}
        onChange={handleChange}
        required
      />
      <button type="submit">
        {employee.id ? "Editar" : "Agregar"} Empleado
      </button>
    </form>
  );
};

export default EmployeeForm;

import React, { useContext } from "react";
import { EmployeeContext } from "../context/EmployeeContext";

const EmployeeItem = ({
  employee,
  setEditableEmployee,
  setDetailedEmployee,
}) => {
  const { dispatch } = useContext(EmployeeContext);

  const handleDelete = () => {
    dispatch({ type: "DELETE_EMPLOYEE", payload: employee.id });
  };

  const handleEdit = () => {
    setEditableEmployee(employee);
  };

  const handleShowDetails = () => {
    setDetailedEmployee(employee);
  };

  return (
```

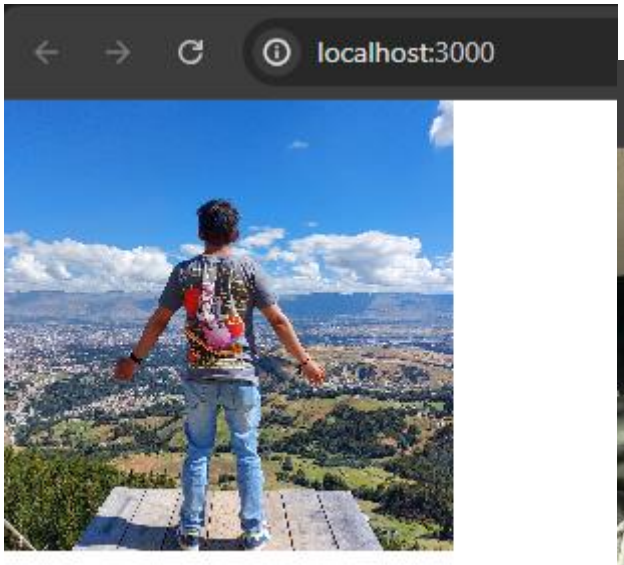
```
    <li>
      {employee.name} - {employee.address} - {employee.city}
      <button onClick={handleEdit}>Editar</button>
      <button onClick={handleDelete}>Eliminar</button>
      <button onClick={handleShowDetails}>Mostrar Detalles</button>
    </li>
  );
};

export default EmployeeItem;

import React, { useContext, useState } from "react";
import { EmployeeContext } from "../context/EmployeeContext";
import EmployeeItem from "./EmployeeItem";
import EmployeeForm from "./EmployeeForm";
import EmployeeCard from "./EmployeeCard";

const EmployeeList = () => {
  const { state } = useContext(EmployeeContext);
  const [editableEmployee, setEditableEmployee] = useState(null);
  const [detailedEmployee, setDetailedEmployee] = useState(null);
  const [filter, setFilter] = useState("");
  const [currentPage, setCurrentPage] = useState(1);
  const employeesPerPage = 5;

  const handleFilterChange = (e) => {
    setFilter(e.target.value);
    setCurrentPage(1); // Reset to first page when filter changes
  };

  const handleFilter = () => {
    return state.employees.filter(
      (employee) =>
        employee.name.toLowerCase().includes(filter.toLowerCase()) ||
        employee.address.toLowerCase().includes(filter.toLowerCase()) ||
        employee.city.toLowerCase().includes(filter.toLowerCase())
    );
  };

  const filteredEmployees = handleFilter();

  // Logic for displaying employees
  const indexOfLastEmployee = currentPage * employeesPerPage;
  const indexOfFirstEmployee = indexOfLastEmployee - employeesPerPage;
  const currentEmployees = filteredEmployees.slice(
    indexOfFirstEmployee,
    indexOfLastEmployee
  );
```

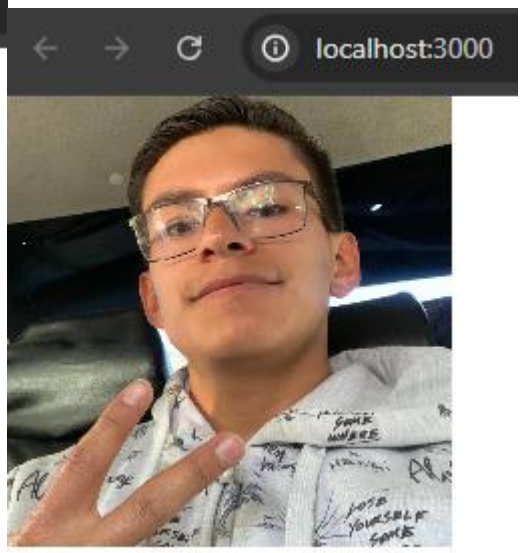
```
const paginate = (pageNumber) => setCurrentPage(pageNumber);

const pageNumbers = [];
for (
  let i = 1;
  i <= Math.ceil(filteredEmployees.length / employeesPerPage);
  i++
) {
  pageNumbers.push(i);
}

return (
  <>
    <EmployeeForm
      editableEmployee={editableEmployee}
      setEditableEmployee={setEditableEmployee}
    />
    <div className="filter-container">
      <input
        type="text"
        placeholder="Filtrar empleados"
        value={filter}
        onChange={handleFilterChange}
      />
    </div>
    <ul>
      {currentEmployees.map((employee) => (
        <EmployeeItem
          key={employee.id}
          employee={employee}
          setEditableEmployee={setEditableEmployee}
          setDetailedEmployee={setDetailedEmployee}
        />
      ))}
    </ul>
    {detailedEmployee && (
      <EmployeeCard
        employee={detailedEmployee}
        onClose={() => setDetailedEmployee(null)}
      />
    )}
    <div className="pagination">
      {pageNumbers.map((number) => (
        <button key={number} onClick={() => paginate(number)}>
          {number}
        </button>
      ))}
    </div>
  </>
```

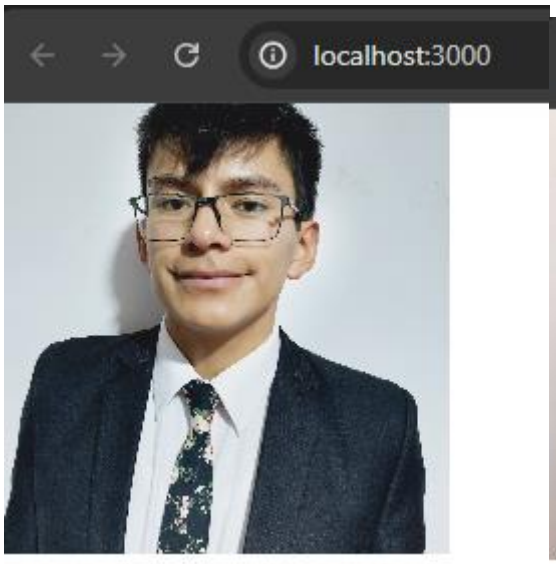
```
  );
};

export default EmployeeList;

import React from "react";
import EmployeeProvider from "./context/EmployeeContext";
import EmployeeList from "./components/EmployeeList";

const App = () => {
  return (
    <EmployeeProvider>
      <div className="container">
        <h1>Administrar Empleados</h1>
        <EmployeeList />
      </div>
    </EmployeeProvider>
  );
};

export default App;
```

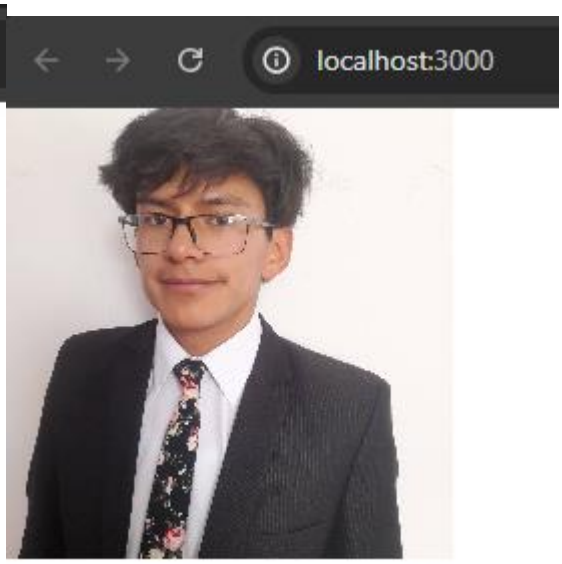
Vista desde el LocalHost:

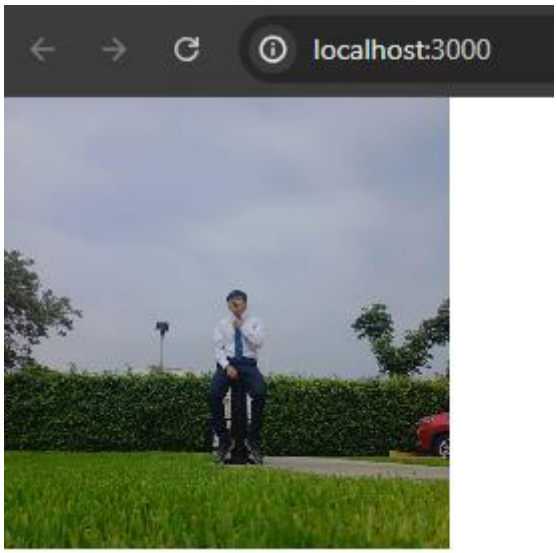







localhost:3000


localhost:3000


localhost:3000


localhost:5173
Administrar Empleados
Nombre
Dirección
Ciudad
Agregar Empleado
Filtrar empleados
Cristian Torres Cordova - Pasaje. Rio Chilca #304 - Chilca
Editar
Eliminar
Mostrar Detalles
Cristian Torres Cordova - Pasaje. Rio Chilca #304 - Chilca
Editar
Eliminar
Mostrar Detalles
Cristian Torres Cordova - Pasaje. Rio Chilca #304 - Chilca
Editar
Eliminar
Mostrar Detalles
Cristian Torres Cordova - Pasaje. Rio Chilca #304 - Chilca
Editar
Eliminar
Mostrar Detalles
Cristian Torres Cordova - Pasaje. Rio Chilca #304 - Chilca
Editar
Eliminar
Mostrar Detalles
1
2