«Talento Tech»

Desarrollo Web 3

Clase 09

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 9 | Next.js

Temario:

- Que es Next.js y cómo instalarlo
- Introducción a Server Side Rendering como Next js
- Introducción a Static Site Generation
- SSR vs SSG
- Componente Image
- Pages

Buenos Aires aprende
Agencia de Habilidades para el Futuro

¿Qué es Next.js?

¿Qué es Next.js?

Next.js es un framework de React que se ha vuelto esencial en el desarrollo web. Su objetivo es hacer que el desarrollo con React sea más accesible y eficiente. Una de las ventajas clave es su integración con Server Side Rendering (SSR) y Static Site Generation (SSG), técnicas que mejoran la velocidad de carga y la optimización de las páginas web.

Ventajas

- Simplifica el desarrollo React
- Mejora el rendimiento mediante técnicas avanzadas de renderizado
- Ofrece enrutamiento simple con la carpeta "pages"
- Compatible con la comunidad y herramientas de React

Instalación y configuración de Next.js

Para comenzar un proyecto con Next.js, utilizamos el comando **npx create-next-app** seguido del **nombre del proyecto**. Este comando crea una estructura de proyecto inicial y maneja las dependencias necesarias. Al explorar la estructura del proyecto, encontramos la carpeta "pages" que juega un papel crucial en la definición de las rutas de la aplicación.

Paso a paso

Lo primero que debemos hacer para instalar Next.js es abrir la terminal o CLI de nuestra preferencia y escribir el siguiente comando:

```
npx create-next-app@latest
```

A continuación en la consola de comando veremos las siguientes preguntas:

```
What is your project named? my-app
Would you like to use TypeScript? No / Yes
Would you like to use ESLint? No / Yes
Would you like to use Tailwind CSS? No / Yes
Would you like to use `src/` directory? No / Yes
Would you like to use App Router? (recommended) No / Yes
Would you like to customize the default import alias (@/*)? No / Yes
What import alias would you like configured? @/*
```

Estas preguntas en general, se responden de acuerdo a las necesidades del proyecto o en nuestro caso a la de la clase. Igualmente les dejo una recomendación para la clase:

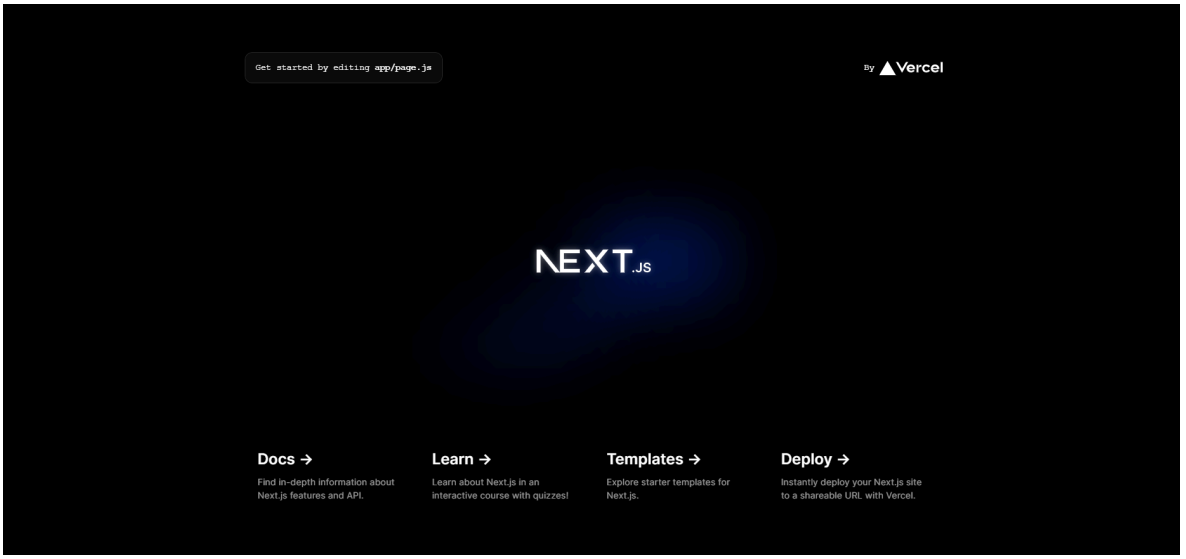
- **What is your project name?** next-ap //Puede ser cualquiera, sean creativos
- **Would you like to use TypeScript?** No
- **Would you like to use ESLint?** Yes
- **Would you like to use Tailwind CSS?** No
- **Would you like to use `src/` directory?** No
- **Would you like to customize the default import alias (@/*)?** No

Una vez respondidas las preguntas, empezará a hacer el build de nuestra aplicación. Finalizado esto, recuerden usar el comando **cd nombre_del_proyecto** y luego, dentro, ejecutar el siguiente comando:

```
npm run dev
```

Si todo salió bien, debería haber levantado la aplicación en http://localhost:3000/ y

aparecerá la siguiente pantalla:



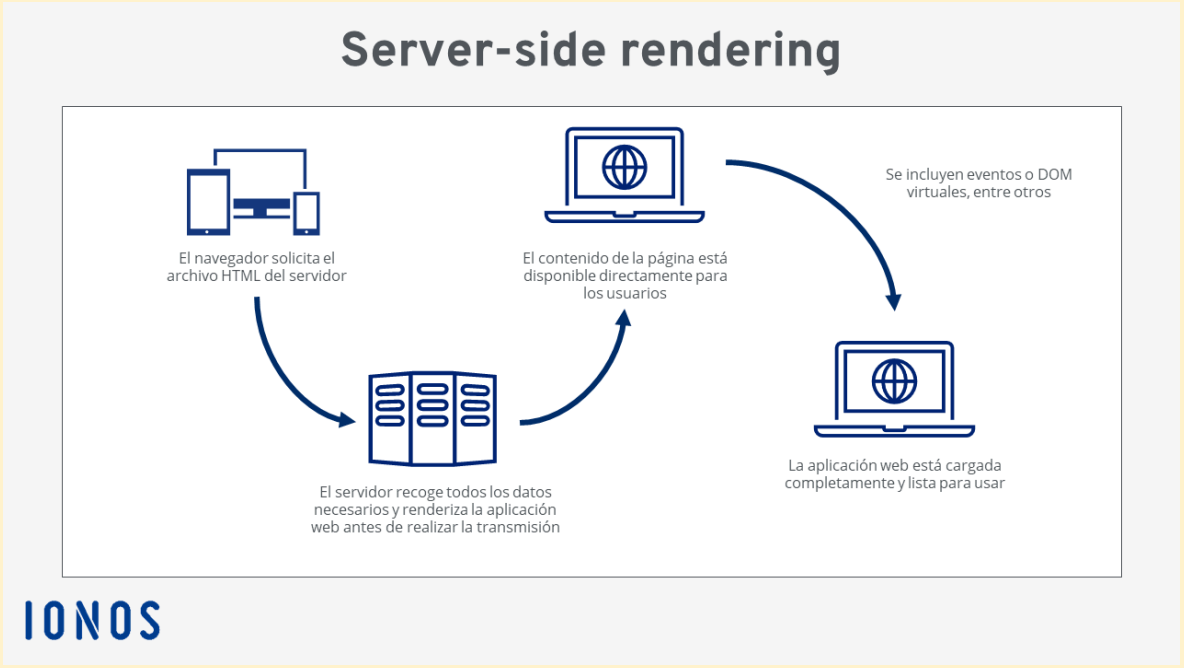
Si tenemos esta pantalla en nuestro localhost, felicidades ya tenemos una aplicación de Next.js. En el próximo tema veremos las bondades que nos trae este framework.

Server Side Rendering (SSR)

¿Qué es Server Side Rendering?
En un enfoque de Server Side Rendering, la generación y renderizado de la página web ocurre en el servidor antes de que se envíe al navegador del usuario. En lugar de depender únicamente del cliente (navegador) para renderizar la página, el servidor realiza esta tarea inicialmente.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Proceso detallado



Implementación

La implementación de SSR se simplifica mediante funciones especiales como **getServerSideProps**. Esta función permite ejecutar código en el servidor para obtener datos y pasarlos como propiedades a nuestros componentes antes de que la página se envíe al navegador.

En este ejemplo, la función **getServerSideProps** se ejecuta en el servidor al cargar la página "about.js". Puede realizar solicitudes a bases de datos, servicios externos, o cualquier operación que sea necesaria para obtener los datos necesarios. Luego, estos datos se pasan como propiedades al componente “AboutPage”.

```
// Archivo: pages/about.js
const AboutPage = ({ data }) => {
```

```
    return (
        <div>
            <p>{data.description}</p>
        </div>
    );
};
export async function getServerSideProps() {
    // Lógica para obtener datos desde el servidor
    const data = fetch('...')
    return {
        props: {
            data,
        },
    };
}
```

Ventajas

- **Mejora el tiempo de carga inicial:** El usuario recibe una página completamente renderizada desde el principio, mejorando los tiempos de carga inicial.
- **Optimización para motores de búsqueda:** Al generar contenido en el servidor, las páginas son más fácilmente indexables por los motores de búsqueda, mejorando el SEO.
- **Compatibilidad con Redes Sociales:** Mejora la representación de las páginas compartidas en redes sociales al proporcionar contenido renderizado.

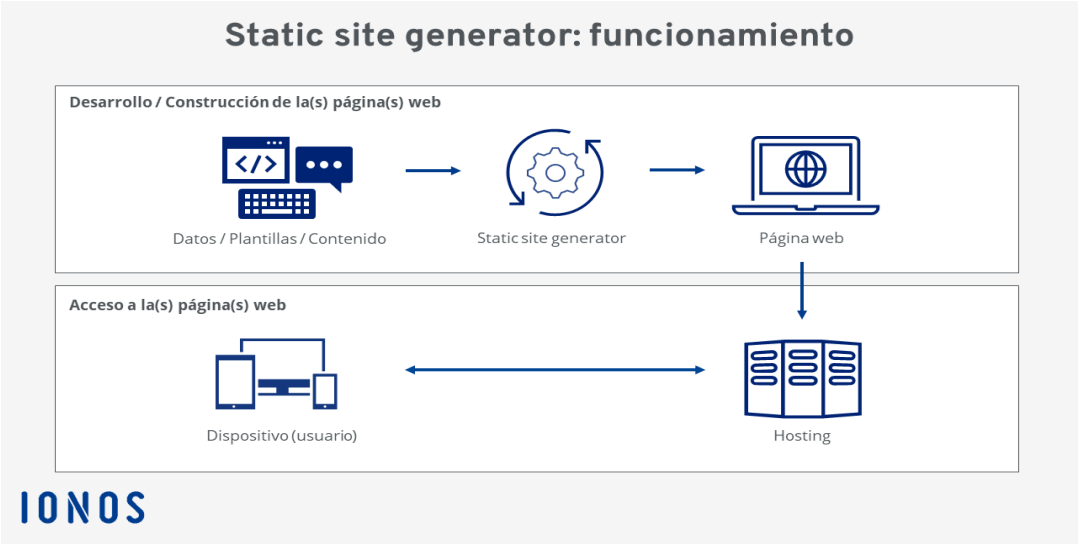
Desventajas

- **Mayor carga en el servidor:** El servidor realiza más trabajo al generar cada página, lo que puede aumentar la carga en entornos de alta concurrencia.
- **Menor interactividad inicial:** La interactividad completa de la aplicación no está disponible hasta después de que la página se carga inicialmente.

Static Site Generation (SSG)

¿Qué es Static Site Generation (SSG)?
Static Site Generation implica la creación y generación de páginas web estáticas durante el proceso de compilación, en lugar de hacerlo en cada solicitud del usuario. En lugar de depender de un servidor para construir la página dinámicamente en tiempo real, SSG permite precompilar las páginas antes de que la aplicación se despliegue.

Proceso detallado



Implementación

Next.js simplifica la implementación de SSG mediante funciones como **getStaticProps**. Esta función se utiliza para obtener datos durante la fase de compilación y pasarlos como propiedades a nuestros componentes antes de que se generen las páginas estáticas.

En este ejemplo, la función **getStaticProps** se ejecuta durante la fase de compilación. Puede realizar solicitudes a bases de datos, servicios externos, o cualquier operación

necesaria para obtener los datos estáticos que se incorporarán en la página. Luego, estos datos se pasan como propiedades al componente “BlogPost”.

```
// Archivo: pages/blog/[slug].js
const BlogPost = ({ post }) => {
  return (
    <div>
      <h1>{post.title}</h1>
      <p>{post.content}</p>
    </div>
  );
};

export async function getStaticProps(context) {
  // Lógica para obtener datos estáticos
  const post = fetch('...')
  return {
    props: {
      post,
    },
  };
}
```

Ventajas

- **Rendimiento Mejorado:** Al servir páginas estáticas, el tiempo de carga es significativamente más rápido ya que no hay necesidad de construir la página en cada solicitud.
- **Escalabilidad Simplificada:** Las páginas estáticas son más fáciles de escalar, ya que no requieren recursos en tiempo real para cada solicitud.
- **Eficiencia en Recursos**: Con SSG, los recursos del servidor se utilizan de manera más eficiente al generar páginas durante la compilación.

Desventajas

- **Datos Dinámicos Limitados:** SSG es ideal para contenido estático o semi-estático. Para datos altamente dinámicos, se pueden necesitar enfoques alternativos.
- **Compilación Más Lenta:** En aplicaciones con una gran cantidad de páginas, la fase de compilación puede ser más lenta.

SSR vs SSG

¿Cuándo Utilizar Cada Técnica?
SSR: Para aplicaciones con contenido dinámico que cambia frecuentemente.
Cuando se necesita interactividad completa en el lado del cliente después de la carga inicial.
SSG: Para contenido más estático o semi-estático que no cambia con frecuencia.
En situaciones donde se prioriza un tiempo de carga rápido y eficiencia de recursos.

Conclusiones

Ambas técnicas tienen sus fortalezas y deben seleccionarse según los requisitos específicos de la aplicación.
SSR es ideal para aplicaciones dinámicas, mientras que SSG brinda un rendimiento eficiente para contenido más estático.
En muchos casos, una combinación de ambas técnicas (híbridas) puede ofrecer un equilibrio óptimo entre rendimiento y flexibilidad.
Comprender las diferencias y aplicaciones específicas de SSR y SSG permite a los desarrolladores tomar decisiones informadas según las necesidades particulares de cada proyecto.

Componente Image

¿Qué es el componente Image?
El componente Image es una herramienta poderosa de Next.js que simplifica y optimiza la carga de imágenes en una aplicación web. Este componente está diseñado para abordar

problemas comunes relacionados con el rendimiento de las imágenes, como la carga lenta y la gestión ineficiente de recursos.

Características Clave del Componente Image

- **Optimización de Tamaño:** El componente Image realiza automáticamente la optimización de tamaño, generando versiones de imágenes adaptadas a diferentes dispositivos y resoluciones.
- **Carga Asíncrona:** Implementa la carga asíncrona, lo que significa que las imágenes se cargan de manera eficiente sin bloquear el rendimiento de la página.
- **Manejo de Formatos Modernos:** Compatible con formatos de imágenes modernos como WebP, proporcionando una calidad óptima con tamaños de archivo reducidos.
- **Lazy Loading Integrado:** Incorpora la carga perezosa (lazy loading), retrasando la carga de imágenes fuera del campo de visión del usuario hasta que se necesiten.
- **Generación de Imágenes Optimizadas:** Durante la compilación, Next.js genera versiones optimizadas de las imágenes, reduciendo la necesidad de procesamiento en tiempo real.

Uso del Componente Image

Importación: Importa el componente Image desde la biblioteca de Next.js.

```
import Image from 'next/image';
```

Implementación: Utiliza el componente Image en lugar de la etiqueta img estándar.

```
const MyImageComponent = () => {
  return (
    <Image
      src="/path/to/image.jpg"
      alt="Descripción de la imagen"
      width={500}
      height={300}
    />
  );
};
```

Propiedades Adicionales: Personaliza la carga de imágenes utilizando propiedades adicionales como layout, objectFit y más, según las necesidades específicas.

```
<Image
   src="/path/to/image.jpg"
   alt="Descripción de la imagen"
   width={500}
   height={300}
   layout="responsive"
   objectFit="cover"
/>
```

Ventajas del Componente Image

- Mejora el Rendimiento
- Acelera la carga de la página al optimizar la entrega de imágenes
- Ahorra Ancho de Banda. Reduce el uso de ancho de banda gracias a la carga perezosa y la generación de versiones optimizadas
- Cumple con Buenas Prácticas. Implementa automáticamente mejores prácticas para la gestión de imágenes, como lazy loading y generación de versiones optimizadas.

Consideraciones Importantes

➔ **Ubicación de las Imágenes:** Asegúrate de que las imágenes estén ubicadas en un directorio accesible por Next.js para que pueda optimizarlas durante la compilación.
➔ **Proporciones de Imagen Fijas:** Si conoces las dimensiones exactas de las imágenes, proporcionarlas como propiedades width y height mejora aún más la eficiencia.

Conclusión

El componente Image de Next.js es una herramienta esencial para mejorar el rendimiento y la eficiencia en la carga de imágenes en aplicaciones web. Al implementar este componente, los desarrolladores pueden proporcionar una experiencia de usuario más rápida y eficiente, especialmente en entornos donde la gestión de imágenes es crítica para el rendimiento general de la aplicación (ejemplo: Instragram).

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Pages

¿Qué son las "Pages" en Next.js?
En Next.js, cada archivo JavaScript dentro de la carpeta "pages" representa una página en la aplicación. La estructura de carpetas y archivos dentro de "pages" define automáticamente las rutas de la aplicación. Este enfoque simplifica el enrutamiento y proporciona una manera intuitiva de organizar componentes en función de la navegación de la aplicación.

Creación de Páginas
Cada archivo en la carpeta "pages" representa una página. Por ejemplo, index.js dentro de "pages" define la página principal.

```
/pages
  /index.js    // Página principal
  /about.js    // Página "Acerca de"
  /contact.js  // Página "Contacto"
```

Definición de Rutas
La estructura de carpetas y archivos en "pages" determina automáticamente las rutas de la aplicación. Por ejemplo, about.js se asocia a la ruta "/about".

Componentes React
Cada archivo en "pages" es un componente React que define el contenido y la estructura de la página asociada.

```
// Ejemplo: /pages/about.js
const AboutPage = () => {
  return (
    <div>
      <h1>Acerca de Nosotros</h1>
```

```
        <p>¡Bienvenido a nuestra página de acerca de nosotros!</p>
      </div>
    );
};
export default AboutPage;
```

Enrutamiento Dinámico
Next.js permite el enrutamiento dinámico mediante carpetas y archivos dentro de "pages". Por ejemplo, [slug].js podría ser una página dinámica que acepta un parámetro "slug".

```
// Ejemplo: /pages/blog/[slug].js
const BlogPost = ({ slug }) => {
  return (
    <div>
      <h1>{slug}</h1>
      {/* Contenido del blog */}
    </div>
  );
};
export default BlogPost;
```

Ventajas de Organizar Páginas en "Pages"

- **Estructura Clara:** La estructura de carpetas y archivos refleja claramente la estructura de navegación de la aplicación.
- **Fácil Mantenimiento:** Es fácil añadir, modificar o eliminar páginas al trabajar directamente en la carpeta "pages".
- **Enrutamiento Intuitivo:** El enrutamiento se gestiona automáticamente según la ubicación de los archivos en "pages".
- **Escalabilidad:** Manejar varias páginas es sencillo y escalable, lo que facilita el crecimiento de la aplicación.

Conclusión

La organización de páginas en la carpeta "pages" es una característica distintiva de Next.js que simplifica significativamente el desarrollo web. Al utilizar esta estructura intuitiva, los desarrolladores pueden crear y mantener aplicaciones de manera eficiente, aprovechando la capacidad de Next.js para gestionar automáticamente el enrutamiento basado en la organización de archivos.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Desafío #3 | Parte 2

Creación de Aplicación de Gestión de Usuarios

Segunda Parte: Navegación y Actualización de Usuarios

En esta fase, nos vamos a enfocar en la navegación entre páginas y la actualización de información de usuarios.

1. **Navegación con React Router DOM 6:**
Configura un enrutador principal que gestione las rutas de la aplicación. Esto será en el mismo componente App.jsx que teníamos antes, vamos a sobre-escribirlo. Ahora su propósito cambia a ser un gestor de rutas.

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import UserList from './UserList';
import UserDetails from './UserDetails';
import UserForm from './UserForm';
import UserEdit from './UserEdit';
import UserDelete from './UserDelete';
const App = () => {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<UserList />} />
        <Route path="/users/:id" element={<UserDetails />} />
        <Route path="/create" element={<UserForm />} />
        <Route path="/edit/:id" element={<UserEdit />} />
        <Route path="/delete/:id" element={<UserDelete />} />
      </Routes>
    </Router>
  );
};
export default App;
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from
'react-router-dom';
import UserList from './UserList';
import UserDetails from './UserDetails';
import UserForm from './UserForm';
import UserEdit from './UserEdit';
import UserDelete from './UserDelete';

const App = () => {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<UserList />} />
        <Route path="/users/:id" element={<UserDetails />} />
        <Route path="/create" element={<UserForm />} />
        <Route path="/edit/:id" element={<UserEdit />} />
        <Route path="/delete/:id" element={<UserDelete />} />
      </Routes>
    </Router>
  );
};

export default App;
```

2. **Detalles de Usuario:**
Implementa una página de detalles de usuario que muestra información detallada sobre un usuario específico.Crea un componente UserDetails.jsx para mostrar detalles de usuario.

```
import React, { useEffect, useState } from 'react';
import { useParams } from 'react-router-dom';
const UserDetails = () => {
```

```
  const [user, setUser] = useState({});
  const { id } = useParams();
  useEffect(() => {
    // Llamar a la función para obtener los detalles del usuario
    fetchUserDetails();
  }, [id]);
  const fetchUserDetails = async () => {
    try {
      const response = await fetch(`URL_DE_TU_API/${id}`);
      const data = await response.json();
      setUser(data);
    } catch (error) {
      console.error('Error en la solicitud: ', error);
    }
  };

  return (
    <div>
      <h1>Detalles de Usuario</h1>
      <p>ID: {user.id}</p>
      <p>Nombre: {user.name}</p>
      <p>Email: {user.email}</p>
       <Link to={`/delete/${user.id}`}>Eliminar Usuario</Link>
      <Link to={`/edit/${user.id}`}>Editar Usuario</Link>
      <Link to={`/`}>Volver</Link>
    </div>
  );
};
export default UserDetails;
```

3. **Actualizar Usuarios:**
Permite la edición de la información de un usuario existente.Crea un componente

Buenos Aires aprende
Agencia de Habilidades para el Futuro

UserEdit.jsx para el formulario de edición.

```
import React, { useEffect, useState } from 'react';
import { useParams, useNavigate } from 'react-router-dom';
const UserEdit = () => {
  const [user, setUser] = useState({});
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const { id } = useParams();
  const navigate = useNavigate();
  useEffect(() => {
    // Llamar a la función para obtener los detalles del usuario
    fetchUserDetails();
  }, [id]);

  const fetchUserDetails = async () => {
    try {
      const response = await fetch(`URL_DE_TU_API/${id}`);
      const data = await response.json();
      setUser(data);
      setName(data.name);
      setEmail(data.email);
    } catch (error) {
      console.error('Error en la solicitud: ', error);
    }
  };
  const handleUpdate = async () => {
    try {
      const response = await fetch(`URL_DE_TU_API/${id}`, {
        method: 'PUT',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ name, email }),
```

```
    });
    if (response.ok) {
      navigate(`/users/${id}`);
    } else {
      console.error('Error al actualizar usuario');
    }
  } catch (error) {
    console.error('Error en la solicitud: ', error);
  }
};
return (
  <div>
    <h1>Editar Usuario</h1>
    <label>Nombre: </label>
    <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
    <br />
    <label>Email: </label>
    <input type="text" value={email} onChange={(e) => setEmail(e.target.value)} />
    <br />
    <button onClick={handleUpdate}>Actualizar</button>
  </div>
);
};
export default UserEdit;
```

Con estos pasos, deberías tener una aplicación que implementa la navegación con React Router DOM 6 y permite ver detalles y actualizar información de usuarios. La próxima parte se centrará en la eliminación de usuarios y la implementación de la Context API.


Buenos Aires
aprende
Buenos
Aires
Ciudad