

**FUNDACION UNIVERSITARIA INTERNACIONAL  
DE LA RIOJA COLOMBIA**

**ESPECIALIZACION EN INGENIERIA DE SOFTWARE  
(DESARROLLO DE APLICACIONES WEB)**

**ACTIVIDAD 2  
DESARROLLO DE UN FRONT-END UTILIZANDO REACT**

**HAROLD FERNANDO ROJAS LOPEZ**

**12 ENERO 2025**

<b>INTRODUCCIÓN</b>	<b>3</b>
<b>DESARROLLO DE LA ACTIVIDAD</b>	<b>4</b>
<b>COMPONENTES REACT.</b>	<b>4</b>
Main	5
Header.	6
Footer	7
Principal	7
ApiTest	8
CbxCities	9
CbxRoutes	10
InputWithValidation	11
Breadcrumb	12
ListCities	13
ListRoutes	14
CreateRoute	15
DeleteRoute	17
SearchRouteByName	18
<b>HOOKS UTILIZADOS EN LA SOLUCIÓN</b>	<b>20</b>
Hooks estándar	20
Hook personalizado	21
Otros Hooks de React	22
<b>VISTAS</b>	<b>23</b>
Creación de una ruta	23
Mensaje informando creación de ruta	23
Revisión de nueva ruta en el listado	24
Eliminación de la ruta 5	24
Ruta 5 eliminada correctamente	25
<b>CONCLUSIONES</b>	<b>26</b>

# INTRODUCCIÓN

El trabajo realizado consiste en el desarrollo de una aplicación web utilizando tecnologías modernas de desarrollo front-end, con un enfoque específico en el uso de React, una de las bibliotecas más populares y poderosas para la creación de interfaces de usuario interactivas. En esta aplicación, se hace uso de los principales conceptos fundamentales de React, tales como el manejo de estados mediante el hook `useState` y la gestión de efectos secundarios con `useEffect`, que permiten una gestión eficiente de la UI y la manipulación de datos en tiempo real.

Además, para optimizar y modularizar el código, se ha implementado un hook personalizado, adaptado a las necesidades específicas de la aplicación. Este custom hook ayuda a simplificar la lógica y mejora la reutilización de componentes dentro de la aplicación.

Para la gestión de rutas y la navegación entre las diferentes secciones de la aplicación, se ha utilizado React Router, permitiendo que los usuarios puedan acceder fácilmente a diferentes vistas de la aplicación, como mínimo cuatro rutas diferentes han sido implementadas, entre ellas las rutas para visualizar, crear y eliminar rutas de transporte, así como consultar información sobre precios y servicios.

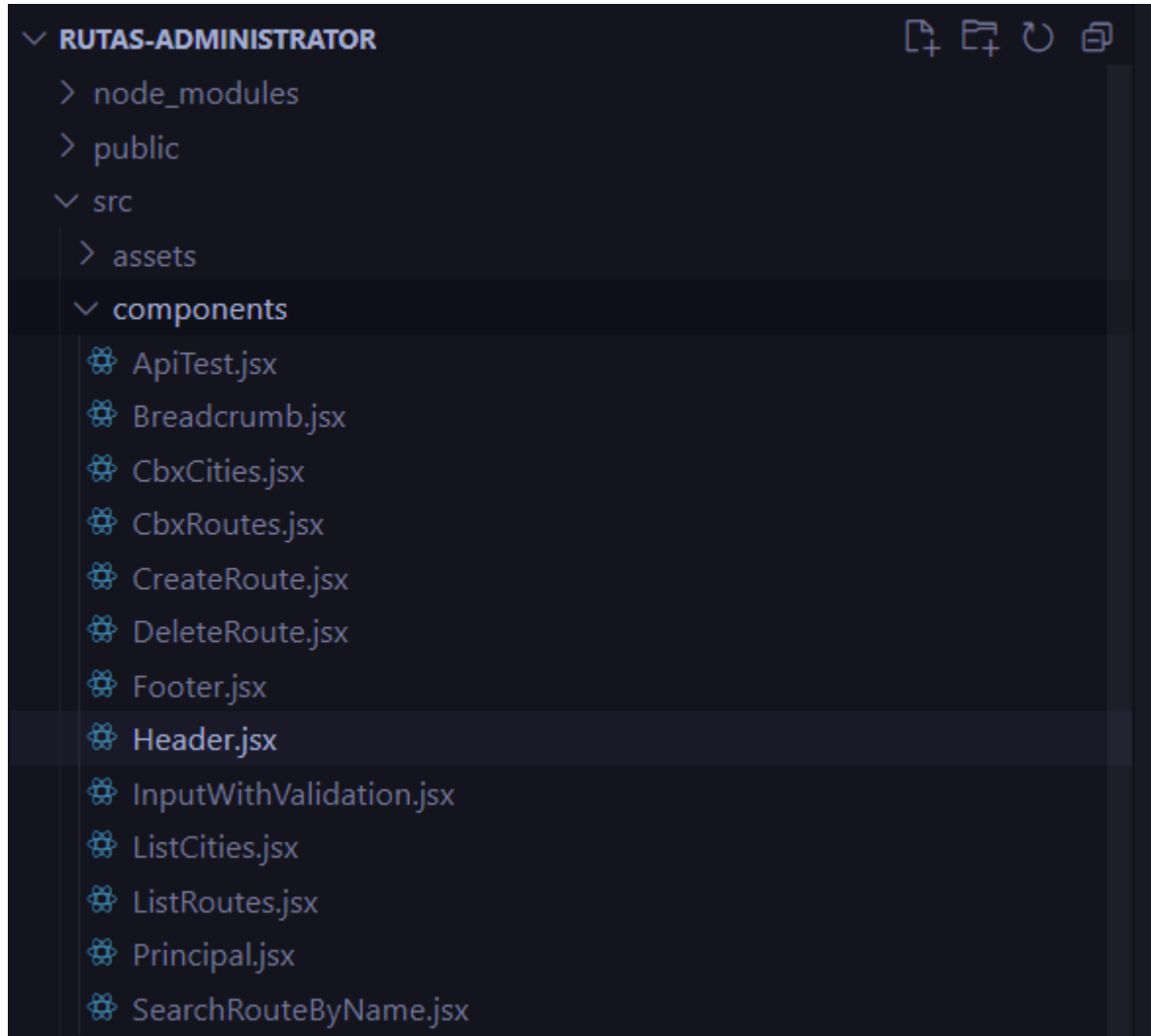
En cuanto a la parte visual y de diseño, se han aplicado estilos mediante hojas de estilo CSS, utilizando tanto Bootstrap como estilos personalizados, para ofrecer una interfaz amigable y coherente con el propósito de la aplicación. La integración de Bootstrap ha permitido el uso de una plantilla de diseño previamente estructurada para blogs, lo que ha facilitado la creación de una interfaz estética y funcional, mientras que los estilos personalizados han sido aplicados para adaptarse a las necesidades específicas del proyecto.

Este trabajo representa un enfoque integral para el desarrollo front-end de una aplicación dinámica y fácil de usar, que emplea buenas prácticas en la construcción de interfaces interactivas y con una estructura sólida basada en las tecnologías actuales.

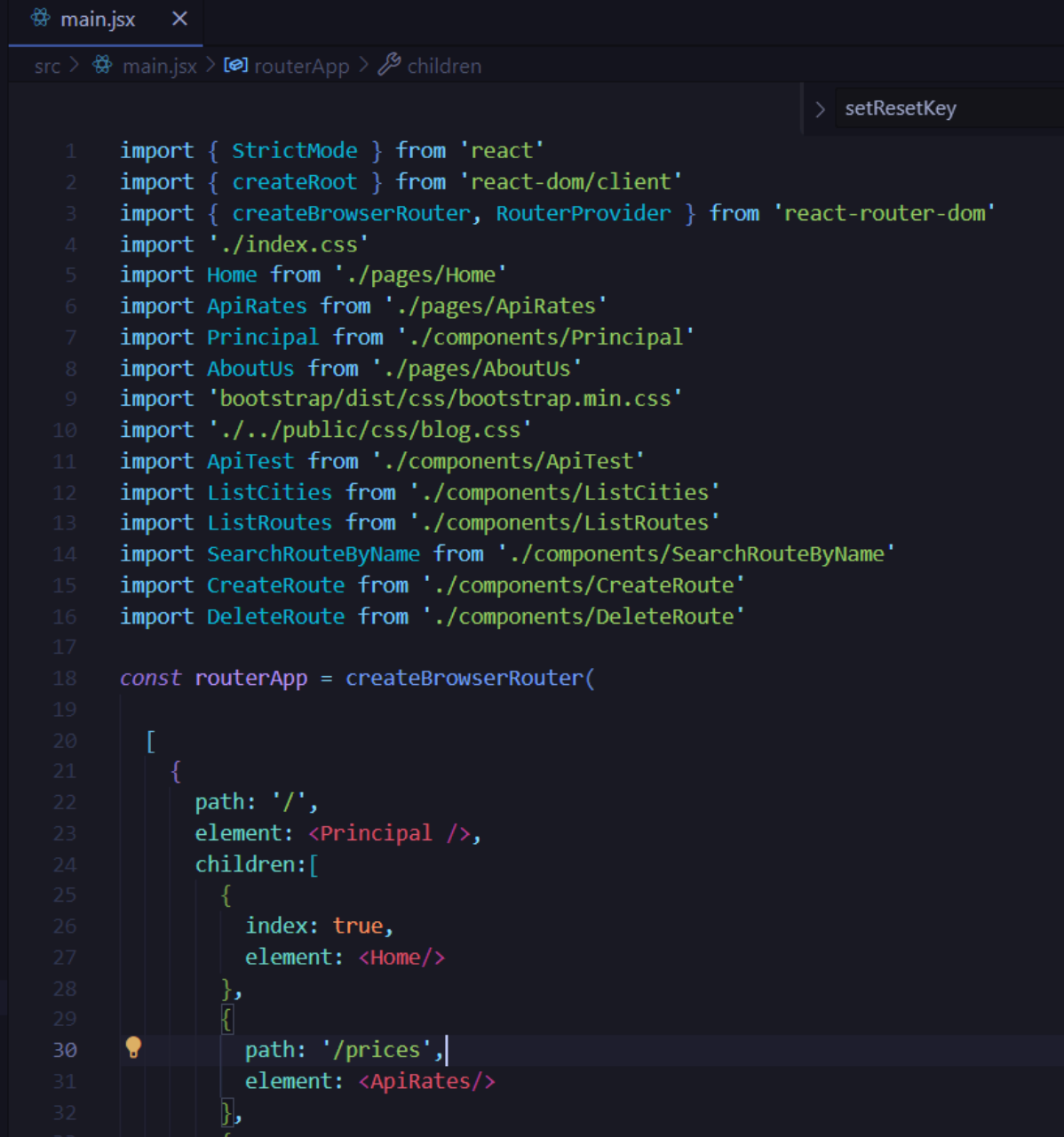
# DESARROLLO DE LA ACTIVIDAD

## COMPONENTES REACT.

A continuación se muestra imagen de los componentes desarrollados para la aplicación.



# Main



```
1 import { StrictMode } from 'react'
2 import { createRoot } from 'react-dom/client'
3 import { createBrowserRouter, RouterProvider } from 'react-router-dom'
4 import './index.css'
5 import Home from './pages/Home'
6 import ApiRates from './pages/ApiRates'
7 import Principal from './components/Principal'
8 import AboutUs from './pages/AboutUs'
9 import 'bootstrap/dist/css/bootstrap.min.css'
10 import './../public/css/blog.css'
11 import ApiTest from './components/ApiTest'
12 import ListCities from './components/ListCities'
13 import ListRoutes from './components/ListRoutes'
14 import SearchRouteByName from './components/SearchRouteByName'
15 import CreateRoute from './components/CreateRoute'
16 import DeleteRoute from './components/DeleteRoute'
17
18 const routerApp = createBrowserRouter(
19   [
20     {
21       path: '/',
22       element: <Principal />,
23       children:[
24         {
25           index: true,
26           element: <Home/>
27         },
28       ],
29     },
30     {
31       path: '/prices',
32       element: <ApiRates/>
33     },
34   ]
35 )
```

El componente utiliza react-router-dom para gestionar las rutas de navegación. Dentro de este componente se configura y se proporciona un enrutador que define las rutas de la aplicación. A continuación, se describe su estructura y funcionamiento:

## 1. Dependencias Importadas:

- Se importan varias dependencias clave, como StrictMode, createRoot de react-dom/client, y createBrowserRouter de react-router-dom.
- También se importan los estilos de la aplicación desde archivos CSS, incluyendo Bootstrap y un archivo de estilos personalizado (blog.css).

## 2. Enrutador:

- Se utiliza `createBrowserRouter` para definir el conjunto de rutas de la aplicación. El objeto `routerApp` contiene las rutas principales, donde cada ruta está asociada con un componente que se renderiza al acceder a esa ruta.
- La ruta principal `(/)` está asociada al componente `Principal`, que a su vez tiene varias rutas hijas (como `/prices`, `/api-test`, `/about-us`, etc.).
- Se configuran rutas específicas para diferentes páginas de la aplicación, como `Home`, `ApiRates`, `ApiTest`, `AboutUs`, `ListCities`, `ListRoutes`, `CreateRoute`, y `DeleteRoute`.

## 3. Rendimiento del Componente:

- El enrutador `RouterProvider` se monta en el DOM dentro de un contenedor de `StrictMode`. Esto asegura que la aplicación se renderice con las mejores prácticas de React para el desarrollo, como la verificación de errores y la detección de posibles problemas.

## 4. Estilos:

- Se incluyen dos archivos CSS importantes para el diseño visual de la aplicación: uno de `Bootstrap`, que proporciona una serie de estilos predefinidos para los componentes, y otro de `blog.css`, que es un archivo de estilos personalizado específico de la aplicación.

En resumen, este componente actúa como el punto de entrada para la aplicación React, configurando el enrutador para manejar la navegación entre diferentes componentes, y aplicando los estilos necesarios para la interfaz de usuario.

## Header.

---

Home	Api test	Precios	Sobre nosotros
------	----------	---------	----------------

---

El componente `Header` incluye una barra de navegación con enlaces a las secciones: `Home`, `Api test`, `Precios` y `Sobre nosotros`. Utiliza clases de `Bootstrap` para el diseño responsivo y organiza el contenido de manera clara y accesible. Su propósito es proporcionar un punto de navegación principal para los usuarios.

## Footer

© 2025 RUTAS ADMINISTRATOR. Todos los derechos reservados.

Diseñado para optimizar la gestión de rutas en Colombia.

El componente `Footer` muestra un pie de página con dos párrafos. El primero indica los derechos de autor para "RUTAS ADMINISTRATOR" en 2025, y el segundo describe el propósito del sistema: "Diseñado para optimizar la gestión de rutas en Colombia". Utiliza clases de Bootstrap para darle estilo, como `py-5`, `text-center`, `text-body-secondary`, y `bg-body-tertiary`, lo que asegura un diseño limpio y responsivo. Su propósito es proporcionar información adicional al final de la página.

## Principal

El componente Principal sirve como un contenedor para la estructura principal de la página. Incluye:

- Header: Muestra el encabezado de la aplicación con el logo y la barra de navegación.
- Outlet: Es el espacio donde se renderizan las rutas hijas, permitiendo la carga dinámica de contenido dependiendo de la ruta activa.
- Footer: Muestra el pie de página con la información de derechos de autor y descripción del propósito.

Utiliza la clase container de Bootstrap para el diseño responsivo del contenido principal. Su propósito es proporcionar una estructura común para las páginas de la aplicación, incluyendo el encabezado, el contenido principal y el pie de página.

## Pruebas de la API

En este módulo puedes realizar las siguientes operaciones utilizando nuestra API:

[Listar ciudades](#)[Listar rutas](#)[Buscar rutas por nombre](#)[Crear una ruta](#)[Eliminar una ruta](#)

El componente ApiTest presenta una interfaz para interactuar con la API de la aplicación. Incluye:

- Un encabezado que indica que el módulo es para "Pruebas de la API".
- Una lista de operaciones disponibles que se pueden realizar con la API:
  - Listar ciudades
  - Listar rutas
  - Buscar rutas por nombre
  - Crear una ruta
  - Eliminar una ruta

Cada operación está representada como un enlace (Link) que redirige a las rutas correspondientes para ejecutar cada acción.

Utiliza las clases de Bootstrap container, mt-4, list-group y btn btn-link para el estilo y la disposición responsiva. Su propósito es proporcionar al usuario un conjunto de opciones para interactuar con la API de manera fácil y clara.



## CbxCities

Ciudad origen

Seleccione

Seleccione

Bogotá

Medellín

Cali

Barranquilla

Cartagena

Bucaramanga

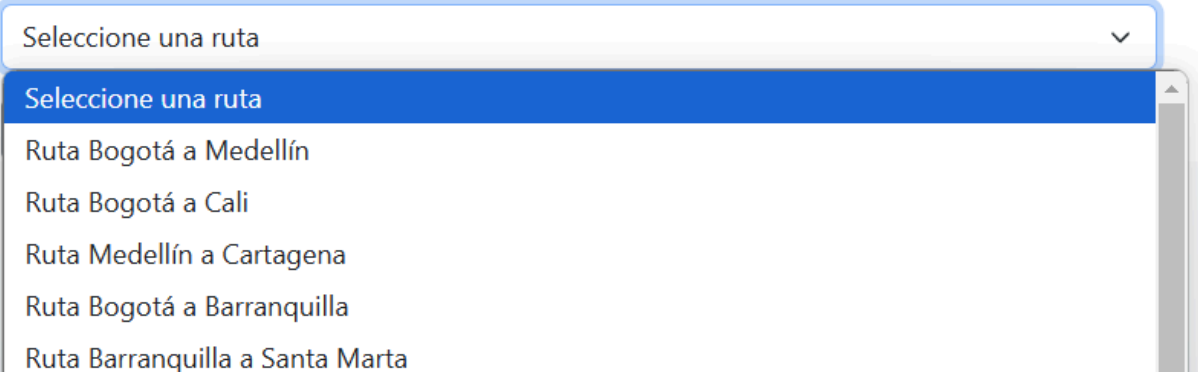
El componente CbxCities es un selector (dropdown) que permite al usuario seleccionar una ciudad de una lista cargada desde una API. Su funcionalidad es la siguiente:

- Obtención de datos: Utiliza un hook personalizado useFetchData para obtener la lista de ciudades desde la API en la URL <http://localhost:9090/ciudades/api/v1/ciudades>.
- Manejo de estado: Se gestionan tres estados: cities (datos de las ciudades), loading (si los datos están siendo cargados) y error (si ocurre un error al cargar los datos).
- Manejo de selección: Cuando el usuario selecciona una ciudad, se dispara la función onCitySelect pasada como prop, pasando el id de la ciudad seleccionada.
- Interfaz:
  - Un label para el título del selector, que se recibe como prop title.
  - Un select que muestra las ciudades cargadas, con un valor predeterminado vacío y un mensaje de "Seleccione".
  - En caso de que los datos estén cargando, muestra "Loading cities...". Si hay un error, muestra "Error loading cities".

Utiliza clases de Bootstrap como mb-3, form-label, y form-select para el diseño. Su propósito es proporcionar un selector de ciudades funcional y estilizado, con manejo de estados de carga y error.

## CbxRoutes

Selecciona una ruta:



The screenshot shows a web form with a label 'Selecciona una ruta:' and a dropdown menu. The dropdown menu is open, displaying a list of routes. The first item in the list is 'Seleccione una ruta', which is highlighted in blue. Below it are five other routes: 'Ruta Bogotá a Medellín', 'Ruta Bogotá a Cali', 'Ruta Medellín a Cartagena', 'Ruta Bogotá a Barranquilla', and 'Ruta Barranquilla a Santa Marta'. The dropdown menu has a light blue border and a small downward arrow icon on the right side.

El componente CbxRoutes es un selector de rutas que permite al usuario elegir una ruta de una lista obtenida desde una API. Su funcionamiento es el siguiente:

- Obtención de datos: Utiliza el hook personalizado `useFetchData` para obtener la lista de rutas desde la API en la URL `http://localhost:9090/rutas/api/v1/rutas`. Tiene un control de recarga de datos con el estado `shouldReload`, que se activa si la prop `reload` cambia.
- Manejo de estado:
  - `shouldReload`: Controla si debe recargarse la lista de rutas.
  - `wasRouteSelected`: Indica si se ha seleccionado una ruta.
  - `selectedRouteDetails`: Almacena los detalles de la ruta seleccionada (nombre, ciudad de origen, ciudad de destino y distancia).
- Selección de ruta:

Cuando el usuario selecciona una ruta, la función `handleChange` se encarga de:

  - Actualizar la ruta seleccionada.
  - Mostrar los detalles de la ruta seleccionada.
  - Si no se selecciona ninguna ruta, limpia los detalles y deselecta la ruta.
- Efectos
  - Se utilizan tres `useEffect`:
    - El primero restablece la selección de la ruta si `selectedRouteId` cambia.
    - El segundo activa la recarga si `reload` cambia.
    - El tercero desactiva la recarga después de que se haya realizado.
  - Interfaz
    - Un select para elegir una ruta de la lista.
    - Mensajes de carga o error según el estado de la obtención de datos.
    - Si una ruta es seleccionada, se muestran los detalles de la ruta elegida

Utiliza clases de Bootstrap como `form-label`, `form-select`, y `text-muted` para el diseño. Su propósito es proporcionar un selector de rutas con detalles de la ruta seleccionada, con manejo de estados de carga, error y selección.

# InputWithValidation

## Ingrese el nombre de la ruta

Solo se permiten caracteres alfanuméricos.

Aceptar

El componente `InputWithValidation` es un campo de entrada de texto con validación que permite al usuario ingresar un valor alfanumérico. Su funcionalidad es la siguiente:

- Estado:
  - `inputValue`: Almacena el valor del campo de entrada.
  - `error`: Almacena el mensaje de error cuando el valor ingresado no cumple con los criterios de validación.
- Validación:
  - Entrada alfanumérica: El valor ingresado es validado para asegurarse de que solo contenga caracteres alfanuméricos y espacios. Si el valor no es válido, se muestra un mensaje de error.
  - Campo no vacío: Al presionar el botón de "Aceptar", se valida que el campo no esté vacío. Si está vacío, se muestra un mensaje de error; si es válido, se ejecuta la función `onSubmit` con el valor ingresado y luego se limpia el campo de entrada.
- Interfaz
  - Un encabezado que muestra el título pasado como prop.
  - Un campo de entrada de texto (`input`) donde el usuario puede escribir.
  - Un botón para enviar el valor.
  - Si hay un error, se muestra un mensaje de error debajo del campo de entrada.

Utiliza clases de Bootstrap como `form-control`, `text-danger`, y `btn btn-primary` para el estilo. Su propósito es proporcionar un campo de entrada con validación en tiempo real y validación adicional al enviar el formulario.

## Breadcrumb

---

Home

Api test

---

[Api test](#) / Listar ciudades

El componente Breadcrumb es un componente de navegación que muestra una ruta de navegación (breadcrumb) para indicar la ubicación actual del usuario en la aplicación. Su funcionalidad es la siguiente:

- Props:
  - `currentPage`: Recibe el nombre de la página actual que se está visualizando, y lo muestra como la última parte de la ruta de navegación.
- Estructura:
  - Enlace de navegación: Muestra un enlace (Link) hacia la página `/api-test` como parte de la ruta de navegación.
  - Página activa: Muestra el nombre de la página actual (`currentPage`) en un `li` con la clase `active`, lo que indica que es la página en la que el usuario se encuentra actualmente.
- Interfaz:
  - Utiliza las clases de Bootstrap `breadcrumb`, `breadcrumb-item`, y `active` para construir la ruta de navegación.
  - El `aria-label="breadcrumb"` mejora la accesibilidad para los usuarios con lectores de pantalla.

Este componente se utiliza comúnmente para proporcionar un contexto de navegación al usuario, ayudando a entender en qué parte de la aplicación se encuentra.

# Ciudades

Bogotá
Medellín
Cali
Barranquilla
Cartagena
Bucaramanga
Pereira
Santa Marta
Manizales
Cúcuta

El componente ListCities es responsable de mostrar una lista de ciudades obtenida de una API. Su funcionalidad es la siguiente:

- Obtención de datos

- Usa el hook personalizado useFetchData para hacer una solicitud a la API en la URL 'http://localhost:9090/ciudades/api/v1/ciudades' y obtener la lista de ciudades.
- Si los datos están siendo cargados, se muestra un spinner de carga.
- Si ocurre un error durante la carga, se muestra un mensaje de error en una alerta.
- Interfaz
  - Breadcrumb: Se muestra un componente Breadcrumb que indica la página actual ("Listar ciudades").
  - Listado de ciudades: Si los datos son recibidos con éxito, se muestra una lista (<ul>) de ciudades, donde cada ciudad se representa como un elemento de lista (<li>) con su nombre.
  - Utiliza clases de Bootstrap para el estilo, como `list-group` para la lista de elementos y `spinner-border` para el indicador de carga.

Este componente tiene como objetivo mostrar un listado dinámico de ciudades y manejar los estados de carga y error de manera adecuada.

## ListRoutes

Home	Api test	Precios	Sobre nosotros
------	----------	---------	----------------

[Api test](#) / Listar rutas

### Lista de Rutas

Id	Nombre	Ciudad Origen	Ciudad Destino	Distancia (km)
1	Ruta Bogotá a Medellín	Bogotá	Medellín	415
2	Ruta Bogotá a Cali	Bogotá	Cali	465
3	Ruta Medellín a Cartagena	Medellín	Cartagena	640
4	Ruta Bogotá a Barranquilla	Bogotá	Barranquilla	970
5	Ruta Barranquilla a Santa Marta	Barranquilla	Santa Marta	103

El componente ListRoutes tiene como objetivo mostrar una lista de rutas obtenida de una API. A continuación, se describe su funcionalidad:

- Obtención de datos:
  - Utiliza el hook personalizado useFetchData para realizar una solicitud HTTP a la URL 'http://localhost:9090/rutas/api/v1/rutas' y obtener una lista de rutas.

- Si los datos están siendo cargados, se muestra un mensaje de carga dentro de una alerta ("Cargando rutas...").
- Si ocurre un error durante la carga de los datos, se muestra un mensaje de error dentro de una alerta ("Error: {error}").
- Interfaz:
  - Breadcrumb: Muestra un componente Breadcrumb que indica la página actual ("Listar rutas").
  - Tabla de rutas: Una vez que los datos son cargados correctamente, se muestra una tabla con la información de las rutas:
    - Columnas:
      - Id: Identificador de la ruta.
      - Nombre: Nombre de la ruta.
      - Ciudad Origen: Nombre de la ciudad de origen de la ruta.
      - Ciudad Destino: Nombre de la ciudad de destino de la ruta.
      - Distancia (km): Distancia entre las ciudades de origen y destino.
      - La tabla utiliza clases de Bootstrap, como table y table-striped, para su estilo.

Este componente gestiona los estados de carga y error y presenta la información de las rutas de manera organizada en una tabla.

## CreateRoute

[Api test](#) / Crear ruta

### Crear Ruta

Nombre de la ruta

Ciudad origen

Ciudad destino

Kilómetros

Este componente permite la creación de rutas a través de un formulario. A continuación, se describe su funcionalidad:

Estado y lógica:

**1. Estados de entrada:**

- routeName: almacena el nombre de la ruta.
- originCityId: almacena el ID de la ciudad de origen.
- destinationCityId: almacena el ID de la ciudad de destino.
- distance: almacena la distancia en kilómetros entre las dos ciudades.
- routeCreated: almacena los datos de la ruta creada después de la solicitud exitosa.
- errorMessage: muestra mensajes de error si hay algún problema con la entrada o la creación de la ruta.

**2. Funcionalidad:**

- **Validación del formulario:**
  - Los campos de nombre de la ruta, ciudad de origen, ciudad de destino y distancia son obligatorios.
  - La distancia debe ser mayor que 0.
  - La ciudad de origen y la de destino no pueden ser iguales.
  - El nombre de la ruta solo permite caracteres alfanuméricos y un máximo de 50 caracteres.
- **Creación de la ruta:**
  - La función createRoute realiza una solicitud POST para enviar los datos del formulario a la API.
  - Si la solicitud es exitosa, se actualiza el estado con la ruta creada y se muestra un mensaje de éxito.
  - Si hay un error (como un código de estado 400), se muestra el mensaje de error correspondiente.
- **Incrementar distancia:**
  - Se incluye un botón para aumentar la distancia en 1 km.
- **Cancelar la creación de la ruta:**
  - Un botón de cancelación resetea el formulario y elimina los mensajes de error.

**3. Interfaz de usuario:**

- Un campo de entrada para el nombre de la ruta con validación en tiempo real.
- Un componente CbxCities se utiliza para seleccionar la ciudad de origen y destino.
- Un campo de entrada para la distancia con un botón para incrementar el valor.
- Botones para crear la ruta o cancelar el proceso.
- Si la ruta se crea correctamente, se muestra un mensaje de éxito con los detalles de la ruta creada.



Componentes usados:

- **Breadcrumb:** Muestra la página actual "Crear ruta".
- **CbxCities:** Componente para seleccionar una ciudad, usado tanto para la ciudad de origen como de destino.

Este componente proporciona una experiencia interactiva y valida adecuadamente los datos antes de enviarlos a la API, además de gestionar el estado y mostrar mensajes de éxito o error según corresponda.

## DeleteRoute

[Api test](#) / Eliminar una Ruta

### Eliminar una Ruta

Selecciona una ruta:

Seleccione una ruta

Cancelar

Eliminar

Este componente permite eliminar una ruta existente seleccionada por el usuario. A continuación, se describe su funcionalidad:

Estado y lógica:

#### 1. Estados de entrada:

- `selectedRouteId`: almacena el ID de la ruta seleccionada para su eliminación.
- `successMessage`: muestra un mensaje de éxito cuando la ruta se elimina correctamente.
- `errorMessage`: muestra un mensaje de error si ocurre algún problema al intentar eliminar la ruta.
- `reloadRoutes`: controla la recarga de rutas después de la eliminación.

#### 2. Funcionalidad:

- **Selección de ruta:**
  - El usuario selecciona la ruta a eliminar desde un componente `CbxRoutes`. Cuando se selecciona una ruta, el estado `selectedRouteId` se actualiza con el ID de la ruta seleccionada.
- **Confirmación de eliminación:**
  - Antes de proceder con la eliminación, se solicita una confirmación del usuario a través de `window.confirm`.

- **Eliminación de la ruta:**
  - Se realiza una solicitud DELETE a la API con el ID de la ruta seleccionada. Si la respuesta es exitosa (status 204), se muestra un mensaje de éxito y se limpia el formulario.
  - Si hay un error, se muestra un mensaje de error adecuado.
- **Cancelar la acción:**
  - Si el usuario decide cancelar, el estado se limpia y se eliminan los mensajes de éxito y error.

### 3. Interfaz de usuario:

- Un componente CbxRoutes permite seleccionar una ruta para eliminar.
- Los botones permiten al usuario cancelar la acción o eliminar la ruta seleccionada.
- Los mensajes de éxito o error se muestran dinámicamente dependiendo del resultado de la operación.

Componentes usados:

- **Breadcrumb:** Muestra la página actual "Eliminar una Ruta".
- **CbxRoutes:** Componente para seleccionar una ruta para eliminar.

Este componente permite eliminar una ruta de forma interactiva con confirmación, validación de selección y gestión adecuada de mensajes de éxito o error.

## SearchRouteByName

[Api test](#) / Buscar rutas por nombre

### Buscar rutas por nombre

#### Ingrese el nombre de la ruta

#### Rutas encontradas:

ID	Nombre	Ciudad Origen	Ciudad Destino	Distancia (km)
9	Ruta Pereira a Manizales	Pereira	Manizales	50
10	Ruta Cali a Pereira	Cali	Pereira	215

Este componente permite buscar rutas por nombre y mostrar los resultados, con manejo de estados de carga, error y resultados vacíos. A continuación, se detalla su funcionalidad:

Estado y lógica:

**1. Estados de entrada:**

- `searchUrl`: almacena la URL de la API para realizar la búsqueda de rutas por nombre.
- `hasQueried`: indica si el usuario ha realizado una búsqueda.

**2. Lógica:**

- **Búsqueda de rutas:**
  - El componente `InputWithValidation` permite al usuario ingresar el nombre de una ruta. Al hacer clic en buscar, se actualiza el estado `searchUrl` con la URL de la API para buscar la ruta por nombre.
  - Se realiza una solicitud GET a la API `http://localhost:9090/rutas/api/v1/rutas/buscar?nombre=....`
- **Manejo de estados de carga y error:**
  - Si los datos están cargando, se muestra un mensaje de carga.
  - Si ocurre un error, se muestra un mensaje de error, dependiendo del código de estado de la respuesta (500 para errores generales y 204 para no encontrar resultados).
- **Mostrar resultados:**
  - Si la búsqueda devuelve rutas, se muestra una tabla con las rutas encontradas, con las columnas de ID, nombre, ciudad origen, ciudad destino y distancia.

**3. Interfaz de usuario:**

- Un componente `InputWithValidation` para ingresar el nombre de la ruta y realizar la búsqueda.
- Mensajes dinámicos según el estado de la búsqueda, como carga, error o no resultados.
- Una tabla para mostrar las rutas encontradas, con la información de ID, nombre, ciudades de origen y destino, y distancia en kilómetros.

Componentes usados:

- **Breadcrumb**: Muestra la página actual "Buscar rutas por nombre".
- **InputWithValidation**: Componente para la validación e ingreso del nombre de la ruta.

Este componente facilita la búsqueda de rutas por nombre, con manejo de errores, visualización de resultados y mensajes adecuados según el estado de la búsqueda.

# HOOKS UTILIZADOS EN LA SOLUCIÓN

## Hooks estándar

- **useState:**
  - **Componente CreateRoute:** Se utiliza para manejar los valores de los campos del formulario como routeName, originCityId, destinationCityId, distance, routeCreated, errorMessage, entre otros.
  - **Componente DeleteRoute:** Maneja el estado de la ruta seleccionada (selectedRouteId), los mensajes de éxito y error (successMessage, errorMessage), y el estado de recarga de rutas (reloadRoutes).
  - **Componente SearchRouteByName:** Almacena la URL de la API (searchUrl) para realizar la búsqueda de rutas, el estado de consulta (hasQueried) y los resultados obtenidos de la búsqueda (routes).

El hook useState es fundamental en cada uno de estos componentes para gestionar los valores dinámicos y los estados de interacción del usuario (formulario, selección de rutas, resultados de búsqueda, mensajes de error o éxito).

## Hook personalizado

```
useFetchData.jsx X
src > hooks > useFetchData.jsx > useFetchData > useEffect() callback >

1  import { useState, useEffect } from 'react';
2
3  const useFetchData = (url, shouldReload) => {
4    const [data, setData] = useState(null);
5    const [loading, setLoading] = useState(true);
6    const [error, setError] = useState(null);
7    const [status, setStatus] = useState(null);
8
9    useEffect(() => {
10      const fetchData = async () => {
11        try {
12          setError(null);
13          const response = await fetch(url);
14          if (!response.ok) {
15            throw new Error('Error fetching data');
16          }
17          setStatus(response.status);
18          const result = await response.json();
19          setData(result);
20        } catch (err) {
21          setError('Error fetching data');
22        } finally {
23          setLoading(false);
24        }
25      };
26
27      if (shouldReload) {
28        ⚠️ fetchData();
29      }
30
31      fetchData();
32    }, [url, shouldReload]);
33
34    return { data, loading, error, status };
35  };
```

- **useFetchData:**
  - **Componente ListRoutes y Componente SearchRouteByName:** Este hook personalizado maneja la lógica de obtener los datos de las rutas desde el servidor. Realiza una llamada fetch a la API y gestiona el estado de carga, error y los datos obtenidos.
- **Necesidad y valor:**
  - **Centralización de lógica de datos:** En lugar de duplicar la misma lógica de solicitud HTTP y manejo de errores en cada componente, useFetchData centraliza esta lógica. Esto hace que el código sea más limpio y reutilizable.
  - **Reutilización:** Este hook es reutilizado en varios componentes, mejorando la mantenibilidad y evitando redundancias. Cada vez que sea necesario realizar una solicitud similar, este hook puede ser usado sin duplicar el código.

## Otros Hooks de React

- **useEffect:**
  - En el componente CbxRoutes, se utiliza el hook useEffect para gestionar los efectos secundarios relacionados con la carga de rutas desde la API. Este hook se ejecuta cuando el componente se monta y cada vez que se actualiza el reload de las rutas, lo que asegura que se realice una nueva solicitud a la API para obtener las rutas actualizadas. En concreto, useEffect se encarga de invocar la función que obtiene los datos de las rutas y actualiza el estado de la lista de rutas, proporcionando una manera automática de mantener los datos sincronizados con la interfaz. Además, este comportamiento ayuda a optimizar el rendimiento, evitando solicitudes innecesarias y asegurando que solo se realicen cuando sea necesario.

En resumen, los hooks utilizados son:

- **useState:** Para manejar el estado local en cada componente.
- **useFetchData:** Un hook personalizado para gestionar las solicitudes HTTP y el manejo de datos de la API, mejorando la reutilización y centralización de la lógica de obtención de datos.

# VISTAS

En la aplicación se ha utilizado Bootstrap 5.3.3, específicamente la plantilla Blog, para diseñar una interfaz moderna y adaptable a diferentes dispositivos. Se aprovecharon los componentes predefinidos de Bootstrap para estructurar las páginas de manera eficiente. Además, se incorporaron estilos CSS personalizados para ajustar ciertos elementos visuales según las necesidades específicas de la aplicación.

## Creación de una ruta

[Api test](#) / Crear ruta

### Crear Ruta

Nombre de la ruta

Ruta Cartagena Bogota

Ciudad origen

Cartagena

Ciudad destino

Bogotá

Kilómetros

75

+1

Crear Ruta

Cancelar

## Mensaje informando creación de ruta

[Api test](#) / Crear ruta

### Crear Ruta

Nombre de la ruta

Ciudad origen

Seleccione

Ciudad destino

Seleccione

Kilómetros

0

+1

Crear Ruta

Cancelar

#### Se ha creado la ruta: Ruta Cartagena Bogota

- Ciudad Origen: Cartagena
- Ciudad Destino: Bogotá
- Distancia: 75 km

# Revisiòn de nueva ruta en el listado

RUTAS ADMINISTRATOR

Home

Api test

Precios

Sobre nosotros

Api test

Listar rutas

Lista de Rutas

Id	Nombre	Ciudad Origen	Ciudad Destino	Distancia (km)
1	Ruta Bogotá a Medellín	Bogotá	Medellín	415
2	Ruta Bogotá a Cali	Bogotá	Cali	465
3	Ruta Medellín a Cartagena	Medellín	Cartagena	640
4	Ruta Bogotá a Barranquilla	Bogotá	Barranquilla	970
5	Ruta Barranquilla a Santa Marta	Barranquilla	Santa Marta	103
6	Ruta Cartagena a Santa Marta	Cartagena	Santa Marta	209
7	Ruta Bogotá a Bucaramanga	Bogotá	Bucaramanga	400
8	Ruta Bucaramanga a Cúcuta	Bucaramanga	Cúcuta	200
9	Ruta Pereira a Manizales	Pereira	Manizales	50
10	Ruta Cali a Pereira	Cali	Pereira	215
16	Ruta Cali a Bucaramanga	Bogotá	Bogotá	6
18	Ruta Cartagena Bogota	Cartagena	Bogotá	75

# Eliminaciòn de la ruta 5

RUTAS ADMINISTRATOR

Home

Api test

Precios

Sobre nosotros

Api test

Eliminar una Ruta

Eliminar una Ruta

Selecciona una ruta:

Ruta 5

Informaciòn de la Ruta

Nombre:

Ruta 5

Ciudad Origen:

Medellín

Ciudad Destino:

Cúcuta

Distancia:

12 km

Cancelar

Eliminar

© 2025 RUTAS ADMINISTRATOR. Todos los derechos reservados.

Diseñado para optimizar la gestiòn de rutas en Colombia.



Ruta 5 eliminada correctamente

RUTAS ADMINISTRATOR

Home

Api test

Precios

Sobre nosotros

[Api test](#) / Eliminar una Ruta

Eliminar una Ruta

Selecciona una ruta:

Seleccione una ruta

La ruta ha sido eliminada correctamente.

Cancelar

Eliminar

# CONCLUSIONES

1. **Aplicación práctica de React y sus conceptos fundamentales:** El desarrollo de la aplicación permitió comprender y aplicar los principios fundamentales de React, como el uso de componentes funcionales, el manejo de estados con useState y la gestión de efectos secundarios con useEffect. Estas herramientas demostraron ser esenciales para construir interfaces dinámicas y reactivas, fortaleciendo el entendimiento de su funcionamiento y mejores prácticas.
2. **Reutilización de código y optimización mediante custom hooks:** La implementación de un custom hook permitió modularizar la lógica de la aplicación y reutilizar funcionalidades específicas, optimizando el código y mejorando su mantenibilidad. Este enfoque reforzó la importancia de abstraer lógica compleja en componentes reutilizables para simplificar el desarrollo.
3. **Navegación eficiente y estructurada con React Router:** La utilización de React Router para gestionar múltiples rutas dentro de la aplicación proporcionó un sistema de navegación claro y eficiente. Este aprendizaje subrayó el valor de estructurar aplicaciones modernas mediante rutas dinámicas, facilitando una experiencia de usuario fluida y bien organizada.
4. **Importancia del diseño y la estética en el desarrollo front-end:** La integración de Bootstrap y estilos personalizados mediante CSS resaltó la relevancia de combinar frameworks de diseño con ajustes específicos para lograr una interfaz visualmente atractiva y funcional. Este trabajo demostró cómo un diseño bien implementado mejora significativamente la usabilidad y el impacto de una aplicación.