

Star Wars Collection

Bienvenido al proyecto “Star Wars Collection”, una aplicación web en React que permite explorar personajes, planetas y vehículos del universo Star Wars. La aplicación presenta una interfaz interactiva y visualmente atractiva utilizando diversas tecnologías web.

Descripción del Proyecto.

“**Star Wars Collection**” es una aplicación web diseñada para mostrar información sobre personajes, planetas y vehículos del universo Star Wars. Los usuarios pueden explorar estas categorías, ver detalles específicos y añadir elementos a una lista de favoritos. La aplicación también incluye una pantalla de bienvenida animada y una barra de navegación fija.

Tecnologías Utilizadas.

- **React:** Librería para construir interfaces de usuario.
- **React Router:** Para la gestión de rutas y navegación.
- **Bootstrap:** Para los estilos y la disposición de la interfaz.
- **CSS:** Para la personalización de los estilos visuales.

Estructura del Proyecto.

```
src/
|-- component/
|   |-- card.js
|   |-- favorites.js
|   |-- footer.js
|   |-- navbar.js
|   |-- ScrollToTop.js
|   |-- SplashScreen.js
|-- img/
|   |-- soldado.png
|   |-- star-wars.png
|-- styles/
|   |-- index.css
|-- views/
|   |-- cardinfo.js
|   |-- home.js
|-- store/
|   |-- appContext.js
|   |-- flux.js
|-- layout.js
|-- index.js
|-- pagination.js
```

Index.js

Este es el punto de entrada principal de la aplicación. Aquí se importa el archivo CSS global y se renderiza el componente principal `Layout`.

```
● ● ●

import React from 'react';// Importa React
import { createRoot } from 'react-dom/client';// Importa ReactDOM para
renderizar la aplicación
import "../styles/index.css"; // Importa el archivo de estilos CSS
import Layout from './layout.js';// Importa el componente principal de la
aplicación

// Obtiene el contenedor de la aplicación en el DOM
const root = createRoot(document.querySelector("#app"));

root.render(<Layout />); // Renderiza el componente Layout dentro del
contenedor root
```

Home.js

Página principal que muestra la lista de personajes, planetas y vehículos. Incluye botones para cargar más datos y componentes de visualización.

```
● ● ●

import React from "react";
import { Card } from "../component/card";
import "../.../styles/index.css";

export const Home = () => {
  return (
    <div>
      <div className="container">
        <h1 className="text-center mb-4">People</h1>
        <Card type="people" />
        <h1 className="text-center mb-4 mt-5">Planets</h1>
        <Card type="planets" />
        <h1 className="text-center mb-4 mt-5">Vehicles</h1>
        <Card type="vehicles" />
      </div>
    </div>
  );
};
```

Renderizamos los componentes Card con una propiedad type establecida.

Layout.js

El componente Layout organiza la estructura principal de la aplicación, incluyendo la navegación, el enrutamiento y la presentación de componentes comunes como la barra de navegación, el pie de página y la pantalla de carga. Utiliza **BrowserRouter** para manejar el enrutamiento, **ScrollToTop** para controlar el desplazamiento, y define rutas específicas para diferentes vistas y componentes. La función **injectContext** envuelve el componente para proporcionar un contexto global a toda la aplicación.



```
import React from 'react'; // Importa React para poder utilizar JSX
import { BrowserRouter, Route, Routes } from 'react-router-dom';
//componentes para el enrutamiento
import ScrollToTop from './component/scrollToTop'; // Componente para
desplazar la vista al inicio en cambio de ruta
import 'bootstrap/dist/css/bootstrap.min.css'; // Estilos de Bootstrap
import '../styles/index.css'; // Importa estilos CSS personalizados
import { Home } from './views/home'; // Importa el componente Home
import injectContext from './store/appContext'; //función para inyectar
contexto
import { Navbar } from './component/navbar'; // Componente Navbar
import { Footer } from './component/footer'; // Componente Footer
import { Cardinfo } from './views/cardinfo'; // Componente Cardinfo
import { Favorites } from './component/favorites'; // Componente Favorites
import { SplashScreen } from './component/Splashscreen'; // Componente
SplashScreen
import { Card } from './component/card'; // Componente Card

const Layout = () => {
  const basename = process.env.BASENAME || ''; // Define la base de la
ruta, usando el entorno o una cadena vacía por defecto

  return (
    <div className="background">
      <BrowserRouter basename={basename}> /* Configura el enrutamiento con
una base opcional */
        <ScrollToTop> /*Desplaza la vista al inicio en cambio de ruta */
          <SplashScreen /> /* Muestra la pantalla de presentación*/
          <Navbar /> /* Muestra la barra de navegación */
          <Routes> /* Define las rutas de la aplicación */
            <Route path="/" element={<Home />} /> /*Ruta Home */
            <Route path="/card" element={<Card />} /> /* Ruta Card */
            <Route path="/:type/:uid" element={<Cardinfo />} /> /* Ruta
Cardinfo */
            <Route path="/favorites" element={<Favorites />} /> /* Ruta
Favorites */
            <Route path="*" element={<h1>Not found!</h1>} /> /* Ruta para
mostrar un mensaje de "No encontrado" para rutas no definidas */
          </Routes>
          <Footer /> /* Muestra el pie de página */
        </ScrollToTop>
      </BrowserRouter>
    </div>
  );
};

export default injectContext(Layout);
// Exporta el componente Layout envuelto en el contexto
```

Appcontext.js

Proporciona el contexto global para la aplicación. Incluye el estado global y las funciones que actualizan ese estado. La función `injectContext` envuelve el componente principal en el `Context.Provider`, permitiendo que los componentes hijos accedan al estado global.

```
● ● ●

import React, { useState, useEffect } from "react"; // Importa
                                                    React, useState y useEffect
import getState from "./flux.js"; // Importa la función getState que
                                    configura el estado global

export const Context = React.createContext(null); // Crea un contexto con
                                                 valor inicial null

// Higher-order component (HOC) que envuelve el componente pasado y
// proporciona el contexto
const injectContext = PassedComponent => {
    // Componente que envuelve al componente pasado y gestiona el estado
    // global
    const StoreWrapper = props => {
        // Define el estado del contexto utilizando useState
        const [state, setState] = useState(
            getState({
                // Funciones para acceder y actualizar el estado global
                getStore: () => state.store,
                getActions: () => state.actions,
                setStore: updatedStore =>
                    setState({
                        store: Object.assign(state.store, updatedStore), // Actualiza
                        el estado del store
                        actions: { ...state.actions } // Mantiene las acciones actuales
                    })
            })
        );

        // useEffect se ejecuta después de la renderización inicial (vacío en
        este caso)
        useEffect(() => {
            [], []
        });

        return (
            <Context.Provider value={state}> {/* Proporciona el contexto con el
                                                estado global */}
                <PassedComponent {...props} /> {/* Renderiza el componente pasado
                                                con sus props */}
            </Context.Provider>
        );
    };

    return StoreWrapper; // Retorna el componente envolvente con el contexto
};

export default injectContext; // Exporta la función que inyecta el contexto
```

Flux.js

Contiene las funciones de acción para gestionar el estado global. Aquí se definen las funciones para obtener datos de la API de Star Wars y manejar los favoritos.

```
● ● ●

const getState = ({ getStore, getActions, setStore }) => {
    return {
        store: {
            people: [], // Lista de personas
            planets: [], // Lista de planetas
            vehicles: [], // Lista de vehículos
            favorites: [], // Lista de favoritos
            showSplash: true, // Controla si se muestra la pantalla de inicio
        },
        actions: {
            // Obtiene personas desde la API y actualiza el estado
            fetchPeople: async () => {
                try {
                    const response = await fetch('https://www.swapi.tech/api/people');
                    if (!response.ok) throw new Error('Failed to fetch people');
                    const data = await response.json();

                    // Obtiene detalles adicionales para cada persona
                    const detailedPeople = await Promise.all(
                        data.results.map(async (person) => {
                            const detailResponse = await fetch(person.url);
                            if (!detailResponse.ok) throw new Error(`Failed to fetch details for ${person.name}`);
                            const detailData = await detailResponse.json();
                            return {
                                ...detailData.result.properties,
                                uid: person.uid,
                                type: 'person',
                            };
                        })
                    );
                    setStore({ people: detailedPeople }); // Actualiza el estado
                } catch (error) {
                    console.error('Error fetching people from SWAPI', error);
                }
            },
            // Obtiene planetas desde la API y actualiza el estado
            fetchPlanets: async () => {
                try {
                    const response = await fetch('https://www.swapi.tech/api/planets');
                    if (!response.ok) throw new Error('Failed to fetch planets');
                    const data = await response.json();
                }
            }
        }
    }
}

siguiente pagina
```

```
// Obtiene detalles adicionales para cada planeta
const detailedPlanets = await Promise.all(
    data.results.map(async (planet) => {
        const detailResponse = await fetch(planet.url);
        if (!detailResponse.ok) throw new Error(`Failed
            to fetch details for ${planet.name}`);
        const detailData = await detailResponse.json();
        return {
            ...detailData.result.properties,
            uid: planet.uid,
            type: 'planets',
        };
    })
);

setStore({ planets: detailedPlanets }); // Actualiza el estado
} catch (error) {
    console.error('Error fetching planets from SWAPI', error);
}
},
// Obtiene vehículos desde la API y actualiza el estado
fetchVehicles: async () => {
    try {
        const response = await fetch('https://www.swapi.tech/api/vehicles');
        if (!response.ok) throw new Error('Failed to fetch vehicles');
        const data = await response.json();

        // Obtiene detalles adicionales para cada vehículo
        const detailedVehicles = await Promise.all(
            data.results.map(async (vehicle) => {
                const detailResponse = await fetch(vehicle.url);
                if (!detailResponse.ok) throw new Error(`Failed to fetch details
                    for ${vehicle.name}`);
                const detailData = await detailResponse.json();
                return {
                    ...detailData.result.properties,
                    uid: vehicle.uid,
                    type: 'vehicles',
                };
            })
        );
        setStore({ vehicles: detailedVehicles }); // Actualiza el estado
    } catch (error) {
        console.error('Error fetching vehicles from SWAPI', error);
    }
},
// Oculta la pantalla de inicio
hideSplashScreen: () => {
    setStore({ showSplash: false });
},
// Añade un ítem a favoritos si no está presente
addFavorite: (item) => {
    const store = getStore();
    if (!store.favorites.some((fav) => fav.uid === item.uid)) {
        setStore({ favorites: [...store.favorites, item] });
    }
},
// Elimina un ítem de favoritos
removeFavorite: (item) => {
    const store = getStore();
    setStore({ favorites: store.favorites.filter((fav) => fav.uid !== item.uid) });
};

},
},
};

export default getState;
```

Pagination.js

Componente para manejar la paginación. Permite a los usuarios navegar entre páginas de datos.

```
● ● ●

import React from 'react';
import '../styles/index.css'; // Importa los estilos

// Componente de paginación
const Pagination = ({ currentPage, totalPages, onPageChange }) => {
    // Maneja el clic en el botón "Previous"
    const handlePrevClick = () => {
        if (currentPage > 1) { // Solo cambia a la página anterior si no estamos en la primera página
            onPageChange(currentPage - 1); // Llama a la función de cambio de página con la página anterior
        }
    };

    // Maneja el clic en el botón "Next"
    const handleNextClick = () => {
        if (currentPage < totalPages) { // Solo cambia a la página siguiente si no estamos en la última página
            onPageChange(currentPage + 1); // Llama a la función de cambio de página con la página siguiente
        }
    };

    return (
        <div className="pagination">
            {/* Botón para ir a la página anterior */}
            <button
                className="pagination-button"
                onClick={handlePrevClick}
                disabled={currentPage === 1} // Deshabilita el botón si estamos en la primera página
            >
                Previous
            </button>
            {/* Información de la página actual */}
            <span className="pagination-info">
                Page {currentPage} of {totalPages}
            </span>
            {/* Botón para ir a la página siguiente */}
            <button
                className="pagination-button"
                onClick={handleNextClick}
                disabled={currentPage === totalPages} // Deshabilita el botón si estamos en la última página
            >
                Next
            </button>
        </div>
    );
};

export default Pagination; // Exporta el componente
```

En este caso lo usaremos para poder navegar entre cards

SplashScreen.js

Muestra una pantalla de bienvenida animada que desaparece después de 3 segundos.

```
import React, { useEffect, useState, useContext } from 'react';
import { Context } from "../store/appContext";
import logo from '../img/star-wars.png';
import "../styles/index.css";

export const SplashScreen = () => {
    const { store, actions } = useContext(Context);
    const [visible, setVisible] = useState(true);
    const [lightsaberActive, setLightsaberActive] = useState(false);

    useEffect(() => {
        // Activar el sable láser después de un breve retraso para que se vea la animación
        const lightsaberTimer = setTimeout(() => {
            setLightsaberActive(true);
        }, 500);

        const hideTimer = setTimeout(() => {
            setVisible(false);
            actions.hideSplashScreen();
        }, 2500); // Aumentado a 2.5 segundos para dar tiempo a la animación del sable

        return () => {
            clearTimeout(lightsaberTimer);
            clearTimeout(hideTimer);
        };
    }, [actions]);

    return (
        <div className={`splash-screen ${visible ? 'visible' : 'hidden'}`}>
            <img src={logo} alt="Logo" className="splash-logo" />
            <div className="lightsaber-container">
                <div className={`lightsaber ${lightsaberActive ? 'active' : ''}`}></div>
            </div>
        </div>
    );
};


```

Footer.js

Pie de página con información de derechos de autor.

```
// Componente Footer como un componente funcional
export const Footer = () => (
    <footer className="footer mt-auto py-3 star-wars-footer"> /* Estilos y estructura del pie de página */
        <div className="footer-content"> /* Contenido principal del footer */
            <div className="footer-logo"> /* Contenedor para el logo */
                <img src={logo} width="150" height="auto"/> /* Logo de Star Wars */
            </div>
            <nav className="footer-nav"> /* Navegación del footer */
                <ul> /* Lista de enlaces */
                    <li><a href="#">Home </a></li>
                    <li><a href="#">Characters </a></li>
                    <li><a href="#">Movies</a></li>
                    <li><a href="#">Gallery</a></li>
                    <li><a href="#">Contact</a></li>
                </ul>
            </nav>
            <div className="social-icons"> /* Iconos de redes sociales enlazados */
                <a href="#" className="fab fa-facebook"></a>
                <a href="#" className="fab fa-twitter"></a>
                <a href="#" className="fab fa-instagram"></a>
                <a href="#" className="fab fa-youtube"></a>
            </div>
        </div>
        <div className="footer-bottom"> /* Información adicional al pie del footer */
            <p>© 2023 Star Wars. All rights reserved. May the Force be with you.</p> /* Mensaje de derechos reservados */
        </div>
    </footer>
);


```

Navbar.js

Barra de navegación fija que permite al usuario acceder a diferentes secciones de la aplicación.

```
● ● ●

import React, { useState } from "react"; // Importa React y useState
import logo from "../img/star-wars.png"; // Importa el logo
import { Link } from "react-router-dom"; // Importa Link para navegación
import "../styles/index.css"; // Importa los estilos
import soldado from "../img/soldado.png"; // Importa la imagen del botón de favoritos
import { Favorites } from "./favorites"; // Importa el componente Favorites

export const Navbar = () => {
  const [showFavorites, setShowFavorites] = useState(false); // Estado para
  mostrar/ocultar favoritos

  return (
    <nav className="navbar dark">
      <Link to="/" className="navbar-logo"> /* Enlace a la página de inicio */
        <img
          src={logo} // Logo de Star Wars
          width="200"
          height="auto"
          className=""
        />
      </Link>
      <div className="ml-auto">
        /* Botón para mostrar/ocultar el componente de favoritos */
        <button className="fav-button" onClick={() =>
          setShowFavorites(!showFavorites)}
          Favorites <img src={soldado} width="25" height="auto" className="" />
        </button>
      </div>
      /* Muestra el componente Favorites si showFavorites es verdadero */
      {showFavorites && <Favorites onClose={() => setShowFavorites(false)} />}
    </nav>
  );
};


```

ScrollToTop.js

Este componente asegura de que la página se desplace a la parte superior cada vez que cambiar de ruta.

```
import React from "react"; // Importa React
import PropTypes from "prop-types"; // Importa PropTypes para validación de propiedades

// Componente ScrollToTop como un componente de clase
class ScrollToTop extends React.Component {
  // Método de ciclo de vida que se llama después de que el componente se actualiza
  componentDidUpdate(prevProps) {
    // Comprueba si la ubicación ha cambiado
    if (this.props.location !== prevProps.location) {
      // Desplaza la página a la parte superior
      window.scrollTo(0, 0);
    }
  }

  // Renderiza los hijos del componente
  render() {
    return this.props.children;
  }
}

export default ScrollToTop; // Exporta el componente

// Definición de las propiedades esperadas
ScrollToTop.propTypes = {
  location: PropTypes.object, // Propiedad opcional para ubicación (utilizada para comparar cambios de ruta)
  children: PropTypes.any // Propiedad para los elementos hijos que el componente renderiza
};
```

Card.js

Componente para representar un elemento de la colección, ya sea un personaje, planeta o vehículo.

```
import React, { useEffect, useContext, useRef } from "react"; // Importa React y hooks necesarios
import { Link } from 'react-router-dom'; // Importa Link para navegación
import { Context } from "../store/appContext";
import "../../styles/index.css"; // Importa estilos
import 'bootstrap/dist/css/bootstrap.min.css'; // Importa Bootstrap
import soldado from "../../img/soldado.png"; // Importa imagen para el botón de favoritos

// Componente funcional Card
export const Card = ({ type }) => {
  const { store, actions } = useContext(Context); // Obtiene el estado y acciones del contexto
  const cardContainerRef = useRef(null); // Referencia al contenedor de las tarjetas

  // Efecto para cargar datos según el tipo de tarjeta
  useEffect(() => {
    if (type === "people" && store.people.length === 0) {
      actions.fetchPeople(); // Carga personas si están vacías
    } else if (type === "planets" && store.planets.length === 0) {
      actions.fetchPlanets(); // Carga planetas si están vacíos
    } else if (type === "vehicles" && store.vehicles.length === 0) {
      actions.fetchVehicles(); // Carga vehículos si están vacíos
    }
  }, [type, actions, store.people.length, store.planets.length, store.vehicles.length]);

  const items = store[type] || []; // Obtiene los elementos según el tipo

  // Función para obtener la URL de la imagen según el tipo
  const getImageUrl = (item) => {
    switch(type) {
      case 'people':
        return `https://starwars-visualguide.com/assets/img/characters/${item.uid}.jpg`;
      case 'planets':
        return `https://starwars-visualguide.com/assets/img/planets/${item.uid}.jpg`;
      case 'vehicles':
        return `https://starwars-visualguide.com/assets/img/vehicles/${item.uid}.jpg`;
      default:
        return 'https://starwars-visualguide.com/assets/img/placeholder.jpg';
    }
  };

  // Función para obtener atributos específicos según el tipo
  const getAttributes = (item) => {
    switch (type) {
      case 'people':
        return `Gender: ${item.gender}, Hair Color: ${item.hair_color}, Eye Color: ${item.eye_color}`;
      case 'planets':
        return `Population: ${item.population}, Terrain: ${item.terrain}`;
      case 'vehicles':
        return `Model: ${item.model}, Manufacturer: ${item.manufacturer}`;
      default:
        return null;
    }
  };

  // Función para verificar si un ítem es favorito
  const isFavorite = (item) => {
    switch (type) {
      case 'people':
        return store.favorites.some(fav => fav.type === 'people' && fav.uid === item.uid);
      case 'planets':
        return store.favorites.some(fav => fav.type === 'planets' && fav.uid === item.uid);
      case 'vehicles':
        return store.favorites.some(fav => fav.type === 'vehicles' && fav.uid === item.uid);
      default:
        return false;
    }
  };
}
```

```

// Función para alternar el estado de favorito
const toggleFavorite = (item) => {
  if (isFavorite(item)) {
    actions.removeFavorite(item);
  } else {
    actions.addFavorite(item);
  }
};

// Funciones para desplazar horizontalmente el contenedor de tarjetas
const scrollLeft = () => {
  cardContainerRef.current.scrollBy({ left: -300, behavior: 'smooth' });
};

const scrollRight = () => {
  cardContainerRef.current.scrollBy({ left: 300, behavior: 'smooth' });
};

return (
  <div className="carousel-container">
    <button className="scroll-button left" onClick={scrollLeft}>&lt;</button>
    <div className="card-wrap" ref={cardContainerRef}>
      {items.map((item) => {
        const isFavorite = store.favorites.some(fav => fav.uid === item.uid);

        return (
          <div className="card" key={item.uid}>
            <img
              src={getImageUrl(item)} // URL de la imagen
              className="card-img-top"
              alt={item.name}
              onError={(e) => { e.target.onerror = null; e.target.src = "https://starwars-visualguide.com/assets/img/placeholder.jpg" }} // Imagen de reemplazo si ocurre un error
            />
            <div className="card-body">
              <h2 className="card-title text-warning">{item.name}</h2>
              <p className="text-light">{getAttributes(item)}</p>
              <Link to={`/${type}/${item.uid}`}>
                <button className="learn-more-button me-3">Learn more</button>
              </Link>
              <button
                className={`fav-button ${isFavorite ? 'active' : ''}`}
                onClick={() => toggleFavorite(item)} // Alterna el estado de favorito
              >
                <img src={soldado} width="20" height="auto" alt="Favorite" />
              </button>
            </div>
          </div>
        );
      })
    </div>
    <button className="scroll-button right" onClick={scrollRight}>&gt;</button>
  </div>
);
};

```

Favorites.js

Muestra una lista de elementos favoritos guardados por el usuario.

```

import React, { useContext } from 'react'; // Importa React y useContext
import { Context } from '../store/appContext'; // Importa el contexto de la aplicación
import '../.../styles/index.css'; // Importa estilos
import soldado from '../.../img/soldado.png'; // Importa imagen para el botón de eliminar favoritos

// Componente funcional Favorites
export const Favorites = () => {
  const { store, actions } = useContext(Context); // Obtiene el estado y las acciones del contexto

  return (
    <div className="favorites-container">
      <h4>Favorites</h4> /* Título de la sección de favoritos */
      {store.favorites.length > 0 ? ( // Verifica si hay favoritos
        store.favorites.map((item, index) =>
          <div key={index} className="favorites-item"> /* Renderiza cada ítem favorito */
            <img src={soldado} alt={item.name} /> /* Imagen representativa */
            <span>{item.name}</span> /* Nombre del ítem */
            <button onClick={() => actions.removeFavorite(item)}>X</button> /* Botón para eliminar el ítem de favoritos */
          </div>
        )
      ) : (
        <p>No favorites added yet</p> /* Mensaje si no hay favoritos */
      )
    </div>
  );
};

```

Cardinfo.js

Muestra detalles específicos de un personaje, planeta o vehículo seleccionado.

```
import React, { useEffect, useState, useContext } from "react"; // Importa React y hooks necesarios
import { useParams } from "react-router-dom"; // Importa useParams para obtener parámetros de la URL
import "../styles/index.css"; // Importa estilos
import soldado from "../../img/soldado.png"; // Importa imagen de marcador de favoritos
import { Context } from "../../store/appContext"; // Importa el contexto de la aplicación

export const Cardinfo = () => {
  const { uid, type } = useParams(); // Obtiene uid y type de la URL
  const { actions, store } = useContext(Context); // Obtiene el estado y las acciones del contexto
  const [item, setItem] = useState(null); // Estado para almacenar el ítem cargado
  const [description, setDescription] = useState("Loading..."); // Estado para la descripción del ítem
  const [imageLoaded, setImageLoaded] = useState(false); // Estado para controlar la carga de imagen
  // Función para mapear el tipo de ítem a la categoría de la imagen
  const getImageCategory = (type) => {
    const categoryMap = {
      'people': 'characters',
      'planets': 'planets',
      'vehicles': 'vehicles',
      'starships': 'starships',
      'species': 'species'
    };
    return categoryMap[type] || type;
  };
  useEffect(() => {
    // Efecto para obtener los datos del ítem cuando cambian uid o type
    const fetchItem = async () => {
      try {
        const response = await fetch(`https://www.swapi.tech/api/${type}/${uid}`);
        if (!response.ok) {
          throw new Error('Failed to fetch data');
        }
        const data = await response.json();
        setItem(data.result.properties);
        setDescription(data.result.description);
      } catch (error) {
        console.error('Error fetching data from SWAPI', error);
        setDescription("Failed to load description.");
      }
    };
    fetchItem();
  }, [uid, type]); // Dependencias: uid y type
  // Función para añadir el ítem a favoritos
  const addToFavorites = () => {
    const itemToAdd = { ...item, uid: uid, type: type };
    actions.addFavorite(itemToAdd);
  };
  // Función para eliminar el ítem de favoritos
  const removeFromFavorites = () => {
    const itemToRemove = { ...item, uid: uid, type: type };
    actions.removeFavorite(itemToRemove);
  };
  // Si el ítem no está cargado, muestra un mensaje de carga
  if (!item) {
    return <div>Loading...</div>;
  }
  // Función para renderizar los detalles del ítem
  const renderDetails = () => {
    return Object.entries(item).map(([key, value]) => {
      if (key === 'name' || key === 'url') return null; // Excluye 'name' y 'url' del renderizado
      const formattedKey = key.split('_').map(word => word.charAt(0).toUpperCase() + word.slice(1)).join(' ');
      if (Array.isArray(value)) {
        value = value.join(', ');
      } // Une arrays en un string
      return (
        <div key={key} className="attribute">
          <strong>{formattedKey}</strong> {value}
        </div>
      );
    });
  };
  const imageUrl = `https://starwars-visualguide.com/assets/img/${getImageCategory(type)}/${uid}.jpg`; // URL de la imagen del ítem
  // Determina si el ítem está en favoritos
  const isInFavorites = store.favorites.some(fav => fav.uid === item.uid);

  return (
    <div className="container mt-3 card-info-container">
      <div className="row">
        <div className="col-md-6">
          {!imageLoaded && <div>Loading image...</div>} /* Mensaje de carga de imagen */
          <img
            src={imageUrl}
            alt={item.name}
            className={`${`img-fluid rounded ${imageLoaded ? '' : 'd-none'}`}`}
            onLoad={() => setImageLoaded(true)} // Marca la imagen como cargada
            onError={(e) => {
              console.error('Failed to load image: ${imageUrl}');
              e.target.onerror = null;
              e.target.src = "https://starwars-visualguide.com/assets/img/placeholder.jpg"; // Imagen de reemplazo en caso de error
              setImageLoaded(true);
            }}
          />
        </div>
        <div className="col-md-6 d-flex flex-column justify-content-between">
          <div>
            <h2>{item.name}</h2> /* Nombre del ítem */
            <p>{description}</p> /* Descripción del ítem */
          </div>
          <div>
            <h3>Attributes</h3>
            <div className="attributes-container">
              {renderDetails()} /* Renderiza los detalles del ítem */
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};
```