

HW9 | React Routing - Integration

DURACIÓN ESTIMADA

90 minutos

RICK AND MORTY APP

INTRODUCCIÓN

En esta homework integraremos **React Router DOM** para enrutar las distintas vistas de nuestra aplicación. Esto quiere decir que podremos decidir en que path o "link" se renderice cada componente.



INSTRUCCIONES

EJERCICIO 1 | Instalación y configuración

1. Instala **react-router-dom** desde la terminal.
2. Una vez hecho esto, dirígete al archivo **index.js** e importa y envuelve toda tu aplicación con **"BrowserRouter"**.
3. Importa los componentes **"Routes"** y **"Route"** de *react-router-dom* en tu archivo **App.js**.

EJERCICIO 2 | About

Ahora crearemos un componente para presentar nuestro perfil. Crea un componente llamado **About**. Este componente será una vista que contenga tu información.

Esto es completamente libre. Puedes mostrar incluso una imagen tuya. Esto le servirá a las personas que vean tu App para conocer al creador  .

Una vez construido el componente:

1. Dirígete al componente **Nav** e importa la etiqueta **Link**.
2. Crea dos botones. Uno con el texto **"About"** y que te redirija a **/about**, y otro con el texto **"Home"** que te redirija a **/home**.

[NOTA]: podrías utilizar NavLink para darle estilos al link About y Home.

EJERCICIO 3 | Routing

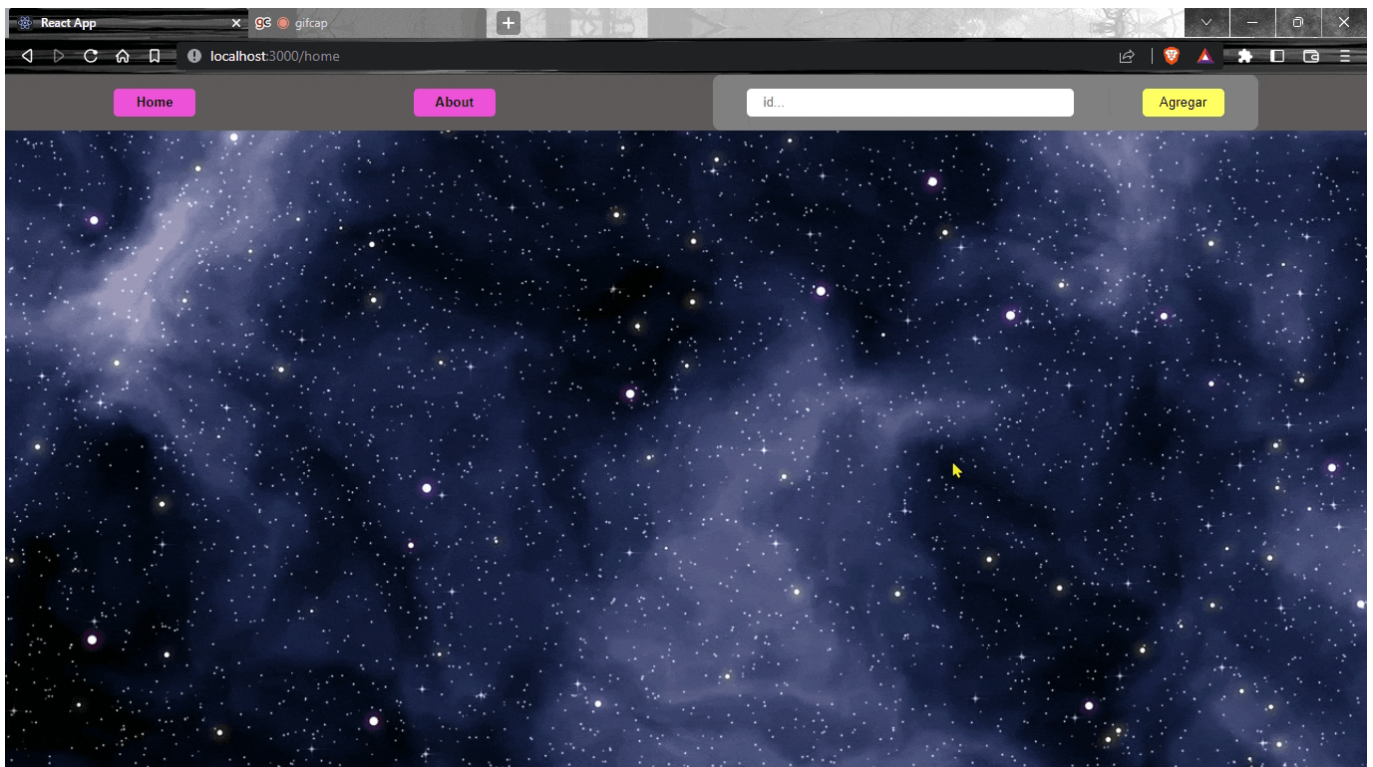
Comenzaremos creando un componente llamador **Deatil** el cual solo mostrara una etiqueta **div** vacía.

Luego, dirígete al archivo **App.js**. Ahora crearemos las rutas de los componentes. Cada componente debe renderizarse en las siguientes rutas:

- **Nav** debe que aparecer en todas las rutas.
- **Cards** debe aparecer solo en la ruta **/home**.
- **About** debe aparecer solo en la ruta **/about**.
- **Detail** debe aparecer solo en la ruta **/detail/:id**.

[NOTA]: ten en cuenta que la ruta del componente **Detail** recibe un parámetro **id**.

Comprueba en tu navegador que cada componente se renderice en la ruta indicada. Debería quedarte de esta manera:



EJERCICIO 4 | Detail redirection

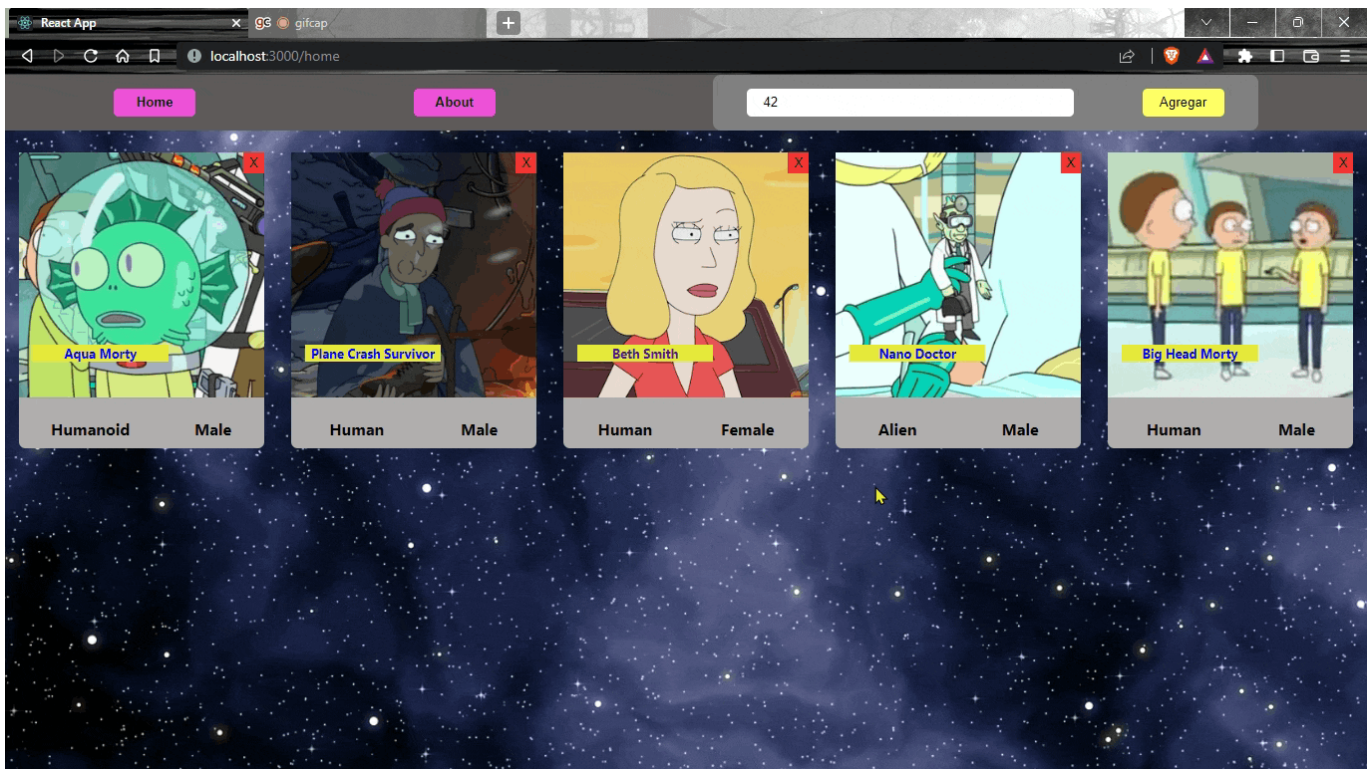
En este ejercicio te encargarás de crear la redirección hacia el *detail* de un personaje. Para esto:

1. Dirígete al componente **Card** e importa la etiqueta **Link**.
2. Envuelve el nombre del personaje en esta etiqueta, y que redirija a la ruta **/detail/:id**.

[NOTA]: debes pasarle como parámetro el **id** del personaje. personaje para usarlo en el Link.

```
// Card.js
...
<Link to={`/detail/${id}`} >
  <h3 className="card-name">{name}</h3>
</Link>
...
```

En este momento, cuando hacemos click sobre el nombre de un personaje nos debe redirección a la ruta especificada.



EJERCICIO 5 | Detail

¡Genial! Las funcionalidades ya están. Ahora es momento de contruir nuestro componente **Detail**. Para esto dirígete a este componente y:

1. Importa **axios**.
2. Importa el hook **useParams** y obten el **id** del personaje.
3. Importa el hook **useState** y crea un estado local con el nombre "**character**" que se inicialice como un objeto vacío.
4. En este paso importaremos el hook **useEffect** de **react**. Una vez importado, copia el siguiente código y pégalo en el cuerpo del componente.

```
useEffect(() => {
  axios(`https://rickandmortyapi.com/api/character/${id}`).then(({ data }) => {
    if (data.name) {
      setCharacter(data);
    } else {
      window.alert('No hay personajes con ese ID');
    }
  });
  return setCharacter({});
}, [id]);
```

[NOTA]: este código es el que buscará al personaje de la API cada vez que el componente se monte. Y luego, cada vez que se desmonte, borrará su información.

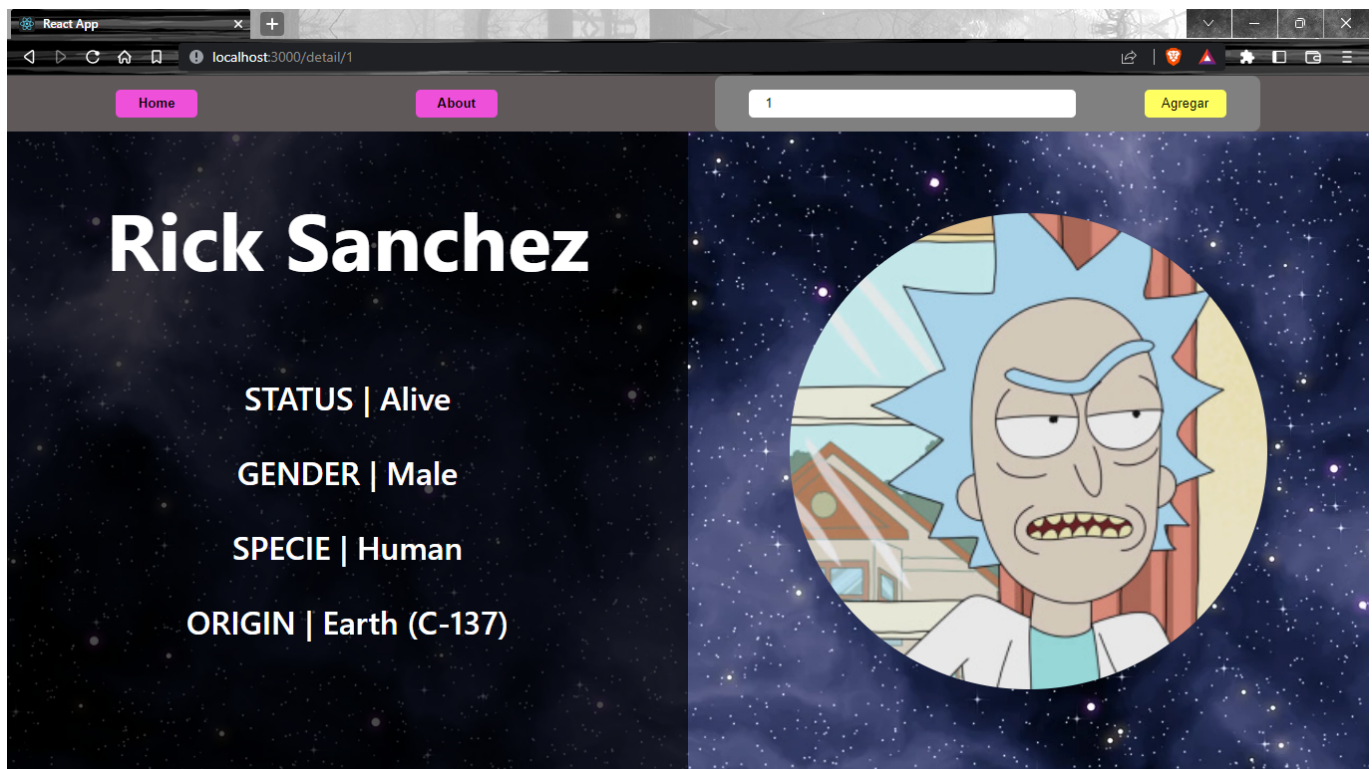
EJERCICIO 6 | Detail rendering

Ahora en el estado local **character** ya tenemos disponible toda la información que necesitamos de nuestro personaje. Por lo que:

1. Renderiza **condicionalmente** cada una de estas propiedades.

- **name**
- **status**
- **species**
- **gender**
- **origin** (ten en cuenta que el nombre se guarda dentro de otra propiedad "name")
- **image**

Debería quedarte algo como esto:



[NOTA]: como la información del personaje se obtiene a partir de una petición asincrónica a la API de Rick & Morty, puede que la información aún no esté disponible cuando la quieras renderizar. ¡Aquí es donde debes aplicar renderizado condicional! Te dejamos la [documentación](#) como ejemplo.

EXTRA CREDIT

Ahora te desafiamos a que crees un nuevo componente llamado **Error**. A este componente le podrás dar los estilos que quieras, pero la idea es que se muestre un mensaje de error 404. ¡Puedes inspirarte en este [ejemplo](#)!

El desafío es el siguiente: haz que este componente se muestre cada vez que el usuario ingrese a cualquier otra ruta que no exista. Es decir que no la hayas especificado en esta homework. Por ejemplo, si creaste una ruta `"/home"` y `"/about"`, y el usuario en el navegador escribe `"/henry"`, debería mostrar el componente Error 404.