

# HW12 | React Redux - Integration

---

## DURACIÓN ESTIMADA

2 horas

---

## RICK AND MORTY APP

## INTRODUCCIÓN

En esta homework crearemos un nuevo espacio para guardar a nuestros personajes favoritos. ¡Podremos agregarlos y eliminarlos!

---

## INSTRUCCIONES

### EJERCICIO 1 | Redux & Configuration

Para comenzar, tendrás que instalar las dependencias **redux**, **react-redux** y **redux-thunk** con el comando:

```
npm i redux react-redux redux-thunk
```

Ahora, dentro de la carpeta **src** crea una nueva carpeta llamada **redux**. Dentro de esta crea los archivos **actions.js**, **store.js** y **reducer.js**.

Dentro del archivo **store.js** haz la configuración del store. Una vez configurado, deberás importarlo en tu archivo **index.js** junto con la etiqueta **Provider** y envolver tu aplicación a con estos elementos.

---

### EJERCICIO 2 | Actions

1. Construye dos *actions-creators*:

- **addFav**: esta función recibe un personaje por parámetro. Deberás retornar una action con el **type** igual a **"ADD\_FAV"**, y el payload igual a ese personaje.
- **removeFav**: esta función recibe un **id** por parámetro. Deberás retornar una action con el **type** igual a **"REMOVE\_FAV"**, y el payload igual a ese id.

2. Exporta ambas funciones.

**[NOTA]:** no olvides que el nombre que asignes en la propiedad "TYPE" de tu acción, debe coincidir exactamente con el nombre de los casos que hayas asignado en tu reducer.



### EJERCICIO 3 | Reducer

Dirígete a tu archivo `reducer.js`. Allí deberás:

1. Crear un **initialState** con una propiedad llamada **"myFavorites"**. Esta propiedad será un arreglo vacío.
2. Luego deberás crear tu reducer. Recuerda que este recibe dos parámetros y que dentro de él hay un switch.

**[NOTA]:** ten en cuenta el modo en el que lo exportas, y cómo lo importas dentro de tu store.

3. Crea un primer caso llamado **"ADD\_FAV"** en el que puedas agregar a un personaje que recibes por payload a tu estado **"myFavorites"**.
4. Crea otro caso llamado **"REMOVE\_FAV"** en el que puedas eliminar a un personaje de tu estado **"myFavorites"** a partir de un **id** que recibas por payload.

**[NOTA]:** ten en cuenta que el **id** que recibes por payload es un string, y el **id** de los personajes es un número.

5. No te olvides de tu caso **default**.



### EJERCICIO 4 | Fav button

¡Ahora crearemos un botón para agregar y eliminar a nuestros personajes de los favoritos!

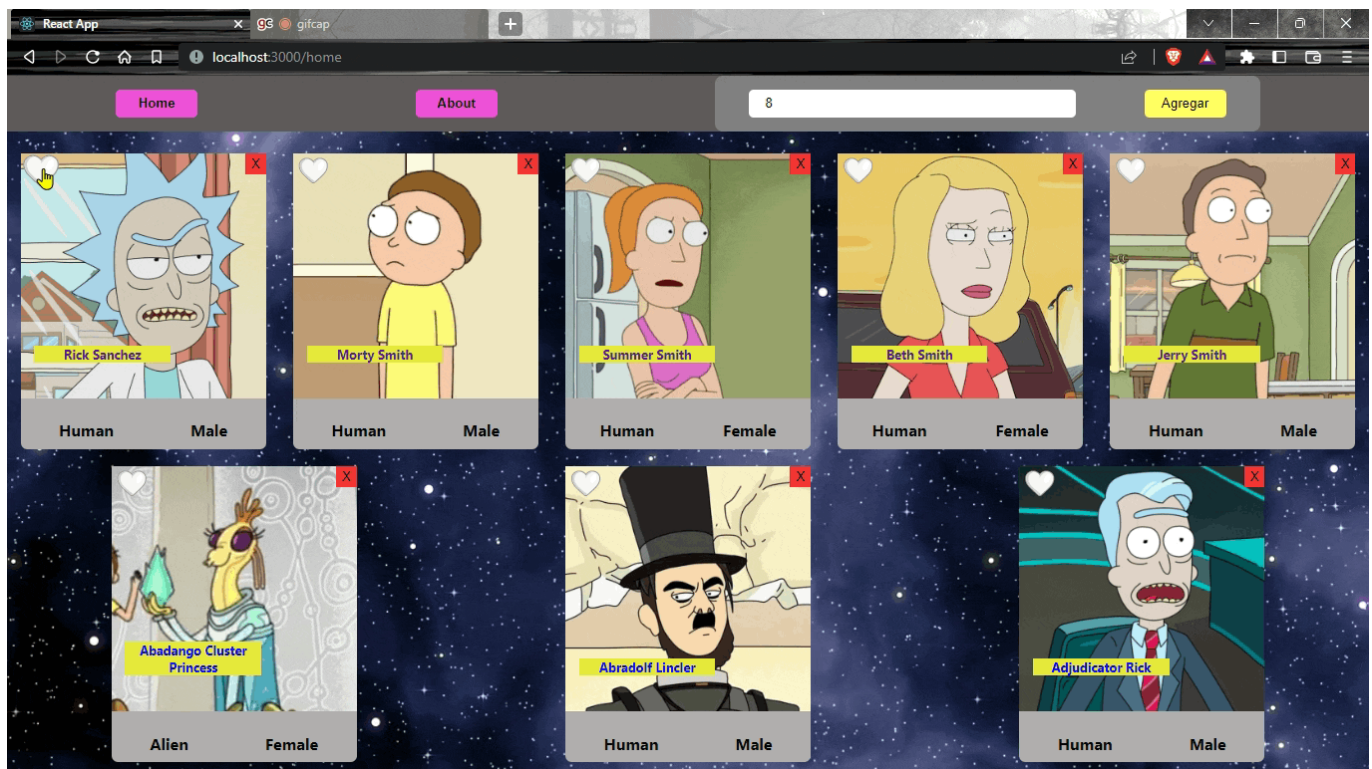
1. Dirígete al componente **Card**. Aquí deberás crear una función **mapDispatchToProps** que contenga dos funciones. Esta debe despachar las dos actions que creaste anteriormente (por lo que deberás importarlas).
2. Conecta esta función con tu componente y recibe ambas funciones despachadoras por props.
3. Crea un estado local en tu componente con el nombre **isFav** e inicialízalo en **false**.
4. Crea una función en el cuerpo del componente llamada **handleFavorite**. Esta función estará dividida en dos partes:
  - Si el estado **isFav** es **true**, entonces settea ese estado en false, y despacha la función **removeFav** que recibiste por props pasándole el **id** del personaje como argumento.
  - Si el estado **isFav** es **false**, entonces settea ese estado en true, y despacha la función **addFav** que recibiste por props, pasándole **props** como argumento.
5. Ahora te ayudaremos a crear un renderizado condicional. Si tu estado local **isFav** es true, entonces se mostrará un botón. Si es false, se mostrará otro botón. Para esto, copia y pega el siguiente código al comienzo del renderizado de tu componente (no te olvides de darle estilos).

```

{
  isFav ? (
    <button onClick={handleFavorite}>♥</button>
  ) : (
    <button onClick={handleFavorite}>♥</button>
  )
}

```

En este punto debería quedarte algo como esto:



- Una vez hecho esto, nos tenemos que asegurar que el status de nuestro estado local se mantenga aunque nos vayamos y volvamos al componente. Para esto vamos a agregar una función **mapStateToProps**. Esa función debe traer nuestro estado global **myFavorites**. Finalmente recíbelo por **props** dentro de tu componente.
- Este **useEffect** comprobará si el personaje que contiene la **Card** ya está dentro de tus favoritos. En ese caso seteará el estado **isFav** en true. Cópialo y pégalo dentro de tu componente (no te olvides de importar este hook).

```

useEffect(() => {
  myFavorites.forEach((fav) => {
    if (fav.id === props.id) {
      setIsFav(true);
    }
  });
}, [myFavorites]);

```

**DESAFÍO:** te desafiamos a que reconstruyas ese `useEffect`, pero utilizando un **bucle For** en lugar de un `.forEach()`.

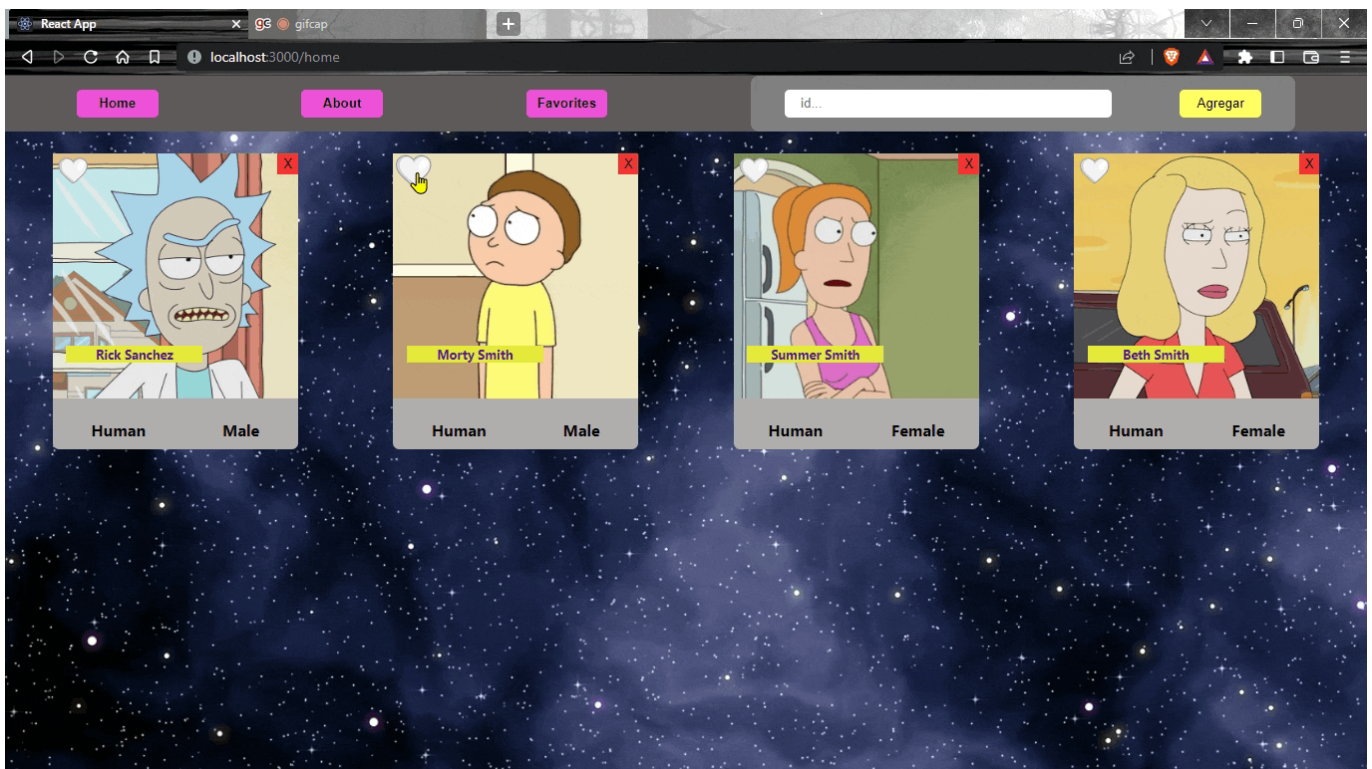
## EJERCICIO 5 | Favorites

Dirígete a tu carpeta de componentes y crea uno llamado **Favorites**.

1. Crea una ruta en el archivo **App.js** para mostrar este componente. El path de la ruta debe ser **/favorites**.
2. Crea un botón en tu **Nav** con el texto "Favorites" que te redirija a esta ruta.
3. Dentro de tu componente **Favorites** crea una función **mapStateToProps**. Esta función debe traer el estado global **myFavorites**. Conecta el componente con la función, y recibe el estado global por props.
4. Una vez que tengas la lista de tus personajes favoritos dentro de tu componente, deberás mapearlo y renderizar una **Card** con información del personaje (no te olvides de pasarle las propiedades del personaje).


## ¡LISTO! YA FUNCIONA TODO

Todo el trabajo que hiciste en esta integración debería darte un resultado y funcionamiento similar a este:



## EJERCICIO EXTRA

¡Ahora te proponemos un desafío!

Si revisas, esta aplicación tiene un pequeño bug que tendrás que resolver... cuando presionas el  de una de las Cards el personaje aparece en la vista de "**Favoritos**". Pero si luego eliminas el personaje presionando en la X, este aún permanece en esa vista. Busca la manera para que cuando elimines un personaje, también se elimine de "**Favoritos**".