

Sprint 4 – Ejercicios

Índice

Índice	1
Objetivo	2
Comentarios generales	2
Ejercicio 1	3
Ejercicio 2	4
Ejercicio 3	5
Ejercicio 4	6
Ejercicio 5	7
Ejercicio 6	8
Ejercicio 7	9
Ejercicio 8	11
Ejercicio 9	12
Ejercicio 10	13
Ejercicio 11	14
Ejercicio 12	14
Ejercicio 13	14
Ejercicio 14	15
Ejercicio 15	15
Ejercicio 16	16
Ejercicio 17	17
Ejercicio 18	18
Ejercicio 19	20
Ejercicio 20	21

Objetivo

El objetivo de este Sprint es aprender sobre:

- React (JSX, Props, State, Componentes).
- Ciclos de Vida.
- Hooks.
- Event Listeners.
- Forms
- HTTP Requests.
- React Router.
- Redux.

Comentarios generales

- Leer en detalle la pauta de cada ejercicio.
- Notar que algunos ejercicios requieren que sea haya dictado una clase previa (teórico) antes de poder resolverlos. Estos ejercicios estarán debidamente señalizados.
- En caso de dudas, pueden recurrir a sus compañeros, docentes (por Slack) y/o sitios en Internet (ej: Stack Overflow). Recuerden la importancia de apoyarse entre ustedes ya que una gran forma de aprender y reforzar conocimientos es explicarle a otro.
- También recomendamos tener una carpeta llamada **ha_bootcamp_sprint4** (o similar) para tener todos los ejercicios de este sprint juntos.

Ejercicio 1

Clase previa: “Introducción a React”.

Pauta:

1. Crear una app utilizando `create-react-app`.
2. Cambiar el título de la aplicación (pestaña en el navegador) a “Mi primer aplicación React”.
3. Borrar todo el contenido que está dentro del primer `div` del componente `App` en `src/App.js`.
4. Crear una carpeta llamada `components` dentro de `src/` y crear un archivo llamado `Welcome.js` dentro de ella.
5. Dentro de este archivo declarar un componente que reciba una *prop* llamada `name` y un mensaje de bienvenida dentro de un `<h1>`. Por ejemplo: si el valor de la *prop* `name` es “María Pérez”, el componente tiene que mostrar una etiqueta `h1` con el texto “¡Bienvenido/a a React, María Pérez!”.
6. Exportar como `default` el componente `Welcome`.
7. Importar el componente `Welcome` desde el componente `App`, y utilizarlo dentro del `div`, ahora vacío, para mostrar un mensaje de bienvenida.
8. Probar distintos valores en la variable `name` y ver como al guardar se actualizan los cambios sin necesidad de refrescar la página.

Extra: Analizar la estructura de archivos y carpetas generadas automáticamente. Identificar archivos desconocidos y discutirlos en clase.

Ejercicio 2

Clase previa: “Introducción a React”.

Pauta:

1. Dentro de una app generada con `create-react-app`, crear cuatro componentes separados:
 - a. `Title`, el cual debe renderizar un elemento `h1` con un título.
 - b. `Subtitle`, el cual debe renderizar un elemento `h2` con un subtítulo.
 - c. `Description`, el cual debe renderizar un elemento `p` que muestre “Mi nombre es ” + `nombre`, siendo `nombre` una variable definida externamente.
2. Borrar el contenido dentro del primer `div` en `App.js`, y en sustituirlo por los tres componentes anteriores, en el orden indicado.

El HTML resultante en el navegador debería ser algo similar a:

```
<div id="root">
  <div class="App">
    <h1>Hola</h1>
    <h2>Subtitulo</h2>
    <p>Mi nombre es María</p>
  </div>
</div>
```

Extra: Investigar sobre buenas prácticas de estructuración de componentes en React, y crear estos archivos de la forma más escalable (por ejemplo, pueden empezar con esta [guía](#)).

Ejercicio 3

Clase previa: “Introducción a React”.

Pauta:

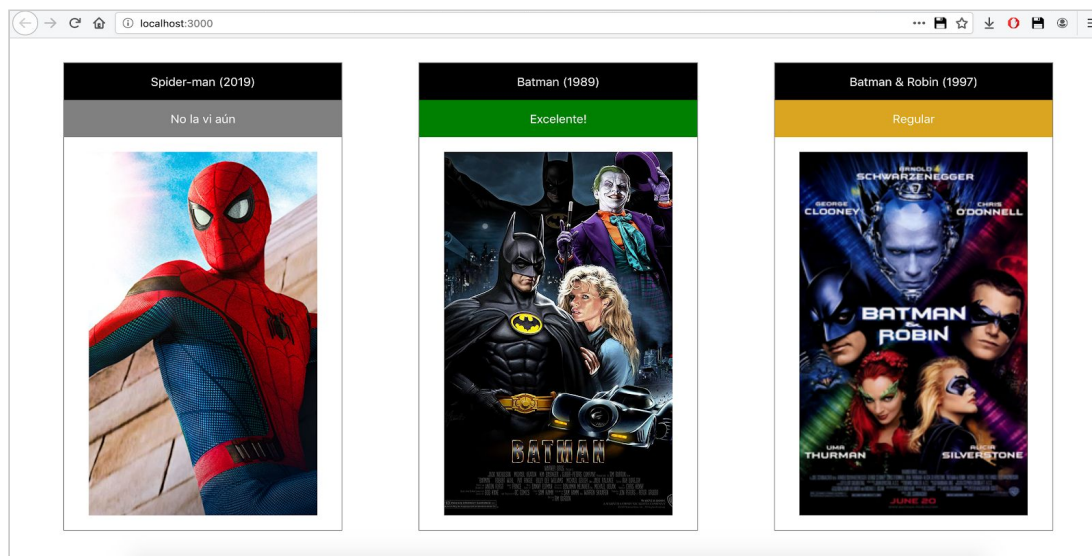
1. Crear un componente `Persona`.
2. Crear las variables `nombre` y `edad` dentro del componente.
3. Renderizar `nombre` dentro de un elemento `<p>`.
4. Agregar otro párrafo `<p>` que muestre el mensaje *“Lo sentimos, no tiene edad legal para beber alcohol”* si la `edad` es menor a 18, o que muestre *“Bienvenido. Lo invitamos a tomar una cerveza”* si la edad es mayor o igual que 18.
5. Modificar el valor de la variable `edad` y ver cómo se actualiza el texto mostrado en la *app*.

Ejercicio 4

Clase previa: “Introducción a React”.

Pauta:

Crear un sitio web usando React, cuyo aspecto final sea como el siguiente:



De cada película se debe conocer: nombre, año, imagen y puntaje (“bueno”, “regular”, “malo”, o que la película no se haya visto aún).

El alumno deberá inventar los datos de las 3 películas. No es necesario utilizar exactamente las mismas imágenes del diagrama, ni siquiera es necesario crear las mismas tres películas, pero en caso de querer hacerlo, pueden usar los siguientes links: [Spider-man](#), [Batman](#) y [Batman & Robin](#).

Intentar deducir los requerimientos y diseñar una solución a partir de lo que ve en el diagrama anterior. Por ejemplo, hacerse preguntas como:

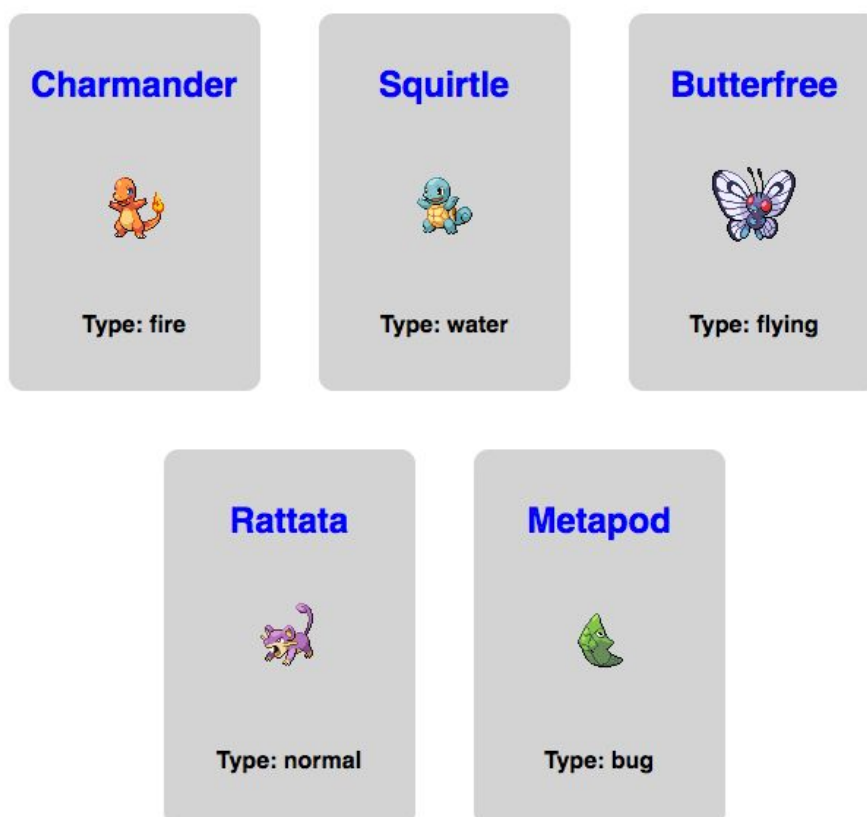
- ¿Qué componentes se deberían crear?
- ¿Qué *props* (datos) hay que pasarle a dichos componentes?

Ejercicio 5

Clase previa: “List Rendering”.

Pauta:

1. Crear una aplicación web que muestre una **lista** con los **Pokemons** presentes en [este archivo](#) (que se deberá descargar y colocar dentro del proyecto).
2. Se deberá crear un componente llamado `Pokemon`. La información deberá ser pasada por *props*.
3. El resultado final debería ser algo así:



Ejercicio 6

Clase previa: “Formularios”.

Pauta:

Crear un formulario HTML que contenga dos campos:

- Número 1.
- Número 2.

El formulario debe tener un botón llamado “**Multiplicar**”. Al hacer click sobre el mismo se debe calcular la multiplicación de ambos números y mostrar el resultado debajo del botón.

Si alguno de los campos está vacío, mostrar el mensaje “*Por favor complete todos los campos*”.

Ingresa dos números para multiplicar

Número 1

Número 2

Multiplicar

El resultado es: 24

Ejercicio 7

Clase previa: “Event listeners”.

Pauta:

Crear una aplicación que consista en dos columnas. La de la izquierda deberá mostrar un listado con todos los productos que comercializa una tienda (disponible en [este archivo](#)). La de la derecha deberá mostrar un “**carrito de compras**”, el cual se compone de los productos seleccionados por el usuario.

Cada vez que se hace click sobre un elemento de la lista izquierda, el mismo se agrega a la lista de la derecha (carrito) y se coloca su cantidad en 1. Si se hace click sobre un elemento que ya había sido agregado al carrito, su cantidad se deberá incrementar en una unidad.

Al hacer click sobre un elemento del carrito, su cantidad se deberá decrementar en una unidad. Al llegar a cero, el elemento se remueve completamente de la lista.

Productos disponibles	Carrito de compras
<input type="radio"/> Frutilla	<input type="radio"/> Manzana (Cantidad: 1)
<input type="radio"/> Uva	<input type="radio"/> Banana (Cantidad: 2)
<input type="radio"/> Naranja	<input type="radio"/> Naranja (Cantidad: 3)
<input type="radio"/> Banana	<input type="radio"/> Zanahoria (Cantidad: 3)
<input type="radio"/> Manzana	
<input type="radio"/> Zanahoria	
<input type="radio"/> Puerro	
<input type="radio"/> Champiñon	
<input type="radio"/> Pan	

Nota: Pensar bien los **componentes** que se deberán crear.

Extra 1: Agregar la funcionalidad de que si se agregan más de 5 unidades de Papel Higiénico o Alcohol en Gel, se muestre un mensaje abajo del carrito que diga: *“Lo sentimos. No es posible comprar más unidades. Otras familias también necesitan abastecerse”*.

Extra 2: Agregar la funcionalidad de mostrar el precio de cada producto y el costo total de los productos del carrito.

1. En ambas listas (izquierda y derecha), agregar el precio unitario de cada producto.
2. Agregar el precio total de los productos del carrito.

Productos disponibles	Carrito de compras
<div>⊕ Frutilla (\$50.3 c/u)</div>	<div>⊖ Banana (Cant: 1) (\$55.9 c/u)</div>
<div>⊕ Uva (\$23 c/u)</div>	<div>⊖ Puerro (Cant: 1) (\$30 c/u)</div>
<div>⊕ Naranja (\$76 c/u)</div>	<div>⊖ Naranja (Cant: 8) (\$76 c/u)</div>
<div>⊕ Banana (\$55.9 c/u)</div>	<div>⊖ Uva (Cant: 4) (\$23 c/u)</div>
<div>⊕ Manzana (\$31 c/u)</div>	<div>⊖ Huevo (Cant: 1) (\$29 c/u)</div>
<div>⊕ Zanahoria (\$17.6 c/u)</div>	
<div>⊕ Puerro (\$30 c/u)</div>	
<div>⊕ Champiñon (\$80 c/u)</div>	
	<div>Precio total: \$814.9</div>

Ejercicio 8

Clase previa: “External Data Access”.

Pauta:

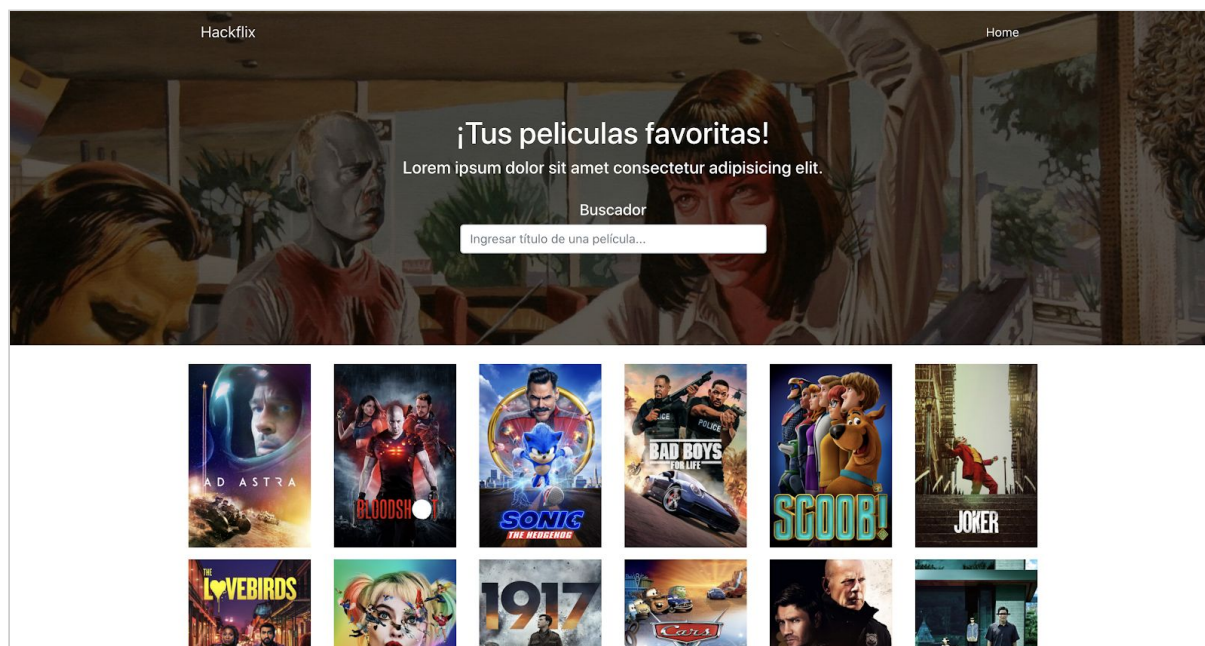
1. Duplicar el proyecto de las **tres películas** y usar dicho proyecto como base para este ejercicio.
2. En lugar de “inventar” los datos de las tres películas, se deberán obtener del servicio **The Movie Database** (TMDb). Para eso, cada alumno deberá crearse una cuenta gratuita en [el sitio](#) y obtener una **API Key**.
3. La idea es tener un componente llamado `Movie` al cual se le pasará (como *prop*) únicamente el `id` de la película.
4. Por más de que los datos de la película provengan de un servicio externo, el alumno igual deberá elegir las tres películas a mostrar.

Ejercicio 9

Clase previa: “Event listeners”.

Pauta:

Crear la aplicación **Hackflix**, que consiste de un listado de películas como se ve en el siguiente diagrama.



Las películas se deberán obtener de [este archivo](#) JSON (el cual se deberá descargar y colocar dentro del proyecto).

A medida que se escribe en el campo de texto, se deberán filtrar las películas del listado. Es decir, sólo se deberán mostrar las películas cuyo título contiene el texto ingresado.

En caso de que no haya resultados, mostrar un mensaje de error acorde.

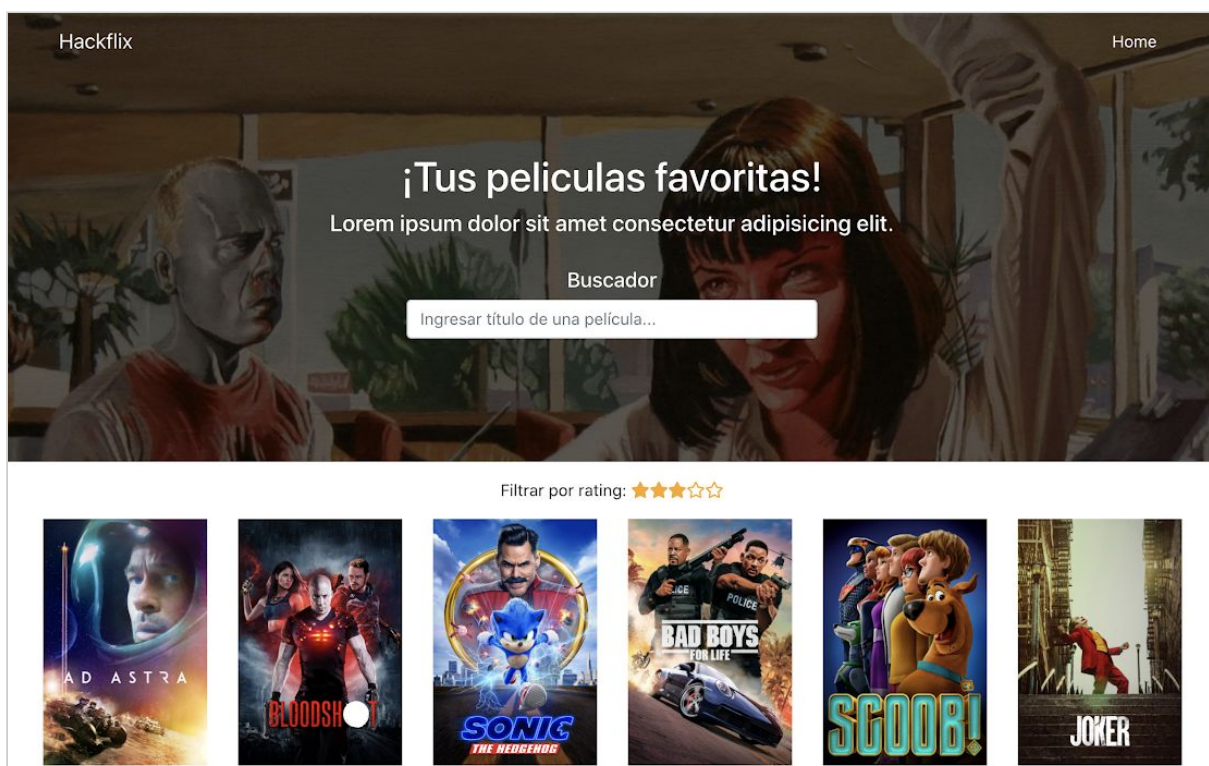
⚠ Nota: Pensar bien los **componentes** que se deberán crear.

Ejercicio 10

Clase previa: “Event listeners”.

Pauta:

Continuar con el ejercicio de **Hackflix** y agregar la funcionalidad de **filtrar por calificación** de las películas.



Notar que las películas tienen un atributo llamado `vote_average` cuyo valor oscila entre 1 y 10. La consigna es que cada estrella tenga un valor de dos puntos de rating. De esta manera, si se seleccionaron 4 estrellas, se deben filtrar las películas con 8 o más rating.

⚠ Nota: No utilizar jQuery ni manipular el DOM de forma directa (aunque lo hagan con JavaScript “puro” o “vanilla”). De hecho, esto aplica para todos los ejercicios de este sprint. En caso de usar Bootstrap, sólo se deberá importar su CSS.

Ejercicio 11

Clase previa: “Event listeners”.

Pauta:

Continuar con el ejercicio de **Hackflix** y agregar la funcionalidad de que al hacer click sobre una película, se abra un **modal** mostrando información detallada de la misma (título, rating, descripción, etc). Analizar si es conveniente crear un **componente** para el modal.

⚠ Nota: No utilizar jQuery ni manipular el DOM de forma directa (aunque lo hagan con JavaScript “puro” o “vanilla”). De hecho, esto aplica para todos los ejercicios de este sprint. En caso de usar Bootstrap, sólo se deberá importar su CSS.

Ejercicio 12

Clase previa: “Event listeners”.

Pauta:

Continuar con el ejercicio de **Hackflix**, pero ahora, en lugar de obtener las películas desde el archivo `movies.json`, se deberán obtener desde la API de **The Movie Database** (<https://developers.themoviedb.org/3/discover/movie-discover>) vía **AJAX**.

Ejercicio 13

Clase previa: “Event listeners”.

Pauta:

Continuar con el ejercicio de **Hackflix** y agregar la funcionalidad de **paginación**. La idea es que se muestren sólo 20 películas a la vez. Para ver películas adicionales se deberá pasar a la siguiente página. Analizar si es conveniente crear un **componente** para la paginación.

Ejercicio 14

Clase previa: “Event listeners”.

Pauta:

Continuar con el ejercicio de **Hackflix** y agregar la funcionalidad de **scroll infinito**. Es decir, en lugar de tener un paginador, las películas adicionales se mostrarán cuando el navegante desciende con el scroll y llega al final de la página.

Ejercicio 15

Clase previa: “Routing”.

Pauta:

Continuar con el ejercicio de **Hackflix** y agregarle rutas usando **React Router**.

1. La home de la app (pantalla en la que se estaba trabajando en los ejercicios anteriores) será la ruta “/”.
2. Al hacer click sobre una película, se deberá mostrar una nueva página conteniendo información detallada de la misma, con la ruta “/pelicula/:id”, siendo `id` el identificador de la película. Para esto se sugiere crear un componente llamado `MovieDetails`. Tal vez resulte útil utilizar el hook llamado [useParams](#). La idea es que cada vez que se entre a la página de una película, se haga una llamada a la API específica para traer los datos de la misma.
Nota: el Modal creado anteriormente deberá dejar de funcionar.
3. Mientras se hace el *request*, mostrar un **spinner** o **loader**, para indicarle al navegante que los datos de la película pueden demorar en aparecer.
4. Si no existe la película, mostrar un mensaje de error adecuado.
5. En la página de la película, deberá haber un botón para volver a la Home.

6. Agregar una página de “**Sobre Nosotros**” con información sobre Hackflix y una página de “**Contacto**” (usar *lorem ipsum* como textos). Agregar los links correspondientes en el navbar.
7. Agregar una página de “**Error 404 – Página no encontrada**”.
8. Hacer un redireccionamiento de “/movie/:id” a “/pelicula/:id”.
9. Separar la Home en dos páginas:
 - a. La Home (“/”), para mostrar listado de las últimas películas (API: Discover) y que incluirá el filtro de *rating*.
 - b. Una página (“/buscar”) específica de búsqueda (API: Search), la cual no incluirá el filtro de *rating*.

Ejercicio 16

Clase previa: “State”.

Pauta:

Investigar sobre [hooks personalizados](#). Los mismos permiten extraer cierta lógica de los componentes a una función reutilizable.

Continuar con el ejercicio de **Hackflix** y agregar la funcionalidad de **detectar si el sitio está offline**. Para esto se deberá crear un hook personalizado llamado `useIsOnline` el cual se deberá invocar de la siguiente manera:

```
const isOnline = useIsOnline();
```

Se deberán escuchar los eventos `online` y `offline` del objeto `window`. Tal vez, también se deberá acceder al atributo `navigator.onLine` que indica el estado de conexión en cierto momento.

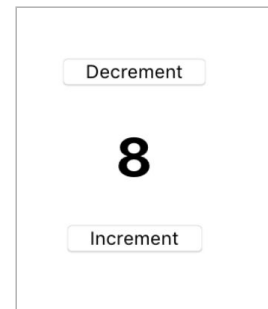
Ejercicio 17

Clase previa: “Redux”.

Pauta:

En este ejercicio se trabajará con un ejemplo simple, para “bajar a tierra” todo el teórico de Redux. Se deberá crear un **contador** que consta de:

- Un lugar donde mostrar el valor actual del contador.
- Un botón que incrementa el valor contador.
- Un botón que decrementa el valor del contador.



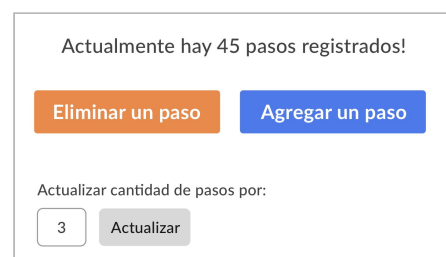
Detalle:

1. Crear proyecto de React.
2. Instalar `redux` y `react-redux`.
3. Crear una *store*, la cual es “inyectada” en toda la aplicación haciendo uso de un componente `Provider`.
4. Crear un componente `Count` el cual podrá acceder a la *store*.
5. Crear los botones correspondientes, los cuales despacharán las acciones necesarias.

Nota: Como buena práctica, colocar las acciones, reducers, creación de store, etc, en archivos separados.

Opcional/Extra: Agregar un `input` y un `button` que permitan *setear* un número cualquiera al contador.

Para ver la utilidad de ejercicio más “bajada a tierra”, se podría imaginar que un contador de este tipo podría servir para contar la cantidad de pasos de una persona.



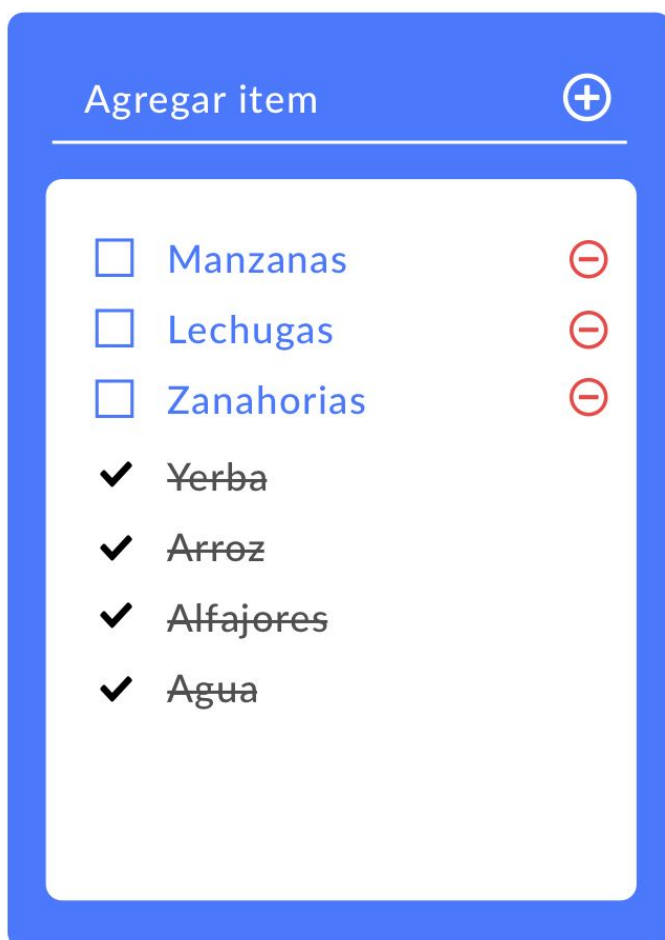
Ejercicio 18

Clase previa: “Redux”.

Pauta:

Crear una aplicación que sirva como una **lista de compras**. La aplicación debe de tener las siguientes características:

1. Permitir al usuario agregar y eliminar items a la lista de compras.
2. Marcar un elemento como comprado.
3. Filtrar la lista de compras por nombre de los elementos y estado (si fue comprado ya o no).



The mockup shows a blue-bordered container with a white background. At the top, there is a header bar with the text "Agregar item" and a plus icon in a circle. Below the header, there is a list of items. Each item consists of a checkbox, the item name, and a minus icon in a circle. The items are: Manzanas, Lechugas, Zanahorias, Yerba, Arroz, Alfajores, and Agua. The first three items have unchecked checkboxes, while the last four have checked checkboxes.

Item	Estado
Manzanas	No comprado
Lechugas	No comprado
Zanahorias	No comprado
Yerba	Comprado
Arroz	Comprado
Alfajores	Comprado
Agua	Comprado

Extra: Permitir que el usuario gestione sus propias listas de compra. Por ejemplo, que tenga una lista de compras sólo para artículos de panadería, otra para vegetales, etc. Similar a la aplicación que se puede observar a continuación:



Ejercicio 19

Clase previa: “Redux”.

Pauta:

Crear una aplicación que permita a un usuario **iniciar sesión** y luego cambiar sus datos personales (*settings*).

Detalle:

1. Crear tres rutas:
 - a. “/” la cual será pública (una pantalla de bienvenida con un link a “/login”).
 - b. “/settings” la cual será privada (ante la ausencia de autenticación, redirige a “/login”).
 - c. “/login” la cual será pública.
2. En “/login”, crear un formulario de **autenticación**, el cual hará uso del microservicio <https://ha-auth-react.now.sh/auth> para obtener un JWT. Se debe hacer un POST de las credenciales en formato JSON:

```
{  
  "username": "hack",  
  "password": "academy"  
}
```

Consideraciones:

1. Una vez que el usuario haya iniciado sesión en la aplicación, se deberá de guardar la información relacionada a la sesión (token, username) en Redux.
2. Los componentes que necesiten mostrar la información del usuario deberán conectarse a la *store* de Redux y obtenerla de ahí.
3. ¿Qué sucede al recargar la página?

Ejercicio 20

Clase previa: “Redux”.

Pauta:

Agregar persistencia con `redux-persist` a la autenticación del ejercicio 28.