Enunciado 3

Las cosas podían haber sucedido de cualquier otra manera y, sin embargo, sucedieron así. El camino – Miguel Delibes, 1950.

Y como sucedieron así, es hora de una vez más revisar qué tanto hemos aprendido de programación, de React y qué tan agudizado tenemos el ojo; para retarnos a enfrentar esto, el equipo de iMaster ha preparado un código, que como suele ser normal en la cita semanal, no esta funcionando del todo bien con 17 pruebas que, por algún motivo, no todas están siendo exitosas.

Al parecer el equipo de iMaster ha comenzado a explorar otros terrenos, o complementando los ya caminados, para esta semana se buscó darle un nuevo nivel al hook ***useState*** utilizando el hook ***useReducer*** y, aunque las pruebas son varias el funcionamiento es sencillo y el contexto es el siguiente:

- La aplicación es una sencilla lista de tareas, cada tarea tiene como atributos un id, una descripción y un estado y estas tareas pueden ser adicionadas, completadas y eliminadas.
- Cada tarea tiene persistencia en el Localstorage del navegador
- El aplicativo podría funcionar manejando el estado con el ***useState,*** pero debido a que el estado de cada tarea es un poco más complejo, el equipo optó por seguir la recomendación de la documentación y utilizar ***useReducer***. *useReducer a menudo es preferible a useState cuando se tiene una lógica compleja que involucra múltiples subvalores o cuando el próximo estado depende del anterior. useReducer además te permite optimizar el rendimiento para componentes que activan actualizaciones profundas, porque puedes pasar hacia abajo dispatch en lugar de callbacks*[1].
- Para usar el ***useReducer*** se debe implementar un Reducer, Un Reducer no

[1] Referencia de la API de los Hooks, Documentación ReactJS, https://es.reactjs.org/docs/hooks-reference.html#usereducer

es más que una función que para el caso del proyecto a revisar, cuenta con un switch que en cada caso o ***action*** va a retornar un nuevo estado o ***state***.

- Como la aplicación es una lista de tareas, el componente se debe pintar nuevamente cada que tenga una modificación, para ello se utiliza el ***useEffect***.
- La aplicación funciona con el componente de entrada que es el ***TodoApp***, este componente cuenta con 3 manejadores, para los 3 eventos, casos o ***action*** con los que se cuenta, el ***add***, el ***delete*** y el ***toggle***, estos métodos son invocados por eventos en la interfaz, como dar clic sobre el botón Agregar (***add***), clic sobre el botón Borrar (***delete***) o clic sobre el nombre de la tarea (***toggle***).
- Estos manejadores invocan dos elementos, un action con una propiedad ***type*** que es obligatoria, que indica el nombre de la acción a ejecutar por parte del ***Reducer*** (***add***, ***delete***, ***toggle***) y otra llave o propiedad normalmente llamada ***payload*** que es el valor para enviar a la función correspondiente, así, una nueva tarea para el ***add*** o el id de la tarea para el ***delete*** y ***toggle***.
- El otro elemento es un ***dispatch*** con la información del ***action*** correspondiente o con un objeto equivalente al definido en el ***action***.
- Es decir que nuestro ***TodoApp*** invoca al hook ***useReducer*** que tiene definidas las acciones que puedo ejecutar con la lista de tareas y para poder actualizar la lista tareas entonces, desde los manejadores debo invocar un ***dispatch*** que actualiza y retorna el estado de mi aplicación.
- El ***Reducer*** que se utiliza se llama ***todoReducer***.

Ahora la lista de los errores que se han detectado:

- En el título se indica el nombre de la app y un contador de las tareas en la lista pero el contador siempre marca -1 al parecer no esta obteniendo el dato desde la longitud de la lista de tareas.
- En el formulario para añadir tareas, se hizo una validación de la longitud del nombre de la tarea, si la tarea no tiene al menos 2 caracteres entonces debe romper con la ejecución y no crear la nueva tarea, pero la validación no esta

correcta y permite cadenas vacías.

- Relacionado con el mismo formulario al parecer se esta manteniendo el evento por defecto del ***form*** y se está actualizando la página cada que se da clic o ***enter*** al momento de crear una nueva tarea y la recarga de la página no debería estar sucediendo. Revisar el ***handleSubmit***.
- Dentro de la lista de tareas del componente ***TodoListItem*** se debería estar utilizando etiquetas ***li*** para la lista de tareas, pero al parecer están envueltas en algo incorrecto o que sobra.
- Adicionalmente el número de la tarea se encuentra iniciando en 0 cuando la primera tarea debería iniciar en 1, al parecer se está utilizando solo el ***índex***.
- Finalmente, el ***todoReducer*** creado no está ejecutando ningún action de los enviados, se cree que los nombres no coinciden con los que se han indicado a lo largo de este documento.

Su tarea, como ya se debe imaginar, es corregir dichos errores y levantar nuevamente el aplicativo, por favor recuerde que son 17 pruebas y por ningún motivo elimine o manipule la carpeta __***snapshots***__ del proyecto.

El equipo de iMaster envía una captura de como se vería el aplicativo funcionando correctamente:

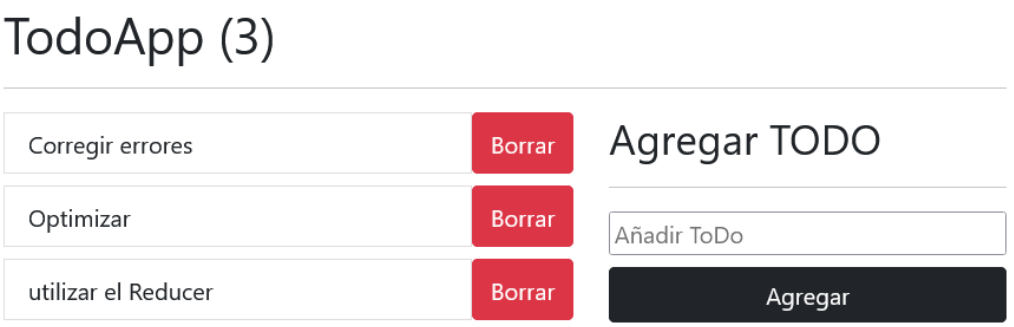

Ilustración 1 TodoApp funcionando

Al igual que con nuestros proyectos anteriores, tenemos algunas herramientas a nuestro favor y algunos requerimientos que debemos cumplir/instalar para poder ejecutar elproyecto.

1. Debe contar con Node.js instalado en su equipo, para ello solo requiere ir al sitio oficial (https://nodejs.org/es/) y seleccionar el paquete que se ajuste a su máquina.La interventoría recomendó utilizar la versión LTS.
2. Verifique la instalación de Node.js escribiendo en su terminal ***node -v***.
3. Una vez todo correcto, puede descomprimir el paquete de código que se le ha enviado y almacenarlo en una carpeta nombrada con su documento de identidad, todos los documentos son necesarios, no olvide incluir ninguno.
4. Con la carpeta nombrada correctamente, abra el proyecto con el editor de código de su preferencia, debe visualizar una estructura como la siguiente:

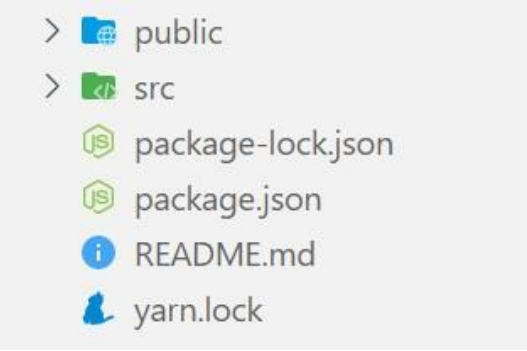

Ilustración 2 Estructura del proyecto

5. Ahora ya esta listo para ejecutar el proyecto, para ello debe abrir una terminaldentro de la carpeta del proyecto y ejecutar el comando ***npm install***.
6. Si todo esta correcto, deberá tener una nueva carpeta llamada "***node_modules***".
7. Recuerde que cuando crean que ya cumple con todos los puntos solicitados, puedeejecutar el validador, para hacerlo debe abrir una terminal dentro de la carpeta delproyecto y ejecutar el comando ***npm run test***.
8. Ahora solo resta comprimir el proyecto y enviarlo a la plataforma.
9. Antes de comprimir la carpeta nombrada con su número de documento de

identidad, donde se encuentran todos los archivos, debe eliminar la carpeta "***node_modules***" del proyecto.

10. Ahora ya puede crear su archivo comprimido, el único formato permitido es ***.zip***.
11. El archivo comprimido ***.zip*** debe conservar como nombre su número de documentode identidad para proceder a la carga en la plataforma.
12. Finalmente siga las instrucciones de la plataforma para recibir una evaluación.

Recuerde que el reto puede ser subido por un único integrante del equipo, indicando el número de documento de identidad de los demás compañeros que hayan trabajado en el proyecto, separados por comas y sin puntos u otra clase de separadores, ej.

```
1111111,2222222,33333333,4444444
```

La carpeta del proyecto debe ser nombrada solo con el número de documento de identidaddel usuario que hace la carga a la plataforma así la presente por todo su equipo, ej.