Almacenes (práctica)

Tiempo estimado: 25min

El objeto de esta práctica es afianzar, reforzar y consolidar los conocimientos teóricos presentados en la lección.

Al finalizarla, el estudiante:

- Habrá creado acciones.
- Habrá creado creadores de acciones.
- Habrá creado un almacén a través del cual obtener la hora actual.

Objetivos

El objeto de la práctica es presentar cómo trabajar con Flux. Aunque todavía no hemos visto todo, aún nos queda las vistas, vamos a comenzar a practicar un poco. La idea es crear una aplicación React que muestre la hora actual.

Esta hora se obtendrá de un almacén. Por otra parte, habrá un elemento en *background* que cada tres segundos solicitará, mediante una acción, que se compruebe si la hora ha cambiado.

Para darle algo de vida a la aplicación, el almacén considerará que ha cambiado en bloques de tres acciones. O sea, las acciones múltiplos de tres harán que el almacén considere que ha cambiado la hora. En ese momento, generará un evento de cambio para que el componente React que muestra la hora, la muestre de nuevo.

Preparación del entorno

Para comenzar, crearemos un proyecto de aplicación React mediante el generador de Justo:

1. Abrir una consola.
2. Si no tenemos instalado el generador de Justo para React, instalarlo:

   ```
   > npm install -g justo-cli justo-generator-react
   ```
3. Mostrar la ayuda del generador:

   ```
   > justo -g react help
   ```
4. Crear el directorio de la práctica e ir a él.
5. Invocar el generador de Justo:

   ```
   > justo -g react
   ```
   Responder a las preguntas realizadas por el generador. Se puede responder con los valores predeterminados, salvo:

   - Responder N cuando indique si usar React Router. No es necesario para la práctica.
   - Seleccionar flux como dependencia del proyecto.
6. Crear los directorios y archivos específicos de la arquitectura Flux:

   ```
   > justo -g react flux
   ```
7. Instalar las dependencias del proyecto:

   ```
   > npm install
   ```
8. Mostrar el catálogo de tareas automatizadas del proyecto:

   ```
   > justo -c
   ```
9. Invocar la tarea build del catálogo del proyecto para construir la aplicación:

   ```
   > justo build
   ```
10. Abrir el archivo dist/index.html con el navegador.

Debe aparecer el mensaje Hello World!

Creación de acciones

Vamos a comenzar definiendo las acciones. En nuestro caso, sólo tenemos una, pero podríamos tener tantas como fuera necesario, tal como veremos en la práctica de la próxima lección:

1. Ir a la consola.
2. Generar el archivo de acciones mediante el generador de Justo:

   ```
   > justo -g react flux action file
   ```

   Respuestas:

   - Carpeta donde añadirlo dentro de app/actions: <none>.
   - Nombre del archivo: ClockAction.
3. Añadir una acción al archivo de acciones ClockAction mediante el generador de Justo:

   ```
   > justo -g react flux action
   ```

   Respuestas:

   - Subcarpeta donde se encuentra el archivo dentro de app/actions: <none>.
   - Archivo de acciones: ClockAction.
   - Tipo de acción: check-time.

     Esto es el identificador único de la acción en la aplicación.
   - Nombre literal: CHECK_TIME.

     Esto es el nombre de la propiedad que usaremos en los controladores y que tiene asignado el tipo de acción.
   - Nombre del método creador: checkTime.
4. Editar el archivo app/actions/ClockAction.js y echarle un vistazo.

Creación del elemento background

A continuación, vamos a crear un elemento que disparará la acción de comprobación de hora cada tres segundos:

1. Crear el archivo app/lib/checker.js:

   ```
   //imports
   import ClockAction from "../actions/ClockAction";

   /**
    * Create a CHECK_TIME action periodically.
    */
   export default function check(timeout = 3000) {
     setTimeout(function() {
       ClockAction.checkTime();
       check(timeout);
     }, timeout);
   }
   ```

   Observe cómo genera la acción mediante el método checkTime() del módulo de acciones.
2. Editar el archivo app/index.jsx.
3. Importar la función de comprobación:

   ```
   import checker from "./lib/checker";
   ```
4. Invocarla al final del archivo, debajo de ReactDOM.render():

   ```
   checker(3000);
   ```
5. Guardar cambios.

Creación de almacén

Ahora, vamos a añadir el almacén que tiene como objeto proporcionar los datos sobre la hora que debe mostrar la aplicación. Tal como hemos indicado en los objetivos de la práctica, para darle algo de vidilla, el almacén sólo debe notificar los cambios cada tres acciones recibidas.

1. Ir a la consola.
2. Crear el archivo del almacén mediante el generador de Justo:

   ```
   > justo -g react flux store
   ```

   A la hora de responder:

   - Indicar ClockStore como nombre de la clase almacén.
   - Seleccionar ClockAction como módulo de acciones a usar.

     Se puede usar tantos como sea necesario. Los demás hay que añadirlos manualmente.
   - Seleccionar <another> como medio a través del cual acceder al origen de datos.
3. Editar el archivo app/stores/ClockStore.js. Y echarle un vistazo detenidamente.

   Observe que:

   - El módulo devuelve una instancia del almacén, la que se utilizará en la aplicación.
   - En el constructor, se pasa el despachador. Se utiliza para registrar el controlador de acciones del almacén.
   - El almacén define un atributo changed que actualizará cada vez que atienda tres acciones, indicando así que se ha producido un cambio.
   - El método hasChanged() devuelve si se ha producido un cambio en la última acción procesada.
   - El método __onDispatch() contiene la lógica del controlador de acciones del almacén. Registrado automáticamente en el despachador por la clase Store.
4. Definir el constructor como sigue:

   ```
   /**
      * Constructor.
      *
      * @param dispatcher:dispatcher The dispatcher where to register.
      */
     constructor(dispatcher) {
       super(dispatcher);
       Object.defineProperty(this, "changed", {value: false, writable: true});
       Object.defineProperty(this, "count", {value: 0, writable: true});
     }
   ```

   Utilizaremos el atributo count como contador del número de veces que se ha procesado una acción, de tal manera que sólo consideraremos que el almacén ha sufrido un cambio cada tres acciones procesadas.
5. Definir el controlador de acciones:

   ```
   __onDispatch(action) {
     const type = action.type;
     const data = action.data;

     if (type == ClockAction.CHECK_TIME) {
       this.checkTime();
       if (this.hasChanged()) this.__emitChange();
     }
   }
   ```

   Observe que el controlador primero invoca el método del almacén que se encarga de comprobar si se ha producido un cambio. Y a continuación, si es así, propaga el evento de cambio mediante el método __emitChange() definido en la clase Store.
6. Definir la operación que invocará el controlador de acciones para comprobar la hora:

   ```
   /**
   ```



```
 * Check whether time changed.
 */
checkTime() {
  this.count += 1;
  this.changed = (this.count % 3 == 0);
}
```

Observe que la operación modifica changed cada vez que se ejecuta. Fijará el valor true en múltiplos de tres. Por otra parte, tenga claro que la operación no genera el evento de cambio, es responsabilidad del controlador de acciones. Pero sí debe modificar changed para que así el controlador sepa si se ha producido cambio y, entonces, generar el evento de cambio.

7. Definir la operación de acceso mediante la cual los componentes React obtendrán la hora actual:

```
/**
 * Return the current time.
 *
 * @return date
 */
getCurrentTime() {
  return new Date();
}
```

8. Guardar cambios.

Creación del componente React

Ahora, vamos a crear el componente React que muestra la hora actual a medida que el almacén notifica que ha habido cambios:

1. Ir a la consola.
2. Crear el componente Time, el cual mostrará la hora actual, usando el almacén como reloj:

```
> justo -g react flux component
```

Responda con los valores predeterminados, salvo:

- Nombre del componente: Time.
- Tipo de estructura: Simple.
- ¿Controlar el evento de cambio del almacén? Y.
- Almacén a acceder: ClockStore.

3. Editar el archivo app/components/Time.jsx y echarle un vistazo.
4. Iniciar el estado del componente como sigue:

```
constructor(props) {
  super(props);

  //initial state
  this.state = {
    date: new Date()
  };
}
```

5. Ir al método privado handleStoreChange() y definirlo como sigue:

```
[handleStoreChange]() {
  this.setState({
    date: store.getCurrentTime()
  });
}
```

Este método procesa los eventos de cambio. Cuando se ejecuta es porque se ha producido un cambio de datos que puede afectar al componente. Su función, pues, es muy sencilla: solicitar de nuevo los datos al almacén, en este caso la hora, y actualizar el estado del componente para que así se reproduzca de nuevo.

6. Ir al método render():

```
render() {
```



```
    return (
      <p>
        {this.state.date.toLocaleTimeString()}
      </p>
    );
  }
```

7. Guardar cambios.
8. Editar el archivo app/views/App.jsx.
9. Importar el componente Time:

```
import Time from "../components/Time";
```

10. Ir al método render() y definirlo como sigue:

```
render() {
  return (
    <Time />
  );
}
```

11. Guardar cambios.
12. Ir a la consola.
13. Reconstruir la aplicación y abrirla en el navegador:

    Debe aparecer la hora actual.
14. Esperar 30 segundos y ver que se va actualizando la hora.

