Vistas (práctica)

Tiempo estimado: 35min

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

Al finalizarla, el estudiante:

- Habrá creado una aplicación React mediante Flux.
- Habrá creado un almacén.
- Habrá creado componentes Flux.
- Habrá creado archivos de acciones.
- Habrá realizado inserciones y supresiones de datos mediante Flux.

Objetivos

El objetivo de la práctica es crear una aplicación que consolide los conocimientos teóricos aprendidos a lo largo de las lecciones anteriores. Para ello, crearemos una aplicación que muestre datos en una tabla, obteniéndolos de un almacén. Y por otra parte, que inserte y suprima datos, con su correspondiente actualización mediante evento de cambio, también mediante un almacén.

Preparación del entorno

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

1. Abrir una consola.
2. Crear el directorio de la práctica e ir a él.
3. Invocar el generador de Justo:

   ```
   > justo -g react
   ```

   Responder a las preguntas realizadas por el generador. Puede usar los valores predeterminados, salvo:

   - Responder N cuando pregunte si usar React Router.

     No es necesario para la práctica.
   - Seleccionar Font Awesome como dependencia del archivo index.html.
   - Seleccionar flux como dependencia NPM.
4. Crear los archivos y directorios predeterminados específicos de la arquitectura Flux mediante el generador de Justo:

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

   ```
   > npm install
   ```
6. Invocar la tarea build del catálogo del proyecto para construir la aplicación:

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

   Debe aparecer el mensaje Hello World!

Creación de acciones

Vamos a comenzar definiendo las acciones:

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: BandAction.

3. Añadir la acción mediante la cual añadir una nueva banda 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: BandAction.
   - Tipo de acción: insert-band.
   - Nombre literal: INSERT_BAND.
   - Nombre del método creador: insertBand.

4. Crear la acción para suprimir bandas:

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

Respuestas:

   - Subcarpeta donde se encuentra el archivo dentro de app/actions: <none>.
   - Archivo de acciones: BandAction.
   - Tipo de acción: remove-band.
   - Nombre literal: REMOVE_BAND.
   - Nombre del método creador: removeBand.

5. Editar el archivo app/actions/BandAction.js y echarle un vistazo.

Creación de objeto de datos

Ahora, vamos a definir el archivo de datos, el cual se comportará como origen de datos del almacén:

1. Crear el archivo app/data/bands.js:

```
export default {
  "keane": {
    name: "Keane",
    origin: "England"
  },
  "zero-assoluto": {
    name: "Zero Assoluto",
    origin: "Italy"
  },
  "sisters-of-mercy": {
    name: "Sisters of Mercy",
    origin: "England"
  }
};
```

Creación de almacén

Ahora, vamos a crear el almacén:

1. Ir a la consola.
2. Crear el almacén mediante el generador:

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

Respuestas:

- Nombre de la clase almacén: BandStore.
- Módulo de acciones: BandAction.
- Modo de acceso al origen de datos: <another>.

3. Editar el archivo app/stores/BandStore.js. Y echarle un vistazo.
4. Importar el módulo de datos:

```
import bands from "../data/bands";
```

5. Modificar el controlador de acciones:

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

  if (type == BandAction.INSERT_BAND) {
    this.insertBand(data);
    if (this.hasChanged()) this.__emitChange();
  } else if (type == BandAction.REMOVE_BAND) {
    this.removeBand({id: data.id});
    if (this.hasChanged()) this.__emitChange();
  }
}
```

Observe cómo trabaja el controlador del almacén. Primero, filtra aquellas acciones que son de su interés. Y segundo, las procesa de manera muy similar. Por un lado, invoca la operación del almacén que realiza la operación asociada a la acción. Y a continuación, si se ha producido cambios en el almacén, dispara el evento de cambios para que los componentes React, que tiene asociados, sean notificados.

6. Definir la operación de acceso para la inserción de una nueva banda:

```
/**
 * Insert a band into the data source.
 *
 * @param band:object The band data.
 */
insertBand(band) {
  bands[band.id] = {
    name: band.name,
    origin: band.origin
  };

  this.changed = true;
}
```

Como se trata de una operación de escritura, al finalizar indica si ha acabado modificando el origen de datos. Tras una inserción, está claro que sí. Aunque si fuese posible que fallase, por ejemplo, por una violación de clave primaria, habría que controlar esa situación para no indicar que ha habido modificación en ese caso.

7. Definir la operación de acceso para la supresión de una banda:

```
/**
 * Remove a band.
 *
 * @param query:object  Query object: id.
 */
removeBand(query) {
  this.changed = bands.hasOwnProperty(query.id);
  delete bands[query.id];
}
```

Ahora, la operación sólo termina en cambio si la banda a suprimir está dada de alta.

8. Definir la operación para obtener todas las bandas:

```
/**
 * Return all the bands of the data source.
 *
 * @return object
 */
```



```
findAll() {
  return bands;
}
```

9. Definir la operación para obtener los datos de una banda:

```
/**
 * Return a band.
 *
 * @param query:object  The query object: id.
 * @return object
 */
findOne(query) {
  return bands[query.id];
}
```

Esta operación no es necesaria para esta práctica, pero se añade para mostrar al estudiante cómo podría hacerse si fuera necesario.

10. Guardar cambios.

Creación de componente React

Primero, vamos a crear un componente React cuyos datos proceden del almacén, pero no los consulta el propio componente:

1. Ir a la consola.
2. Crear un componente React mediante el generador:

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

Respuestas:

- Subcarpeta donde crearlo, dentro de app/components: <none>.
- Nombre del componente: BandRow.
- Tipo de componente: Immutable.
- Tipo de estructura: Simple.
- Tipo de implementación: Class.

3. Editar el archivo app/components/BandRow.
4. Importar el módulo de acciones BandAction:

```
import BandAction from "../actions/BandAction";
```

5. Definir el método render() como sigue:

```
render() {
  return (
    <tr>
      <td>{this.props.name}</td>
      <td>{this.props.origin}</td>
      <button onClick={(e) => this.handleRemoveClick(e, this.props.id)}>
        <span className="fa fa-remove" />
        Suprimir
      </button>
    </tr>
  );
}
```

6. Definir el método que controla las peticiones del usuario de suprimir bandas:

```
/**
 * Handle remove click.
 *
 * @param evt:SyntheticEvent  The event object.
 * @param id:string           The band id to remove.
 */
handleRemoveClick(evt, id) {
  if (confirm("¿Estás seguro?")) {
    BandAction.removeBand({id});
  }
```

```
}
```
Observe cómo se realiza la supresión. Se genera su acción asociada, pasándole los datos que necesitará el almacén para realizarla. El componente no accede directamente al almacén. Simplemente, solicita la operación mediante una acción.

7. Guardar cambios.

Creación del componente Flux

Ahora vamos a crear un componente React que accede al almacén para obtener sus datos:

1. Ir a la consola.
2. Crear el componente mediante el generador:

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

   - Subcarpeta de app/components: <none>.
   - Nombre de componente: BandTable.
   - Tipo de estructura: Simple.
   - ¿Controlar el evento de cambio del almacén? Y.
   - Módulo de acciones: BandAction.
   - Almacén: BandStore.

3. Editar el archivo app/components/BandTable.jsx.
4. Importar el componente BandRow:

```
import BandRow from "./BandRow";
```
5. Definir el estado inicial del componente:

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

  //initial state
  this.state = {
    bands: []
  };
}
```
En la propiedad bands del estado, mantendremos los datos de las filas de la tabla.

6. Ir al método componentDidMount() y definirlo como sigue:

```
componentDidMount() {
  store.addListener(() => this[handleStoreChange]());
  this.setState({
    bands: store.findAll()
  });
}
```
Por un lado, hay que registrar el componente en el almacén para que cuando el almacén cambie, le notifique los cambios mediante la acción de cambio. Por otro lado, hay que solicitar las bandas a mostrar en la tabla y añadirlas al estado del componente.

7. Ir al controlador del evento de cambio y definirlo como sigue:

```
[handleStoreChange]() {
  this.setState({
    bands: store.findAll()
  });
}
```
Cada vez que se ejecute el controlador de cambio, es porque los datos han cambiado. Basta con leerlos de nuevo y añadirlos al estado del componente. Así, se disparará una nueva reproducción del componente y se mostrará los nuevos datos.

8. Definir el método render() como sigue:

```
render() {
```



```
    return (
      <table>
        <thead>
          <tr>
            <th>Nombre</th>
            <th>Origen</th>
            <th>Suprimir</th>
          </tr>
        </thead>

        <tbody>
          {
            Object.keys(this.state.bands).map((id) => {
              var band = this.state.bands[id];
              return <BandRow key={id} id={id} {...band} />;
            })
          }
        </tbody>
      </table>
    );
  }
```

9. Guardar cambios.
10. Editar el archivo app/views/AppIndex.jsx.
11. Importar el componente BandTable:

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

12. Fijar el título de la página:

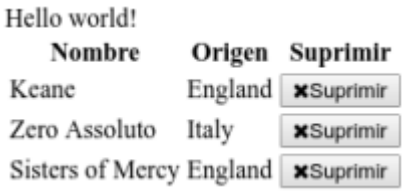
```
componentDidMount() {
  document.title = "Listado de bandas";
}
```

13. Definir el método render():

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

14. Guardar cambios.
15. Ir a la consola.
16. Reconstruir la aplicación y abrirla en el navegador:

    Debe aparecer:



17. Hacer clic en el botón Suprimir de Zero Assoluto.
18. Hacer clic en el botón Suprimir de Sisters of Mercy.
19. Hacer clic en el botón Suprimir de Keane.

Creación de formulario de inserción de datos

Veamos cómo implementar la funcionalidad para añadir nuevas bandas al origen de datos mediante el almacén:

1. Ir a la consola.
2. Crear un componente formulario mediante el generador:

```
> justo -g react flux form
```


Respuestas:

- Añadir a subcarpeta: <none>.
- Nombre de componente: NewBandForm.
- Cancelar la acción predeterminada: Y.
- Controlar los eventos de cambios de los elementos de entrada: Y.
- Módulo de acciones: BandAction.
- Almacén: <none>.

3. Editar el archivo app/components/NewBandForm.jsx.
4. Definir el estado inicial del formulario:

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

  //initial state
  this.state = {
    id: "",
    name: "",
    origin: ""
  };
}
```

5. Ir al método controlador del envío del formulario y solicitar la inserción de los datos en el almacén:

```
handleSubmit(evt) {
  //(1) cancel default actions
  evt.preventDefault();

  //(2) insert band
  BandAction.insertBand({
    id: this.state.id,
    name: this.state.name,
    origin: this.state.origin
  });

  //(3) reset form
  this.setState({
    id: "",
    name: "",
    origin: ""
  });
}
```

Al igual que la supresión, el componente no accede directamente al almacén. Genera una acción que contiene los datos que necesitará el almacén para llevar a cabo la operación.

6. Definir el método render() como sigue:

```
render() {
  return (
    <form onSubmit={(evt) => this.handleSubmit(evt)}
          onChange={(evt) => this.handleChange(evt)}>
      <input type="text"
             name="id"
             value={this.state.id}
             placeholder="Band id"
             required />
      <input type="text"
             name="name"
             value={this.state.name}
             placeholder="Band name"
             required />
      <input type="text"
             name="origin"
             value={this.state.origin}
             placeholder="Band origin" />
```



```
        <input type="submit" value="Insert" />
      </form>
    );
  }
```

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

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

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

```
render() {
  return (
    <div>
      <NewBandForm />
      <BandTable />
    </div>
  );
}
```

11. Guardar cambios.
12. Reconstruir la aplicación y abrirla en el navegador.

    Debe aparecer:

13. Escribir los siguientes datos y hacer clic en Insert:
    - Identificador de la banda: capital-cities.
    - Nombre de la banda: Capital Cities.
    - Origen de la banda: US.

    Debe aparecer los datos de la nueva banda en la tabla.

