upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

DESPLIEGUE DE SERVICIOS MULTIMEDIA
Máster Universitario en Ingeniería de Telecomunicación

AppGaztaroa

Commit 11: "Ejercicio Formularios y Modals con Redux"

Marko Galarza Galarza
Mikel Sagues García

Ejercicio Formularios y Modals con Redux

Commit 11: "Ejercicio Formularios y Modals con Redux"

En este segundo ejercicio no-guiado pondremos en práctica los conocimientos adquiridos en relación a Redux, para crear un formulario que permita al usuario de la aplicación añadir comentarios en relación a las excursiones de Gaztaroa.

Para ello, además de renderizar los componentes de la interfaz gráfica que nos permitan introducir los datos requeridos en el formulario, deberemos actualizar nuestro repositorio Redux de tal forma que los comentarios introducidos se almacenen en el estado de la aplicación. Nótese que no pretendemos modificar la información que se almacena en el servidor. Eso lo dejamos para más adelante.

Para mayor claridad, dividimos los requerimientos de este ejercicio en dos apartados, centrando el primero en la interfaz gráfica y dejando para el segundo el enlace de dicha interfaz con Redux.

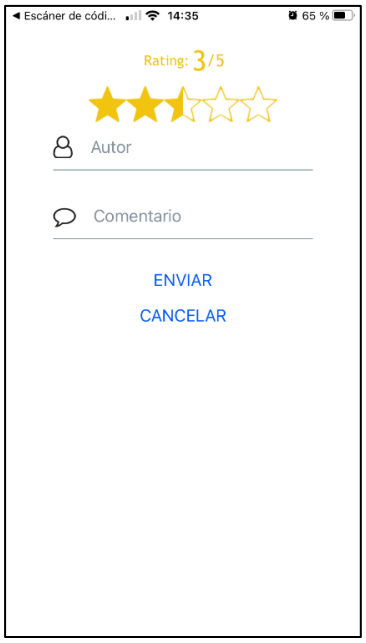
1. Formulario con modal

Antes de nada, es importante aclarar que los formularios no existen en aplicaciones móviles nativas. Lo que haremos, por lo tanto, será "simular" un formulario, combinando componentes de tal forma que repliquemos la funcionalidad clásica de los formularios a la que estamos acostumbrados. Conceptualmente un formulario se puede dividir en dos partes:

- Apariencia visual e introducción de datos.
- Envío y gestión de los datos enviados.

Como primer paso, debemos añadir un nuevo botón (un *Icon*, en realidad) a la vista *detalleExcursión*, para dar acceso al usuario al formulario en que introducirá los datos. El icono empleado es *Pencil* de Font Awesome. Cuando el usuario hace *clic* en dicho botón, la aplicación "lanzará" un *Modal*, en el que renderizaremos los componentes de nuestro formulario. Los Modals son componentes que se muestran por encima de alguna de las vistas de nuestra aplicación y generalmente tienen el cometido de advertir sobre algún evento o pedir la confirmación del usuario sobre alguna acción que esté a punto de llevarse a cabo. También es típico utilizarlos como ventanas de "login", o a modo de breve formulario para que el usuario introduzca algún dato concreto que se necesite en un momento dado. En nuestro caso, lo utilizaremos como contenedor para nuestro formulario, cuyo aspecto será el siguiente[1]:

[1] Nótese que los dos iconos se han centrado en la pantalla.



En la parte superior tenemos un componente para introducir la *valoración* de la excursión (lo mucho o poco que nos ha gustado), compuesto por cinco estrellas y con las siguientes características:

- Se ha empleado el componente *Rating* de la librería *react-native-ratings*[2].
- Al cargar el formulario, el *Rating* tiene que situarse en el valor "5" (5 estrellas).
- La selección del usuario debe "redondearse" a un número entero de estrellas (no permitiremos, por ejemplo, una puntuación de "3.25").

El formulario dispone de dos campos de texto, uno para el nombre y otro para el comentario:

- A la izquierda de cada campo de texto aparecerá un icono. Se puede emplear los que se ven en la figura, o cualquier otro, siempre que tenga coherencia con el tipo de datos a introducir.
- Los campos de texto son componentes *Input* de React Native Elements.

Finalmente, tenemos dos botones: uno para cancelar el envío del comentario y otro para confirmarlo. El botón *Cancelar* cerrará el *Modal* y reseteará sus datos (se recomienda crear una función en nuestro componente para realizar dicha tarea). El botón *Enviar* será la puerta de entrada al trabajo que realizaremos en la segunda parte del ejercicio.

2. Actualizar comentarios en Redux

En este segundo apartado se describen los requerimientos del ejercicio a la hora de añadir los comentarios introducidos por el usuario al repositorio Redux. Es importante seguir las pautas marcadas a continuación, para que el código que generemos sea similar en todos los casos.

[2] https://github.com/Monte9/react-native-ratings

En concreto, lo que se desea hacer es actualizar la variable *comentarios* del estado, para incorporar el comentario introducido por el usuario a través del formulario que hemos creado en la primera parte del ejercicio.

Para ello:

- Los datos introducidos en el formulario se almacenarán en el estado del componente *DetalleExcursion*, para poder ser gestionados con mayor facilidad cuando el usuario acepte enviar dichos datos y se cierre el *Modal* que contiene el formulario. También añadiremos una nueva variable para almacenar el estado del Modal. Es decir, una nueva variable que nos indique si el Modal se está mostrando o permanece oculto. En este caso no lo almacenamos en el *Store* al tratarse de datos locales asociados al formulario.

```
class DetalleExcursion extends Component {
  constructor(props) {
    super(props);
    this.state = {
        valoracion: 5,
        autor: '',
        comentario: '',
        showModal: false
    }
  }
```

- Se crearán dos nuevos tipos de acciones, siguiendo las mismas pautas que ya se emplearon en el caso de los favoritos (*Commit 10 Activity Indicator y addFavoritos*):
  - *POST_COMENTARIO*
  - *ADD_COMENTARIO*

  Ojo! No confundir este nuevo tipo *ADD_COMENTARIO* (cuya función es añadir un nuevo comentario) con el tipo *ADD_COMENTARIO**S*** que ya existe y que empleamos para la carga inicial de todos los datos desde el servidor.

- Y dos nuevas funciones en *ActionCreators* asociadas a dichas acciones:
  - *postComentario()*
  - *addComentario()*

  La primera de ellas será una función *Thunk* que simplemente "simulará" un envío del nuevo comentario a un servidor remoto, tal y como hicimos en el ejercicio anterior, introduciendo un retardo de dos segundos para, a continuación, llamar a *addComentario()*.

- Creamos la función, *toogleModal()* que será la responsable de invertir la variable de estado *showModal* en función del estado del *Modal.* De este modo, disponemos de una función que nos permite mostrar y ocultar el *Modal* a nuestra conveniencia, sin ligarla a ninguna otra funcionalidad de nuestra aplicación, con solo consultar y modificar su estado.

```
toggleModal() {
    this.setState({showModal: !this.state.showModal});
}
```

- Los datos del formulario deben "resetearse" tanto si el usuario decide completar el envío, como si decide cancelarlo. Para ello, trasladaremos dicho cometido a una nueva función de nombre resetForm()

```
resetForm() {
  this.setState({
      valoracion: 3,
      autor: '',
      comentario: '',
      dia: '',
      showModal: false
  });
}
```

- Se creará un método *gestionarComentario* en el componente *DetalleExcursion* para gestionar el envío de los datos al repositorio *Redux*.
- Parámetros a generar antes de despachar la acción al reducer
    - *postComentario()* recibirá cuatro parámetros desde el componente *DetalleExcursion* y creará un quinto parámetro *dia* antes de despachar la acción a *addComentario()* quien, a su vez, lo trasladará al *reducer*:
        - Parámetro 1: ***excursionId***
        - Parámetro 2: ***valoracion***
        - Parámetro 3: ***autor***
        - Parámetro 4: ***comentario***
        - Parámetro 5: ***dia*** (emplear para ello un elemento JavaScript de tipo *Date*, siguiendo las indicaciones que se muestran en la bibliografía).

      De manera alternativa, este mismo proceso se puede realizar generando el comentario en el componente *DetalleExcursion*, de tal forma que se despache a *postComentario()* un único parámetro (el comentario con todos los campos asignados).
- Parámetros generados en el propio reducer
    - El objeto *comentario* que se concatene al array *comentarios* del Store debe tener un parámetro ***id*** válido, que tendrá que ser generado y asignado en el propio *reducer* para evitar de este modo que el reducer reciba, desde dos destinos distintos, el mandato de concatenar dos comentarios con el mismo *id*.

Una vez completado todo lo anterior, nuestra aplicación debería mostrar la funcionalidad que se describe en el vídeo adjunto a la práctica.

Como siempre, si todo es correcto, ya podemos hacer el *commit* correspondiente a este ejercicio y subir a nuestro tablero de Trello un breve documento con las claves del trabajo realizado.

3. Bibliografía

- **Rating**
  https://github.com/Monte9/react-native-ratings
- **Input**
  https://reactnativeelements.com/docs/components/input
- **React-Native-FontAwesome**
  https://github.com/FortAwesome/react-native-fontawesome
  https://fontawesome.com
- **Button**
  https://reactnative.dev/docs/button
- **Modals**
  https://reactnative.dev/docs/modal.html
- **Date**
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date