Introducción

Tiempo estimado: 15min

Vamos a comenzar con el curso de desarrollo de aplicaciones React con Flux. Tiene como objetivo explicar la arquitectura Flux para el desarrollo de aplicaciones webs con React. Asume del estudiante conocimientos previos de React.

Para comenzar, se presenta formalmente Flux, la arquitectura de aplicaciones diseñada por Facebook para el desarrollo de aplicaciones React profesionales. Y finalmente, se resume el plan de estudio del curso.

Al finalizar la lección, el estudiante sabrá:

- Qué es una arquitectura de aplicaciones.
- Qué es Flux.
- Cuál es el programa del curso.

Introducción

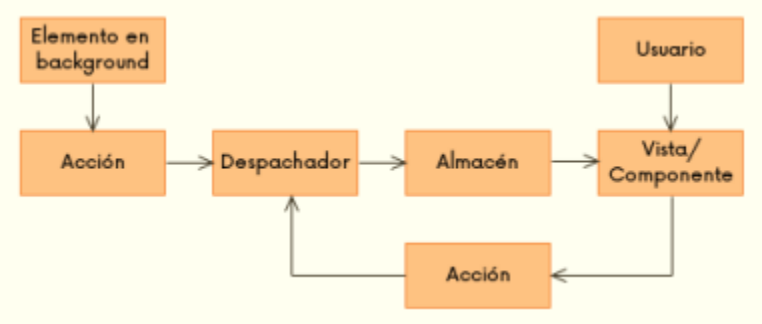
Una arquitectura de aplicaciones (*application architecture*) define la estructura que debe presentar la aplicación. Define los distintos elementos en que se debe dividir con objeto de facilitar, así, su diseño y desarrollo. El objeto de una arquitectura es la máxima que dice: divide y vencerás. Dividir el código de la aplicación en partes con responsabilidades claras y bien definidas.

Flux es una arquitectura de aplicaciones webs, o sea, una manera de ver e implementar aplicaciones webs clientes, organizando sus elementos de una determinada manera. Está diseñada por Facebook y es la arquitectura de facto utilizada en el desarrollo de aplicaciones React. Su objetivo es el desacoplamiento de los componentes de la aplicación con objeto de ayudar a desarrollar aplicaciones fáciles de mantener.

Cuando se diseñó Flux, se tuvo en cuenta principalmente el flujo de datos entre los distintos elementos. Concretamente, se diseñó con un flujo de datos unidireccional (*one-way data flow*). Lo que permite mayor flexibilidad a la hora de añadir nuevas características a la aplicación.

En Flux, básicamente encontramos los siguientes conceptos:

- Las acciones.
- Los controladores de acciones.
- El despachador.
- Los almacenes.
- Los controladores de cambio.
- Las vistas.



Como se puede observar en la imagen, el flujo de datos es unidireccional y va de la acción hasta la vista.

Groso modo, una acción (*action*) representa un evento o hecho acaecido y que hay que procesar o atender como, por ejemplo, la petición de añadir un nuevo empleado a la base de datos, la supresión de un pedido, etc. Puede venir de la interactuación del usuario con la interfaz de usuario o de un elemento interno como, por ejemplo, la solicitud de comprobación automática de correos electrónicos de la bandeja de entrada.

Cuando se genera una acción, la acción se notifica al despachador (*dispatcher*). El elemento de la aplicación que se encarga de determinar quién debe atender la acción. Hay sólo un despachador por aplicación. Pero puede haber varios elementos del sistema que deben atender la acción.

El despachador no hace más que consultar una tabla interna y encontrar los controladores de la acción (*action handlers*). Funciones JavaScript que se encargan de atender la acción en cuestión. Y entonces, las ejecuta, una a una, para que cada uno pueda hacer su parte de atención o trabajo.

El controlador de la acción tiene básicamente un objeto. Solicitar al almacén que realice el acceso a la base de datos que requiere la acción. Por ejemplo, si la acción conlleva leer datos de la base de datos, el controlador solicitará al almacén que recupere los datos. Si la acción conlleva suprimir un registro, le solicitará la supresión del registro. Y así con cualquier tipo de acceso a datos que requiera la acción bajo procesamiento.

Así pues, un almacén (*store*) no es más que la interfaz con el origen de datos, generalmente, un servicio web REST, que actúa como intermediario con la base de datos. Este almacén puede acceder a su origen de datos, por ejemplo, jQuery, los módulos http o request de Node si la aplicación se empaquetó con Browserify. O lo que se prefiere actualmente, las APIs Fetch o WebSockets disponibles en los navegadores. A diferencia del despachador, puede haber varios almacenes en la aplicación. Tantos como sea necesario.

Finalmente, cuando el almacén ha realizado el acceso, el controlador de la acción solicita al almacén que propague el evento de cambio. El evento de cambio (*change event*) representa un cambio en el estado de la aplicación. Como se trata de un cambio, generalmente conllevará la actualización de los datos que se están presentando al usuario. Así, por ejemplo, si la acción se disparó por la entrada de un nuevo correo en el buzón, será necesario informar a la interfaz de usuario que muestre ese nuevo correo. Incluso que incremente el contador de correos pendientes de leer. Una vez el almacén ha leído el nuevo mensaje del buzón de entrada, el controlador de la acción le pedirá al almacén que genere su evento de cambio para que así los componentes correspondientes actualicen lo que están presentando.

En último lugar, tenemos las vistas. Una vista (*view*) no es más que un componente React que representa un elemento de la interfaz de usuario. Las vistas son los elementos de la aplicación que presentan los datos a los usuarios. Como veremos más adelante en el curso, cada vista representa un elemento HTML de la aplicación. Visto lo visto, las vistas son los elementos que actualizarán su contenido cada vez que un almacén propague el evento de cambio. Para ello, cada vista registra su controlador de cambio (*change handler*) ante los almacenes cuyos accesos conlleven cambios en lo que está visualizando al usuario. Así, cuando el controlador de la acción genera el cambio, el almacén notificará a las vistas que pueden verse implicadas por el cambio y, por lo tanto, deben actualizar lo visualizado al usuario.

Hasta aquí la introducción de los componentes de una aplicación React implementada mediante una arquitectura Flux. Pero como las cosas siempre se ven mejor mediante un ejemplo, veamos uno.

Supongamos una aplicación que presenta los correos electrónicos de un usuario. Como sabemos, la interfaz de usuario estará formada por componentes React. Tantos como sea necesario. Cada componente presentará algo al usuario. Por ejemplo, tendremos uno que muestre la lista de correos. Otro que muestre el correo que estamos leyendo. Otro que muestre cuántos correos tenemos pendientes de abrir. Etc., etc., etc.

Por otro lado, tendremos un almacén. Aunque debe quedar claro que podríamos tener tantos como sea necesario. Este almacén será el puente de la aplicación con el servidor de correo. Observe que no es necesario que tras el almacén haya siempre una base de datos propiamente dicha. Este almacén dispondrá de métodos con los que realizar el acceso al servidor. Uno para obtener la lista de correos. Otro para obtener un correo concreto. Otro más para suprimir un correo. Y así tantos como sea necesario.

Flux fija que cualquier cambio de datos debe ser realizado por los almacenes. Hasta aquí todo normal. Pero está claro que los componentes React ser notificados, de alguna manera, de que lo que están presentando al usuario está obsoleto, pues se ha producido cambios que conllevan el cambio también en lo que se visualiza al usuario. Para ello, cada componente registra su correspondiente controlador de

cambio ante los almacenes cuyos cambios le afecten. En nuestro ejemplo, los componentes React se registrarán ante el almacén de correo. En pocas palabras, cada controlador de cambio lo que hace es solicitar a React que vuelva a presentar la vista, *si los datos relacionados con ella han cambiado*. React es lo suficientemente inteligente para detectar si los cambios que se han producido afectan a la vista. Pero es un poco puntilloso. Algo exigente. Requiere que le avisemos que lo compruebe. He ahí donde entra en acción el controlador de cambio. El evento de cambio lo que dice es: *por favor, vuelve a solicitar los datos, los que tienes podrían estar obsoletos, y reprodúcete de nuevo, pero con los nuevos datos*.

Para que todo funcione, es necesario que la aplicación genere acciones, cada una de ellas relacionada con un determinado acontecimiento o evento. Vamos a suponer que tenemos configurada la aplicación para que cada minuto se compruebe si hay correos nuevos. Cada minuto un elemento de la aplicación se levantará, generará la acción de comprobar correo y volverá a dormirse.

Esta acción se generará contra el despachador, recordemos, el que gestiona las acciones. El despachador recorrerá los controladores de acciones que tiene registrados, los cuales han registrado los almacenes, e invocará aquellos que estén relacionados con la acción generada. Siguiendo con nuestro ejemplo, el controlador de acción le pedirá al almacén que compruebe si hay correo nuevo. Si es así, el controlador le pedirá que emita su evento de cambio para que de esta manera, las vistas correspondientes reflejen el correo que ha entrado. En otro caso, dará por terminada la atención sin necesidad de generar el evento de cambio.

Si se genera el evento de cambio, los controladores de cambio lo que harán es pedir a React que actualice los componentes afectados por el nuevo correo. En este punto, el motor de React cogerá los componentes y los volverá a reproducir. Siendo este momento cuando los componentes accederán al almacén y le pedirán lo que deben presentar.

Observemos que el almacén no pasa los cambios a los componentes. Sólo les avisa de que ha habido cambios que les pueden afectar. Son los componentes, con el evento de cambio, los que solicitan de nuevo los datos explícitamente al almacén. En esta nueva consulta, se encontrarán los nuevos datos.

Pero los componentes no lo hacen constantemente. Sólo cuando el almacén les indica que hay cambios a presentar. El estudiante podría pensar que sería más cómodo que el almacén les pasara los cambios directamente, evitando así la llamada explícita de las vistas. Es discutible. La razón por la que no se hace así es bien sencilla. Cada componente tiene su propio conjunto de datos. Y deseamos que los componentes de la aplicación se encuentren desacoplados. Si el almacén tiene que saber qué pasar a cada uno de ellos, no se encontrará desacoplado. Lo único que necesita saber es a quién debe informar de que se ha producido un cambio determinado. Hecho así, los componentes irán al almacén y le solicitarán sus datos concretos.

Al tener desacoplado el almacén del despachador y de los componentes React, conseguimos mejorar principalmente su mantenimiento, su reutilización y su implementación.

Información del curso

Este curso es el segundo de varios dedicados al *framework* React.

Tiene como objetivo presentar los fundamentos del desarrollo de aplicaciones React, usando la arquitectura de aplicaciones Flux de Facebook. Existe otras arquitecturas muy aceptadas como, por ejemplo, Redux.

Al finalizarlo, el estudiante sabrá:

- Qué es una arquitectura de aplicaciones.
- Qué es Flux.
- Cómo diseñar una aplicación React mediante Flux.
- Cómo usar el generador de Justo.js para facilitar la escritura de aplicaciones React con Flux.

Conocimientos previos

El estudiante debe tener conocimientos de HTML, JavaScript, Node y React. Conocimientos de CSS son recomendables, pero nada necesarios para este curso. Mientras que de Justo.js son muy, pero que muy recomendables.

Plan del curso

El curso tiene una duración aproximada de 4 horas. Se divide en 5 lecciones, cada una de ellas con una parte de teoría y generalmente una de práctica Al finalizar el curso, puede realizar un examen de tipo test, con el que evaluar los conocimientos adquiridos.

El enfoque a seguir es muy sencillo: ir lección a lección; primero hay que leer la teoría y, después, realizar la práctica. Se recomienda encarecidamente que el estudiante realice cada lección, tanto teoría como práctica, en el mismo día, con el menor número de interrupciones a lo largo de su estudio. Al finalizar el curso, se recomienda encarecidamente realizar el examen.

A continuación, se enumera las distintas lecciones y el tiempo estimado para su estudio:

Lección	Teoría	Práctica	Descripción
1 Introducción	15min	-	Esta lección.
2 Despachador y acciones	20min	-	Describe qué son y para qué se utilizan el despachador y las acciones en la arquitectura Flux.
3 Almacenes	20min	25min	Describe los almacenes y sus responsabilidades en las aplicaciones desarrolladas bajo la arquitectura Flux.
4 Vistas	10min	35min	Describe las vistas, los componentes React mediante los cuales se presenta datos procedentes de almacenes de la aplicación.
5 API Fetch	10min	-	Describe la API Fetch con la que se suele acceder a datos del servidor web asíncronamente.
Examen	60min		Evaluación de los conocimientos adquiridos.

Información de publicación

Título Desarrollo de aplicaciones React con Flux

Autor Raúl G. González - raulggonzalez@nodemy.com

Primera edición Octubre de 2016

Versión actual 1.0.0

Contacto hola@nodemy.com

