🇪🇸 Historia de ReactJS

Introducción

En la actualidad ReactJS es una de las librerías más famosas y prácticamente cualquier desarrollador ha escuchado hablar de ella. Pero esto no siempre ha sido así, todo tiene un comienzo.

Para entender React es bueno saber por qué se creó la librería y qué problemas se pretendía solucionar con ello. En el año 2011 los desarrolladores de Facebook tenía problemas con la arquitectura de sus aplicaciones → patrón MVC.

En esta arquitectura MVC el usuario interactúa con la aplicación a través de la **vista**, las acciones del usuario se gestionan en el **controlador** y este aplica los cambios necesarios sobre el **modelo** y el **modelo** le notifica a la **vista** los nuevos cambios y le presenta al usuario dichos cambios. Se puede decir que cada acción del usuario provocaba una llamada al servidor.

Desarrollo iterativo

Desarrollar nuevas funcionalidades conlleva la implementación de nuevas funcionalidades, lanzarlas, escuchar el feedback, detectar mejoras y volver a empezar.

Cuando Facebook se dio cuenta de que su modelo de negocio Facebook Ads estaba colapsando por la arquitectura MVC tuvieron que ponerse manos a la obra con un desarrollo iterativo para implementar mejoras.

Curva de aprendizaje

La imposibilidad de manejar un único modelo y llevarlo a cientos por la complejidad de los datos que estaban recogiendo hicieron que su código fuese complejo y cada vez que se unía un nuevo desarrollador al equipo la curva de aprendizaje era inmensa.

Esto provocaba que la improductividad del equipo aumentaba ante una nueva pieza o la perdida de otra… Esto estaba frenando a la plataforma en su crecimiento puesto que la capacidad de desarrollo estaba siendo mermada por su la complejidad de su código.

Conclusión inicial

Estos fueron los puntos que llevaron a la empresa a repensar la manera de hacer las cosas dando vida a una librería que ha ido iterando hasta ahora conocida como ReactJS.

Primeros componentes

En el origen primigenio el equipo de Facebook usaban un sistema de plantillas para poder reutilizar cierta funcionalidad repetida en algunos puntos de su aplicación. Además dentro de esta plantillas teníamos otras plantillas más pequeñas que la componetizaban.

Todo esto estaba basado en un fichero php con una función que disparaba el contenido según la solicitud o información recibida. Es decir a través de sus parámetros generaba un fichero HTML con unos estilos asociados y un JS para la interacción. Para la escritura de estos ficheros HTML usaban XHP que permitía escribir HTML como si fuese un template string.

El problema era que este motor de plantillas era PHP y PHP solo funciona en el servidor.

Origen de React y JSX

Una vez tenían su sistema de plantillas vieron que el volumen de peticiones al servidor no era escalable y fue entonces cuando decidieron que era el momento de dar entrada a nuestro querido Javascript.

Gracias a Javascript introdujeron que toda la lógica de renderizado de template fuese llevada a cabo del lado del cliente y por otro lado todo lo que dependía del server fuese PHP quien lo gestionase, de este modo aliviamos la carga de peticiones.

De este proceso salió FaxJS o lo que conocemos hoy en día ReactJS. Además que permitía la escritura de HTML con Javascript para la creación de pequeños templates dando vida a lo que conocemos como JSX → Lógica + vista en un mismo fichero.

Imperativo vs Declarativo

Cuando el equipo trabajaba sobre una nueva funcionalidad cada miembro aportaba una solución determinada al problema y esto era dado porque trabajaban bajo una metodología imperativa. Esto provocaba que cada nueva funcionalidad debía ser testeada desde la concepción de su solución y complicaba establecer procesos de trabajo uniformes.

De este modo teníamos un código más difícil de mantener y por ello pasaron a una metodología más declarativa, qué hacer sin importar cómo hacerlo. Dicho con un ejemplo más sencillo es como el uso de map → no sabemos si por dentro usa un bucle for pero lo que si sabemos es cómo se usa el el resultado del mismo.

Abstracción de la implementación, como consecuencia es que ante 3 problemas podemos tener varias soluciones bajo una metodología imperativa pero en la declarativa siempre tendremos la misma solución obligando al equipo a hablar un mismo idioma.

Por ello React tiene un enfoque declarativo, es decir trabajaremos con estados, side effects … pero no tendremos que preocuparnos de cómo traer estos cambios al DOM.

Cambios en cascada

Otro problema al que se tenían que enfrentar era que ante cualquier cambio o acción lanzada se tenía que renderizar de nuevo el template y esto provocaba un bajo rendimiento.

Por ello el objetivo era bajar el número de actualizaciones o lanzarlas de una para así mejorar el rendimiento de las aplicaciones. Todo esto era imposible de realizar sin el algoritmo desarrollado denominado Virtual DOM.

DOM de Javascript

Las siglas **DOM** significan **Document Object Model**, o lo que es lo mismo, la estructura del documento HTML. Una página HTML está formada por múltiples etiquetas HTML, anidadas una dentro de otra, formando un árbol de etiquetas relacionadas entre sí, que se denomina **árbol DOM** (*o simplemente DOM*).

Virtual DOM de React

El *Virtual DOM* es una representación en memoria del *DOM de Javascript* que actúa de intermediario entre el estado de la aplicación y el DOM de la interfaz gráfica que está viendo el usuario.

Pero lo realmente maravilloso del Virtual Dom es que solamente almacena los elementos necesarios y de este modo lo hace más liviano y manejable.

Conclusión final

Es una librería de Javascript para crear interfaces de usuario, como has oído React es una librería y no un framework dejando la libertad de usarla de la manera que mejor se adapte a nuestras necesidades. Para ello vamos a aclarar la diferencia entre Librería y Framework.

¿Qué es una `librería`? Una librería es una herramienta que soluciona un problema en tu código y cubre un problema que buscas solucionar. Si se da este caso, podemos decir que tu código usa una librería.

¿Qué es un `framework`? Es algo que intenta darte una solución completa a un problema y te marca la manera en la que tienes que trabajar, es decir, tu código se desarrolla alrededor de un framework, tu no has creado el framework, pero tu código está fuertemente atado a su uso (no tiene validez fuera de el).

Por lo tanto ReactJS es una librería que te soluciona la creación de componentes en la parte de usuario, para todo lo demás puedes usar lo que quieras: ES6, ES7, TypeScript... La ventaja de esto es que tu código no depende 100% de React y su desventaja es que tienes que elegir bien el resto de opciones para completar tu puzzle sin atarte en exceso a la solución que desarrolles.

Esto nos lleva a preguntarnos qué hacer con ReactJS:

- Permite crear interface declarativas. Abstrae al desarrollador de la lógica de manipulación del DOM.
- Está basada en componentes. Pequeñas piezas intercambiables y permitiendo la agrupación de los mismos.
- Aprende una vez y usa donde quieras. Es decir podemos usar React no sólo en Web sino en aplicaciones móviles o de escritorio.
- Esto nos lleva a hablar de ReactDOM que realmente es la parte que nos permite interactuar con la navegación de usuario en las aplicaciones web.