DigitalHouse>

Certified Tech Developer
The Ultimate Degree

Front End III

1. Glosario de integración con APIs

2. Funciones puras: provienen del paradigma de programación funcional y se definen como aquellas que siempre producen el mismo resultado cuando se les dan los mismos argumentos y no producen efectos secundarios.

3. Efecto secundario (side effects): en React, es una operación invocada dentro de un ámbito o *scope* específico, pero que transcurre fuera del mismo. Debido a esto, su resultado podría ser difícil de manejar, afectando y haciendo más imprevisible nuestra aplicación, porque puede generar errores en el estado global de la aplicación.

4. Características de los efectos secundarios: violan el principio de responsabilidad única que establece que una clase o un módulo debe tener solo una razón para cambiar. En el caso de peticiones a APIs, el retorno asincrónico de datos puede provocar cambios de estado al activar acciones adicionales y asincrónicas. Sin embargo, debemos recordar que si bien los efectos secundarios de peticiones a APIs suelen ser los más recurrentes, no son los únicos que existen.

5. Manejo de efectos secundarios en React: nuestra aplicación React tiene que hacer predecibles y controlables los efectos secundarios a través de su ciclo de vida. Luego del render inicial, los efectos secundarios pasarán por distintas fases, y serán manejados a través de los métodos: componentDidMount, componentDidUpdate y componentWillUnmount.

6. Traceroute: es un comando de diagnóstico de redes para mostrar las posibles rutas o caminos de los paquetes y medir las latencias de tránsito y los tiempos de ida y vuelta a través de redes de protocolo de Internet. Permite seguir la pista de los paquetes que vienen desde un punto de red.

7. Asincronía: JavaScript utiliza AJAX para realizar llamadas asincrónicas. Estas, en general, permiten el acceso a datos mediante el objeto XMLHttpRequest (interfaz empleada para realizar peticiones HTTP y HTTPS introducida al mercado por Microsoft para Internet Explorer 5 a finales del siglo XX). AJAX facilitaba las peticiones mediante un código funcional, rápido y eficiente, con el beneficio adicional de evadir la recarga de una página para actualizar su información.

8. Request: cada vez que el cliente solicita un recurso al servidor.

9. Response: cada vez que el servidor devuelve una respuesta al cliente.

10. API (interfaz de programación de aplicaciones): es un conjunto de subrutinas, definiciones, protocolos, funciones y procedimientos, que puede ser usado por otro software como una capa de abstracción. Permite comunicar aplicaciones que no hablan el mismo idioma. Además, habilitan acceso a recursos sin descuidar la seguridad y el control.

11. APIs y React: React es compatible con cualquier biblioteca AJAX. Desde bibliotecas con soporte nativo en navegadores actuales —como Fetch— o bibliotecas third party —como Axios—. Eso permite una gran flexibilidad a la hora de desarrollar peticiones al servidor.

12. Request en React: las llamadas a APIs en un componente de clase de React se realizan durante el ciclo de vida del mismo. Estas podrán ser efectuadas luego de que el componente haya sido montado o cuando el componente se haya

actualizado debido a algún cambio en sus props o state. Es decir, utilizando los métodos componentDidMount() y componentDidUpdate(). Finalmente, para limpiar los efectos de la petición, si el componente es desmontado antes de obtener el response, usaremos componentWillUnmount().

13. Fetch: es una API JavaScript nativa para navegadores (no hay que instalarla) que permite realizar peticiones HTTP asincrónicas por medio de promises que simplifica enfoques previos basados en objetos XMLHttpRequest, con un código más legible y práctico.

14. Parámetro options en Fetch: Fetch tiene un segundo parámetro, un objeto opcional que permite configurar la petición.

Method	**Método HTTP. Por ejemplo: POST, GET, PUT, DELETE y HEAD.**
Credentials	Permite cambiar el modo en el que se realiza el request. Por ejemplo, habilitar el envío de cookies al servidor.
Headers	Son las cabeceras HTTP que permiten tanto —al cliente como al servidor— enviar información extra. Existen un gran número de opciones, podés acceder a más información en: https://developer.mozilla.org/es/docs/Web/HTTP/Headers
Body	Cuerpo de la petición HTTP. Datos que enviamos al servidor.

15. Axios: es una biblioteca third party (hay que instalarla). Es un cliente HTTP que admite la API Promise y que facilita la configuración y ejecución —tanto desde el lado del cliente como del lado del servidor— a través de un cliente HTTP basado en promesas para Node.js. En el servidor utiliza el módulo HTTP de Node.js y en el navegador usa XMLHttpRequest.

16. Sus beneficios son:

- Intercepta requests y responses.

- Transforma datos de requests y responses.
- Cancela solicitudes.
- Transforma automáticamente datos JSON.
- Brinda protección contra XSRF.

17. Instancias bases en Axios: estas permiten refactorizar la configuración en Axios para ser usada en el momento de instanciación, ahorrando la escritura de código repetitivo.
18. Algunas diferencias entre Fetch y Axios:

Fetch	Axios
Es una API nativa. No se instala.	Es un paquete de terceros. Se instala.
Sencillo de usar.	Es aún más sencillo de usar.
No funciona en IE 11.	Funciona en IE 11.
Hay que convertir los datos recibidos a JSON.	Conversión automática.
No usa XMLHttpRequest.	Usa XMLHttpRequest.
Funciona si se instala la dependencia node-fetch.	Funciona en Node.js.

¡Hasta la próxima!