Notas formador curso React 2023

Impartido por CAS-Training (Raul Lapeira) en Diciembre 2023

[**Info general 2**](#_jfjme41hi00o)

[**Pendiente/ solicitudes alumnos 2**](#_y6uk4muhb1oy)

[**Preguntas 2**](#_c4wdm3nkag64)

[Cuando se habla del back de react, ¿es realmente un back como el que tienes tú en tus desarrollos? 2](#_l7x98fv31m2f)

[Tengo una cierta funcionalidad (función flecha) que no sé si se puede usar en funciones privadas o públicas (??). ¿Se puede usar en ambos? 2](#_onlz7wlkhodg)

[¿Podemos tener un listado de scaffoldings de proyectos en React típicos? 3](#_xm6ot5sbahr)

[¿Se carga “lazy” módulo internos resueltos por nodejs igual que componentes propios en React? 3](#_uewpxna88e31)

[El import de React JSX es parecido al de EcmaScript6 3](#_9mpe3xqtfnnt)

[¿Desde swal.fire como puedo provocar una llamada a useEffect? 3](#_mi3ybi6ip1r3)

[Como pasar props del hijo al padre 3](#_a2qw36awrg1l)

[Que diferencia hay entre un outlet, children y un portal (ver en react router) 3](#_vbufwhvfjvid)

[¿Merece la pena hacer testing con React cuando podríamos hacer testing sobre el JS y HTML generado en react usando varias herramientas muy versátiles? 4](#_92u5sfw8pmrc)

[¿Alguna herramienta de diagramado te hace scaffolding y sí es así vale para algo? 4](#_49dtamenzm21)

[¿Es peor tirar de scaffolding que programar el proyecto entero? 4](#_noyy97jn7lyp)

[¿Que es mejor tener las operativas en los componentes más a bajo nivel o tener operativas en “superobjetos” y que se la pasen como props a los “subobjetos” 4](#_6dmjz695w2cz)

[¿Es Angular más estructurado a nivel de carpetas que React? 4](#_et7ysp3qws0l)

[**Conceptos 4**](#_tx7x40j1w91k)

[**Versus 6**](#_476n2zo80yy5)

[Weakly typed versus strongly typed language 6](#_6ssbyqzuyjc)

[Cohesionado versus acoplado 6](#_cgbd9zebdpi7)

[Servicio Web versus Servicio REST 6](#_6na6uv4yot2a)

[Visual studio code versus IDE 7](#_vg4evojeyjsm)

[**Link 7**](#_3bzxlmyi268s)

[**Anexo I. Batallitas del profe 7**](#_6bphceehhzc3)

# 

# 

# Info general

* Curso: React
* Horario: 9 a 14 de lunes a viernes
  + ~10:45 y ~12:45
* Nombre: Raul Lapeira
* Link al drive:
  + (Sin acceso) <https://drive.google.com/drive/folders/1XnBONcNUG1BsnPtylbDa2F5vJS2etbky?usp=sharing>

# Pendiente/ solicitudes alumnos

* Compartir por onedrive
* Pasar URL de evaluación
* **Ampliar la parte de testing y proponer soluciones de proyecto**
* **Buscar el diagrama de CORS o sino usar uno de otro curso**
* React con typescript con algún crud ya montado (es genial para diagramar)
* ~~Mirar scaffolding y~~ [~~deprecación~~](https://github.com/facebook/create-react-app)

# Preguntas

## Cuando se habla del back de react, ¿es realmente un back como el que tienes tú en tus desarrollos?

El término es genérico pero en general será un backend programado en otra cosa. React tiene algunos components que “simulan” un backend, pero siempre corre en el navegador (se puede renderizar en el servidor)

## Tengo una cierta funcionalidad (función flecha) que no sé si se puede usar en funciones privadas o públicas (??). ¿Se puede usar en ambos?

Una cosa es el ámbito de definición y resolución de “nombres” (métodos, variables, tipos,...) y otra cosa es la ejecución en el motor javascript de dichos nombres una vez se ha resuelto la signatura y el cuerpo de esos nombres. Es decir: usar “funciones flecha” no es un mecanismo de seguridad, sino una forma de escribir el código. No es como la visibilidad en java (public, private,...)

## ¿Podemos tener un listado de scaffoldings de proyectos en React típicos?

Sí te refieres a tener “plantillas tipicas de proyecto” no existe algo asi que yo sepa, lo que sí hay son varias herramientas de scaffolding con varias caracteristicas:

* <https://medium.com/@peterdtitan/create-react-app-is-dead-what-next-f05aec3dd2d5>
* <https://www.reddit.com/r/reactjs/comments/14pbzjh/is_it_safe_to_use_createreactapp/>

## ¿Se carga “lazy” módulo internos resueltos por nodejs igual que componentes propios en React?

<https://legacy.reactjs.org/docs/code-splitting.html>

## El import de React JSX es parecido al de EcmaScript6

Sí, es lo mismo

<https://stackoverflow.com/questions/36426521/what-does-export-default-do-in-jsx>

## ¿Desde swal.fire como puedo provocar una llamada a useEffect?

Swal no altera el ciclo de vida del componente envolvente, [tendrias que forzarlo a mano](https://blog.logrocket.com/how-when-to-force-react-component-re-render/)

## Como pasar props del hijo al padre

Hemos visto varios casos en los diagramas

## Que diferencia hay entre un outlet y un portal (ver en react router)

<https://reactrouter.com/en/main/components/outlet>

El outlet es específico de routing en lado de cliente con jerarquía de objetos. El portal solo permite cambiar la posición visual de un objeto en el dom.

## ¿Merece la pena hacer testing con React cuando podríamos hacer testing sobre el JS y HTML generado en react usando varias herramientas muy versátiles?

Es un tema muy personal, depende del proyecto. Yo soy más de testing funcional que además luego puedo convertir fácilmente en un test de carga pero el testing con headless y DOM/JS tiene su ámbito de aplicación también.

## ¿Alguna herramienta de diagramado te hace scaffolding y sí es así vale para algo?

No AFAIK

## ¿Es peor tirar de scaffolding que programar el proyecto entero?

Da igual si el programador usa ChatGPT, scaffolding o lo que sea, el problema es que no sepa lo que haga y eso no se resuelve quitando herramientas, se resuelven enseñando al programador para que use las herramientas de forma eficiente

## ¿Que es mejor tener las operativas en los componentes más a bajo nivel o tener operativas en “superobjetos” y que se la pasen como props a los “subobjetos”

Sí los componentes tienen que tener una visual fija y una operativa dinámica, tiene sentido pasar la operativa desde el padre para que el componente visual no tenga “hardcodeada” la lógica. Sin embargo en proyectos pequeños va a dar un poco igual y sueles ponerlo en el sitio que requiere menos código y facilite la legibilidad.

## ¿Es Angular más estructurado a nivel de carpetas que React?

No, Angular es más fuertemente tipado y SÍ es cierto que la tendencia es “pues ya que eres fuertemente tipado, organiza en carpetas”. Si lo que quieres son menos “funciones volando por ahí” y más tipado estricto entonces puedes usar React con TypeScript.

## Una duda, pasar propiedades o lambdas a los elementos hijos a veces resulta tedioso cuando tienes N hijos, ¿habría alternativas para no tener que estar haciendo esto entre muchos componentes hijos?

* Hemos visto [useContext](https://es.react.dev/reference/react/useContext) para saltar varios niveles de profundidad en los hijos (diapos del PDF)
* Hemos visto lanzar eventos que se capturan en manejadores de eventos de la clase padre (diagrama 30)
* Hemos visto usar una clase ajena a los components para llamar a los métodos de la misma ([caso practico de testing](https://github.com/scriptrance/calculator-react-jest/tree/master))
* Hemos visto definir una function e incluirla como un const en nuestro código (diagrama 32)

## Dicho de otra forma ¿que NO puedes hacer a nivel de organización de operativas?

* No puedes poner use\* fuera del ámbito de React
* No puedes usar sintaxis de React en funciones “básicas” JS
* No puedes usar effects en clases (no tiene pq ya tienes el ciclo de vida de la clase)
* Hay algunas críticas sobre el uso de lambdas con Typescript (aparentemente es un tema de linting)

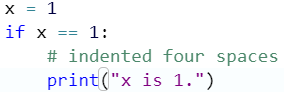
# Conceptos

* Monolítico: Un sistema que no se divide en partes
  + Del mainframe siempre se ha dicho que es monolítico
  + Del java “2 capas” (web + logica) actualmente también se dice que es monolítico
* Scaffolding: Andamiaje, usar una libreria que te simplifica el “setup” de tu proyecto
* SSR: Server side rendering, es darle cosas “ya procesadas y maquetadas” al cliente
* Escalable:
  + A nivel de rendimiento: Si tienes un servidor y das servicio a 100 clientes, al añadir un segundo servidor deberías dar servicio a casi 200
  + A nivel de requisitos: Que el dia de mañana será fácil introducir nuevos requisitos debido a que el “codebase” es fácilmente mantenible
* Functions as first class citizens: la idea de usar funciones como si fueran “tipos potentes” del lenguaje, es decir, que puedas
  + Asignarlo a una variable
  + Meterlo en un array
  + Pasarlo como parámetro
  + Componerlo con otra función
  + …
* Syntactic Sugar: Movidas del lenguaje para escribir menos (facilitarle la vida), en algunas ocasiones no es tán fácil saber si una determinada característica del lenguaje es “azúcar” o es “algo más profundo”.
  + Sería ofensivo decir que la programación orientada a objetos es azucar sintáctico
* División de responsabilidades: Es la idea de que resulta mejor tener piezas de SW especializadas en ámbitos concretos
  + React se salta la “división de responsabilidades” entre el HTML y el JS que manipula el HTML pq han visto (empíricamente) que el HTML y el JS suelen ser totalmente interdependientes
* TDD: Test driven development, cuando se hace testing programático (en código) con la finalidad de poder evaluar cómo evoluciona el proyecto cuando se hace cero seguimiento del mismo XD
* [Fluent programming / interface](https://en.wikipedia.org/wiki/Fluent_interface): Una forma de programar que consiste en ir poniendo llamadas en cascada
* One-liners de código: Una sola línea que hace más cosas de las que debería en beneficio de la legibilidad del código
* Flickering: A nivel de web se suele producir pq refrescamos la página demasiadas veces
* SPA: Single page application, una página que carga en la primera llamada y que a partir de ahí solo llama al servidor para recuperar datos “de lado servidor”
* Lazy loading: Carga diferida de contenidos, aplicado al curso puede ser
  + De componentes que [nosotros programemos en React](https://legacy.reactjs.org/docs/code-splitting.html)
  + De dependencias que React tiene sobre librerías de terceros
* Chunk: “trozaco”, aplicado a webpack es cada uno de los trozos de librerías que quieres cargar en modo lazy
* localStorage: The localStorage object allows you to save key/value pairs in the browser.
* Superobjeto: Un objeto que encapsula a otro (NO es jerarquía)
* Subobjeto: Un objeto que es encapsulado por otro
* Verbose: bla bla bla, hay un meme que dice que *“twitter no se diseño para debatir nombres de clases java, pq solo permite 128 caracteres”*
* Evento: Cualquier cosa que ocurre asíncronamente y que suele tener asociado un mensaje
* Trigger: Es lo que dispara el evento
* Operator ternario: Una forma de evitar escribir if-elses simples
* Transpilar: Pasar de un lenguaje a otro, se hace automáticamente (con SW)
  + Pasar de JSx a JS
  + Pasar de TypeScript a JS
  + Pasar de COBOL a Java
* Deuda técnica: Cantidad de porqueria que puedes poner en tu código antes de que te reviente en la cara
* Arquitectura Hexagonal

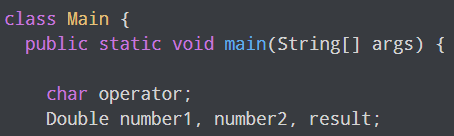
# Versus

## Weakly typed versus strongly typed language

* En un lenguaje débilmente tipado no se especifican los tipos al programar y se van pillando a medida que se declaran variables (ver también ‘dinámicamente tipados’)



* En un lenguaje fuertemente tipado, el tipo debe especificarse y NO se puede cambiar una vez creado el objeto



## Cohesionado versus acoplado

* Cohesionado: Cuando tienes módulos que cooperan bien entre sí pero uno de ellos es fácilmente intercambiable
  + Un coche está cohesionado con sus ruedas (pq gracias a que cumplen un estándar se pueden sustituir por ruedas de otro fabricante)
* Acoplado: Cuando tienes módulos que dependen completamente el uno del otro
  + Un coche está acoplado con su motor (pq el diseño del motor no es estándar y por lo tanto no puedes poner un motor seat a un coche mercedes)

## Servicio Web versus Servicio REST

* “Servicio Web” es una expresión que, a base de abusarla, ya no significa nada concreto
  + Estaban los “servicios web SOAP” que es algo muy concreto
  + Ciertos “Clouds” llaman “servicios web” a las cosas que puedes consumir (incluso sus máquinas virtuales)
* La clave de un servicio REST es que tiene que TRANSFERIR el estado del cliente al llamar al servidor, de forma que el servidor ya no tenga que mantener NADA del cliente pensando en futuras llamadas
  + Esto es muy importante desde el punto de vista de balanceo de carga en el servidor, si el estado lo tiene el cliente, no importa que se caiga un servidor pq puedes llevar al cliente a CUALQUIER OTRO servidor
  + Esto significa que montar servicios HTTP con JSON, NO implica automáticamente tener un servicio REST (depende de lo que se haga con el estado)

## Visual studio code versus IDE

* Visual studio code “pinta y colorea”, es un “rich text editor”
* Un IDE ayuda a desarrollar con numerosas utilidades importantes

# Link

* Intentar meter esto en prácticas
  + <https://medium.com/front-end-weekly/lazy-loading-with-react-and-webpack-2-8e9e586cf442>
* No se llama a useEffect y queremos forzar re-render
  + <https://blog.logrocket.com/how-when-to-force-react-component-re-render/>
* Movidas al usar lambdas en Typescript con React (pendiente revisar)
  + <https://jonhilton.net/typescript-and-react-forbidden-lambdas/>
* Testing con jest de JS normal
  + <https://medium.com/@kando1sr/testing-in-jest-building-and-testing-a-calculator-in-javascript-198afbe4a19b>
* Testing con jest y React
  + <https://github.com/scriptrance/calculator-react-jest/tree/master>

# Anexo I. Batallitas del profe

* Puede sonarnos que un cierto lenguaje tiene funcionalidades que otro lenguaje no tiene pero debido a la filosofía multiparadigma tenemos librerías para hacer casi lo mismo en Java, Scala, NET, Python y PHP