Módulo 1 – ReactJS

# ¿Qué es?

Slide 2

Es una de las librerías, creada por Facebook, mas populares de JavaScript para el desarrollo de aplicaciones web y móviles. Se fundamenta en la creación de fragmentos o bloques reutilizables llamados Componentes, para crear interfaces.

Cabe recalcar que ReactJS no es un framework como si lo son otras alternativas para la creación de web como Angular o Vue. Estos últimos definen un marco de trabajo, estructura de carpetas e interacciones entre servicios y componentes bien definidas. Por otro lado, ReactJS solo se enfoca en la renderización de los Componentes creados por el desarrollador. Dejando de libre elección el patrón de diseño o arquitectura a usar para los demás aspectos de una aplicación web como lo son API’s, Validaciones, etc.

# ¿Por qué escogerlo?

Slide 3 - 6

* Gran comunidad. Al ser el más popular, esto conlleva a que tener una gran comunidad detrás de su desarrollo, mejora y creación de nuevas librerías para agilizar la implementación de componentes. Como puede ser el caso de librerías de Mapas, Gráficos 2D y 3D, manejo de formularios, manejo de Store Global y más.
* Fácil curva de aprendizaje. Al escoger a JSX (JavaScript XML) como sintaxis para la construcción de componentes, facilita la curva de aprendizaje. Debido a que JSX es una extensión para JavaScript que le permite crear componentes que luego serán compilados y renderizados como etiquetas HTML.
* Desarrollo web y móvil (nativo). Además de aplicaciones web, permite un acercamiento más sencillo al desarrollo de aplicaciones nativas. Esto se puede lograr usando el framework Cordova, el cual permite embeber nuestra aplicación web y empaquetarla para sistemas operativas Android o iOS. Sin embargo, también tiene un enfoque más nativo llamado React Native, este último tiene ciertas variaciones en la manera de declarar y usar ciertas etiquetas, pero reutiliza las bases principales de ReactJS como lo son los Hooks, JSX y el ciclo de vida de componentes que conoceremos más adelante.

Captura de HTML vs React Component

# Creación de proyecto

Slide 8 – 10

Seguimos los pasos de la línea de comandos para la creación del proyecto base de ReactJS. Y limpiamos el proyecto para que esté igual al proyecto de la captura.

# Componentes

En la actualidad toda aplicación, ya sea web o móvil, se empieza a diseñar por secciones, páginas y bloques. Es aquí donde se empiezan a definir las bases de cada componente, sus funcionalidades, estilos e interacción con la UI o los datos de nuestra aplicación.

En ReactJS los bloques de nuestra aplicación son los Componentes, en ellos desarrollaremos, de manera declarativa, como se renderizará el componente, si hace uso o no de “props” y sus diferentes interacciones.

Diagrama de estructura básica de una página web  
Header, Navbar, Articles, Footer

# Primer componente

Slide 11 - 12

Crear un componente que renderice una imagen aleatoria desde la API <http://avatar.iran.lieara.run/public>. Utiliza el componente en el archivo de entrada de ViteJs (App.tsx) para renderizar el nuevo componente UserProfile dentro de la carpeta “components”.

Añadir props obligatorios al componente para enviar la url y el nombre. Finalmente, renderizar el componente desde un Array de objetos con las claves url y name.

## Props

Slide 13-14

Los props es un análogo de los parámetros que hemos visto en las funciones. Para enviar un valor en un prop a un componente se lo realiza como los atributos HTML antes revisado, con la diferencia de que ahora no solo pueden recibir String, si no también todos los demás valores primitivos y complejos como JSONs, Clases o incluso, otro Componente de ReactJS.

# Compilación

Slides 15 - 17

Con ReactJS, el programador escribe la aplicación web en JSX, TSX (Typescript), junto con CSS (o SCSS) para al final ver una aplicación web que es solo HTML, CSS y JavaScript. Pero

### ¿Cómo o quién realiza este proceso?

Los encargados de compilar nuestros archivos iniciales de JSX/TSX a lo que el navegador entiende son los conocidos empaquetadores o bundler. Existen una gran variedad para escoger, entre algunos están ViteJS (el que usaremos), Webpack, Create Reactc App, RSpack, Turbopack, Rollup, y muchos más.

Existen 3 pasos principalmente al momento de la compilación:

1. Compilación de JSX, indistintamente del empaquetador usado, este usa Babel que es otra librería que se encarga exclusivamente de transformar todo el JSX en funciones JavaScript equivalentes, como un React.createElement().
2. Bundling, los empaquetadores agrupan todo el código JS, CSS y, de ser el caso, los assets en muchos menos archivos y sobre todo optimizados para que el navegador pueda entenderlos. Por ejemplo, si tienen 4 archivos CSS en todo tu proyecto, al final solo tendrás uno llamado styles.css que contiene toda la información y escrito de una manera mas optimiza, reduciendo el tamaño de variables y sin caracteres especiales o comentarios.

Si bien por defecto la meta de este paso es agrupar todo en 3 archivos resultantes (uno de JS, CSS y HTML), puede darse el caso de que alguno de los archivos resultante sea muy pesado, se considera pesado si pasa los 500 kb. Aquí es cuando es recomendable hacer un Split de la estrategia del bundling del empaquetador. Esto se puede lograr en la configuración del bundlers para declarar que paquetes o componentes deseamos que estén en un archivo diferente. Así logramos tener archivos mas pequeños, lo ideal es que estos adicionales no sean de uso primario, para que solo se carguen en el momento de su uso.

1. Optimización, dentro de este paso s realizan algunas técnicas para optimizar aun m as el código y el peso final de los archivos resultantes:
   1. Minificación, eliminan espacios en blanco, comentarios, renombran variables, pero todo esto sin afectar a su funcionalidad.
   2. Tree Shaking, este paso elimina Código muerto, por ejemplo, funciones que nunca se invocan, o módulos de librerías no importadas.
   3. Code Spliting, este es el paso donde según la configuración del bundler, este hace una separación de los componentes o librerías en mas archivos, para reducir su tamaño.
   4. Lazy loading, modifica la manera de importación de componentes o librerías, con el objetivo de que se invoquen cuando el usuario las necesita.

### ¿Cómo ReactJS actualiza la UI una vez compilado?

ReactJS lleva a cabo las actualizaciones de UI o re-renders a través de su concepto de Virtual-DOM. Este es una representación en memoria del DOM previamente revisando en otros módulos del curso.

Cuando el usuario interactúa con la aplicación web ReactJS primero realiza el cambio en su Virtual-DOM, para luego comparar el Virtual-DOM con el DOM del navegador en un proceso llamado Reconciliation, en caso de existir diferencias ReactJS calcula desde cual componte fue el que cambió para actualizar así todo su árbol de componentes hijos. Así evita tener que re-renderizar componentes hermanos o superiores. Si bien se re-renderizan todos los componentes hijos de alguno que haya sido actualizado, se puede aplicar técnicas de caching o memoization para que un componente cambie solo cuando alguno de los valores que especificamos han cambiado.

# Renderizado condicional y dinámico

Slide 18

El renderizado condicional es una característica que podemos usar al usar JSX, al tener la capacidad de declarar statements de JS podemos incluir condicionales “if”, evaluaciones booleanas, y por ende condicionales ternarias.

Para realizar dentro de las etiquetas de JSX tenemos que realizarlo dentro de “{}”. Dentro de las llaves colocaremos código de JS que realizará la evaluación. En caso de ser una evaluación truthy/falsy, como puede ser user && <comp /> o Boolean (cond) && <comp />, esté hará que el componente de la extrema derecha se renderice si y solo sí toda la cadena de condiciones de la izquierda resulta truthy. Caso contrario el componente no se renderizará.

Hay que tener en consideración ciertas condiciones. Por ejemplo, al usar una variable que puede contener valores numéricos, debido a que si nuestra condición es var1 && <comp /> y el valor de var1 es el número 0, esto provocará que se renderice el numero 0, que no es lo que deseamos. Para evitar estos casos no deseados se recomienda encapsular todas las condiciones dentro del constructor Boolean() o hacer el uso adecuado del operador “!” y “!!” para obtener una respuesta completamente booleana y no truthy o falsy.