Coding Bootcamp

Sprint 4

Temario

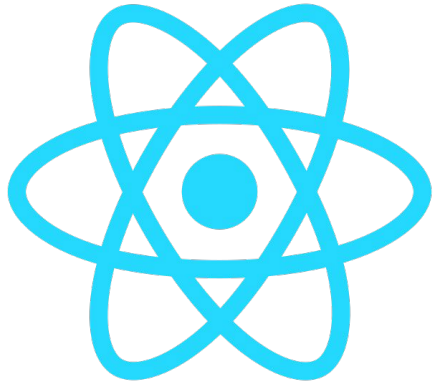
Temario

- ¿Qué es React?
- Repaso de DOM.
- jQuery vs React.
- Virtual DOM.
- Elementos en React.
- JSX.
- Componentes en React.
- React Developer Tools.
- Create React App.

¿Qué es React?

¿Qué es React? (1/2)

React es una librería JavaScript para construir interfaces de usuario (UI), desarrollada y mantenida por Facebook. Es open-source y se creó en 2013.



🧩 Está basado en el paradigma de desarrollo de componentes: pequeños bloques de código que se pueden *componer* para poder lograr interfaces de usuario más complejas.

👩‍💻👨‍💻 Cuenta con una gran comunidad de desarrolladores.

📖 Ver documentación.

¿Qué es React? (2/2)

🏞️ Se dice que React es declarativo, en lugar de imperativo. Ver detalles.

♻️⏱️ Es útil para construir aplicaciones de Front-End cuyos datos, es decir, cuyo estado, cambia frecuentemente.

📱 Además de desarrollar aplicaciones web, con React se pueden desarrollar aplicaciones móviles usando React Native. A partir de (casi) el mismo código JavaScript se pueden crear aplicaciones nativas tanto para Android como para iOS. Es decir, no son aplicaciones híbridas como las que se logran con Cordova.

Otras ventajas de React

- Es mantenido por una gran organización como Facebook.
- Cuenta con un gran ecosistema, compuesto por muchas librerías y frameworks auxiliares para utilizar en conjunto con React. Ej: Redux, React Router, Next.js, Gatsby y otras.
- Grandes organizaciones utilizan React. Además aportan a la comunidad y al ecosistema. Ejemplos: Airbnb, Netflix, Apple, Instagram, Paypal, etc.
- Es muy performante, gracias a algo llamado “Virtual DOM”.

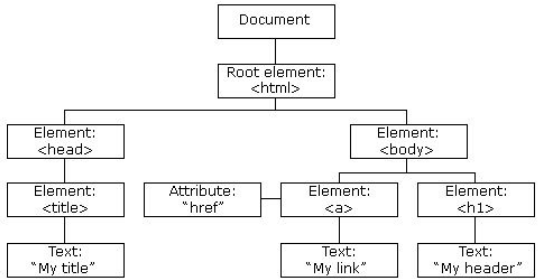
REPASO

DOM

REPASO

¿Qué es el DOM?

- DOM = Document Object Model.
  Documentación: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
- Cuando se carga una página, el browser crea un DOM de la página a partir del HTML. Es una representación del documento que tiene una estructura jerárquica con forma de árbol.
- DOM ≠ HTML.
- DOM ≈ El código que se ve en Dev Tools.
- DOM es una API (conjunto de métodos que permiten modificarlo).
- JavaScript permite modificar el DOM (pero no modificar el HTML).



REPASO

¿Qué es el DOM? – Resumen

- Es la representación que realiza un navegador de un documento HTML, como un árbol de nodos, en donde cada nodo representa una parte del documento.

- Es el conjunto de métodos que permiten modificarlo (API).

REPASO

Objeto `window`

Escribir en la consola:

```
window;
```

El *objeto* `window` representa la ventana abierta del navegador. También se le llama BOM (Browser Object Model) y permite que JavaScript interactúe con toda la ventana, no sólo con el documento HTML.

Esto permite hacer consultas, como por ejemplo, obtener el ancho de la ventana (incluyendo la *scrollbar*) o información de la ubicación (URL):

```
window.innerWidth;
window.location;
```

REPASO

Objeto `document`

Escribir en la consola:

```
document;
```

El *objeto* `document` es una referencia al documento HTML dentro de la ventana.

Es el objeto con el que interactuaremos más frecuentemente.

Es quien nos permite acceder y modificar el DOM.

Nota: También se puede acceder a `document` de esta forma:

```
window.document;
```

REPASO

Manipular el DOM con JS (1/3)

El DOM cuenta con un montón de funciones (métodos) que permiten manipularlo.

Por ejemplo, la función `querySelector` permite seleccionar un elemento de la página web.

Esto es manipulación de DOM usando JavaScript “puro”, sin uso de ninguna librería ni framework. Ej: jQuery.

```
const titulo = document.querySelector("h1");
```

Luego es posible manipular dicho elemento.
Por ejemplo, es posible cambiarle el texto:

```
titulo.textContent = "Cursos de Programación";
```

REPASO

Manipular el DOM con JS (2/3)

El código de la diapositiva anterior, también se podría haber escrito en una sola línea de código. 👉 ¡Ingresar a https://ha.edu.uy y probarlo!

```
document.querySelector("h1").textContent = "Cursos de Programación";
```

Probar también:

```
document.querySelector("p").style.color = "blue";
document.querySelector("p").style.fontSize = "4rem";
document.querySelector(".row").style.border = "10px solid red";
```

REPASO

Manipular el DOM con JS (3/3)

¿Qué se puede hacer con JavaScript y el DOM?

- Modificar todos los elementos HTML en la página.
- Modificar todos los atributos HTML en la página.
- Modificar todos los estilos CSS en la página.
- Remover elementos HTML y atributos.
- Agregar nuevos elementos HTML y atributos.
- Reaccionar a eventos que suceden en la página.

jQuery vs. React

¿Por qué no jQuery? (1/2)

Antiguamente, manipular el DOM era difícil. No existía una forma estandarizada de hacerlo, y que fuese compatible con todos los navegadores.

Por eso, en 2006 nació jQuery con el fin de resolver dicho problema, y lo logró de forma muy satisfactoria. Gracias a su facilidad de uso, durante una década jQuery fue la forma más sencilla de manipular el DOM, y hasta el día de hoy sigue siendo librería sumamente popular.

Sin embargo, los navegadores han mejorado notoriamente. Hoy en día es posible manipular el DOM usando JavaScript “puro”, de forma bastante sencilla. Gracias a CSS3 es posible hacer animaciones que antes se hacían con jQuery.

En línea con lo anterior, a partir de la versión 5, Bootstrap dejará de usar jQuery.

¿Por qué no jQuery? (2/2)

Las necesidades también han cambiado. Antes los sitios web solían ser simples páginas que mostraban información de una empresa.

Actualmente, muchos sitios web se parecen más a una aplicación de escritorio que a una página informativa. Ej: Gmail, WhatsApp Web, Google Slides, Figma, etc.

Estos sitios interactivos requieren de una gran cantidad de modificaciones del DOM. En estos casos, jQuery deja de ser útil y empieza a ser difícil de usar.

jQuery vs. React

jQuery (2006)

- Puede ser útil en páginas web sencillas, donde las modificaciones al DOM son pocas y pequeñas.
- El desarrollador debe manipular el DOM de forma directa.

React (2013)

- Es útil para crear aplicaciones web complejas – Single Page Applications (SPAs). También es cierto para Angular y Vue.js.
- Siguiendo sus patrones, se gana eficiencia y orden en el código. Es muy performante.
- Está basado en el paradigma de desarrollo de componentes.
- Manipula el DOM indirectamente. Es declarativo, en lugar de imperativo.
- También sirve para crear *mobile apps*, usando React Native.

Virtual DOM

Manejo del DOM

Tradicionalmente, era común que el desarrollador web manipulara el DOM de forma directa, por ejemplo, usando JavaScript “puro” o una librería como jQuery.

En React, la manipulación del DOM es realizada internamente, sin intervención directa del desarrollador.

El desarrollador sólo debe definir componentes (y mantener actualizado su estado). Luego, React se encarga de interpretar qué elementos del DOM precisan ser modificados, para actualizar **sólo** esos elementos. Esto hace que React tenga una gran performance.

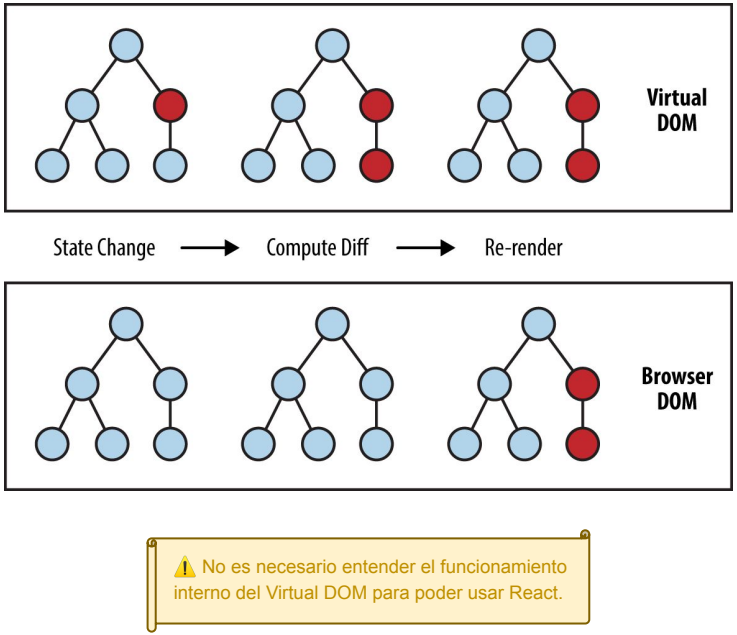
Para esto, React utiliza algo llamado Virtual DOM que es una copia del DOM pero mucho más liviana y rápida.

Virtual DOM (1/2)

Esta es una de las grandes ventajas de React: la responsabilidad del programador es sólo mantener el estado de la *app*.

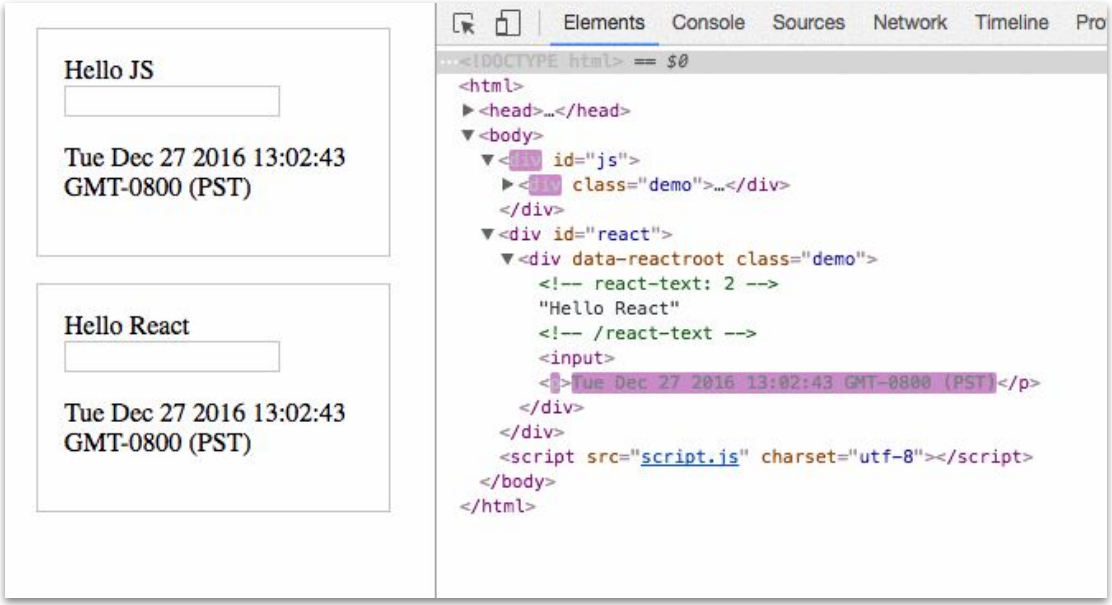
Luego, React se encarga de definir cuál es el conjunto mínimo viable de operaciones a ejecutar sobre el DOM para reflejar dicho cambio de estado.

En React no se actualiza el DOM manualmente.



⚠️ No es necesario entender el funcionamiento interno del Virtual DOM para poder usar React.

Virtual DOM (2/2) – Demo



Ver [demo aquí](#).

Elementos en React

Elementos en React (1/2)

Un elemento es un objeto JavaScript que representa un nodo del DOM. Es el “bloque” más chico en React y describe qué mostrar en pantalla.

Ejemplo de creación de un elemento:

```
import React from "react";

const elemento = React.createElement(
    "h1",
    null,
    "¡Hola Mundo!"
);
```

La función `createElement` recibe tres parámetros:

1. Etiqueta del elemento que se quiere crear.
2. Atributos del elemento (*props*).
3. Contenido del elemento (*children*).

👉 Dado que esta sintaxis no es del todo cómoda ni intuitiva de utilizar, se inventó una sintaxis llamada JSX para facilitar la creación de elementos.

Elementos en React (2/2)

Luego de crear un elemento, para mostrarlo en pantalla (insertarlo en el DOM) se utiliza la función `render`.

```
import ReactDOM from "react-dom";


ReactDOM.render(
    elemento,
    document.getElementById("root")
);
```

Este código inserta a `elemento` (definido en la diapositiva anterior), dentro de la etiqueta HTML que tenga `id="root"`. Ver demo.

Introducción a JSX

JSX

JSX es una extensión a la sintaxis de JavaScript. No es obligatorio usarlo.

Para crear un elemento, en lugar de escribir...

```
const elemento = React.createElement("h1", null, "¡Hola Mundo!");
```

...JSX permite escribir:

```
const elemento = <h1>¡Hola Mundo!</h1>;
```

Internamente, se traduce en:

⚠️ Notar que JSX no es HTML ni un string.

Pueden probar traducir cualquier código JSX a JavaScript “normal” en Babel.

JSX – Expresiones

Dentro de los *tags* de JSX es posible utilizar cualquier expresión JavaScript, ya sea una cuenta matemática o evaluar una función. Para ello se usan llaves `{ }`.

```
const nombreCompleto = usuario => usuario.nombre + " " + usuario.apellido;

const usuario = {
   nombre: "Josefina",
   apellido: "Pérez",
};

const elemento = <h1>Hola {nombreCompleto(usuario)}!</h1>;
```

Cuando este elemento se muestre al usuario, será con “Hola Josefina Pérez”.

JSX – Atributos

A los *tags* de JSX se les puede especificar atributos, al igual que sus *tags* equivalentes en HTML. La diferencia es que en JSX se escriben utilizando nomenclatura camelCase:

- `tabindex` se convierte en `tabIndex`
- `onclick` se convierte en `onClick`

```
const elemento = <div tabIndex="0"></div>;
```

Un caso a tener en cuenta: En JavaScript, `class` es una palabra reservada, por lo que para definir una clase, en JSX se debe usar el atributo `className`. 🤦

```
const elemento = (
    <h1 className="saludo">
        ¡Hola Mundo!
    </h1>
);
```

La clase “saludo” está definida en otro archivo CSS. Por ejemplo:

```
.saludo {
    color: red;
}
```

Componentes en React

Componentes en React (1/5)

React está basado en **componentes**.

Construir una aplicación en React se puede comparar a jugar con Legos, donde cada pieza de Lego es un componente (desarrollado por nosotros o creado por un tercero).

Cada componente debería ser lo más independiente posible, para favorecer la reutilización, pero también fácil de conectar a los demás. Cuanto más chicos, más fácil de mantenerlos y entenderlos.

Los componentes en React son **funciones**. Reciben un *input* (props) y retornan un *output* (lo que se debe mostrar en pantalla).

Componentes en React (2/5)

En lugar de separar artificialmente tecnologías (HTML, CSS y JavaScript) poniendo el maquetado y la lógica en archivos separados, React propone **separar intereses** en componentes.

Por lo tanto, los componentes son unidades ligeramente acopladas que contienen maquetado y lógica de negocio.

Componentes en React (3/5)

Hay dos formas de definir componentes:

1. Functional Component:

```
const MensajeBienvenida = (props) => <h1>Hola {props.nombre}</h1>;
```

`props` significa *properties*. Siempre es un objeto.

2. Class Component:

```
class MensajeBienvenida extends React.Component {
    render() {
        return <h1>Hola {this.props.nombre}</h1>;
    }
}
```

Notar que de esta forma, fue necesario hacer uso de la palabra reservada `this`.

Un Class Component debe contener un método `render`. También puede contener otros métodos.

Componentes en React (4/5)

Ejemplo de creación de un Functional Component y su *rendering*:

```
import React from "react";
import ReactDOM from "react-dom";

const MensajeBienvenida = ({nombre}) => <h1>Hola {nombre}</h1>;

ReactDOM.render(
   <MensajeBienvenida nombre="María" />,
   document.getElementById("root")
);
```

Aquí se está usando Object Destructuring de ES6.

Notar cómo se le pasan los `props` al componente.

Componentes en React (5/5)

Lo que está sucediendo en el bloque de código anterior es:

- Se llama a `ReactDOM.render()` con el elemento `<MensajeBienvenida name="María" />`
- React llama al componente `MensajeBienvenida` con las props `{ nombre: "María" }`
- Nuestro componente `MensajeBienvenida` retorna `<h1>Hola María</h1>`
- React DOM actualiza el DOM para que coincida con `<h1>Hola María</h1>` dentro del `div` con `id="root"`.

Ver demo.

Reutilización de Componentes

Reutilización Componentes (1/2)

La idea de los componentes es poder reutilizarlos. Veamos un ejemplo:

```
const MensajeBienvenida = ({nombre}) => <h1>Hola {nombre}</h1>;

function App() {
    return (
        <div>
            <MensajeBienvenida nombre="María" />
            <MensajeBienvenida nombre="Paula" />
            <MensajeBienvenida nombre="Ana" />
        </div>
    );
}
ReactDOM.render(<App />, document.getElementById('root'));
```

Los componentes se deben nombrar comenzando con mayúscula (Pascal Case). De este modo, React puede distinguirlos de las etiquetas HTML.

Un componente sólo puede retornar un elemento. Un alternativa para evitar este problema es envolver los elementos en un `<div>`.

Reutilización Componentes (2/2)

Estas dos formas también son válidas para retornar una lista de elementos, sin necesidad de crear otro nodo del DOM (como el `<div>` que se usó en la diapositiva anterior).

```
const MensajeBienvenida =
  ({nombre}) => <h1>Hola {nombre}</h1>;

function App() {
  return [

    <MensajeBienvenida nombre="María" />,
    <MensajeBienvenida nombre="Paula" />,
    <MensajeBienvenida nombre="Ana" />,

  ];
}
ReactDOM.render(
    <App />,
    document.getElementById('root')
);
```

```
const MensajeBienvenida =
  ({nombre}) => <h1>Hola {nombre}</h1>;

function App() {
  return (
    <>
      <MensajeBienvenida nombre="María" />
      <MensajeBienvenida nombre="Paula" />
      <MensajeBienvenida nombre="Ana" />
    </>
  );
}
ReactDOM.render(
    <App />,
    document.getElementById('root')
);
```

React Developer Tools

React Developer Tools (1/3)

Con los Developer Tools del navegador se puede ver el DOM con sus elementos y sus estilos. Pero al usar React, hay una jerarquía de componentes por atrás que no se puede ver.

Por esto, el equipo de React desarrolló una herramienta llamada React Developer Tools que permite ver la estructura de componentes con sus `props` y su estado en cada momento. También permite modificarlos y manipularlos de la misma manera que se puede editar el DOM con las Developer Tools del navegador.

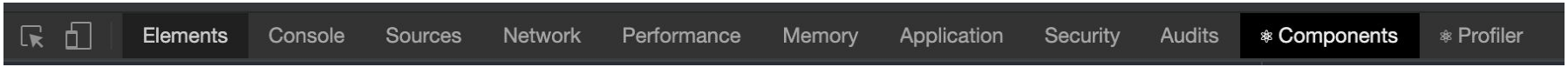
React Developer Tools (2/3)

Se pueden instalar como extensión de Chrome o Firefox.

Permiten inspeccionar árboles de componentes React.

React Developer Tools (3/3)

Las React Developer Tools agregan dos pestañas nuevas en el navegador:



La pestaña Components muestra toda la jerarquía de componentes React que han sido renderizados en la página, incluyendo todos los subcomponentes. Al seleccionar un elemento en la jerarquía se pueden modificar sus `props` y estado.

La pestaña Profiler permite grabar información de *performance* para identificar los distintos cambios que ocurren en una aplicación al interactuar con ella.

Create React App

Herramienta para crear aplicaciones de React

Proyecto base usando Create React App

Create React App (https://create-react-app.dev) es la opción más utilizada (y suele ser la recomendación) para crear proyectos en React.

1. Ejecutar el siguiente comando para crear un proyecto nuevo. Este comando se usará cada vez que se cree un nuevo proyecto:

   ```
   npx create-react-app mi-proyecto
   ```

2. Luego, desde la consola, “pararse” en la carpeta del nuevo proyecto e iniciarlo.

   ```
   cd mi-proyecto
   npm start
   ```

   En caso de que el browser no se abra de forma automática, entrar a http://localhost:3000.