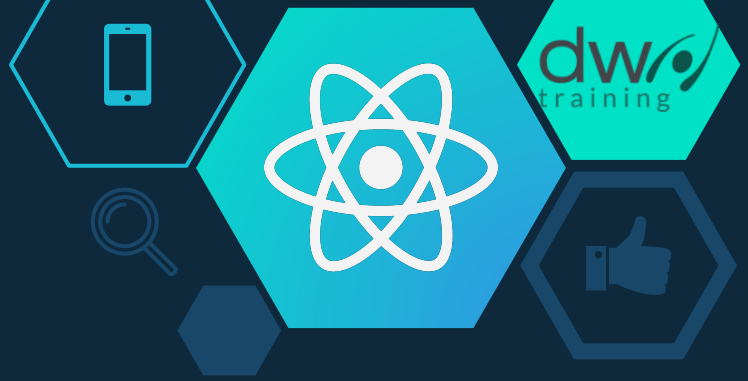


Capacitación
React JS Básico

Hello!

Soy Roque Rueda

Senior Mobile Developer

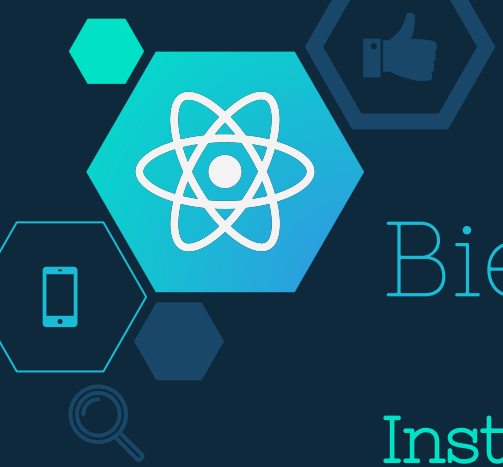
Android, iOS, react-native

6+ años de experiencia en desarrollo de aplicaciones

Presentación

Nombre

A que te dedicas



Bienvenidos a CRJS

Instrucciones

Preguntas bien recibidas

Laboratorios entre modulos

Evaluación teórica

Examen práctico

Reglas

10 min de tolerancia

2 breaks de 10 min

1 Introducción a React

Conocer react

"Programming is an art, and like any other art, you master it by practicing repeatedly." – @kapitanjakc

Contenido

- Historia de **React**
- Instalación de ambiente
- Agregar **React** a un sitio web
- Crear nuestra primera aplicación (Hello World)
- Sintaxis **ES6**
- Babel Transpiler y JSX

Photo by delfi de la Rua on Unsplash

Historia

Creado en 2011 por
Jordan Walke

Historia

- 2013 - JS ConfUS Jordan Walke presenta React y se vuelve **open source**
- 2014 - Gana reputación e inicia a ser **estable**
- 2015 - Comienza a ser utilizado por grandes **empresas**
- 2016 - React se vuelve **mainstream**

Photo by Markus Spiske on Unsplash

Instalación de ambiente

Instalar las herramientas para trabajar con react

Instalación

- **Node.js** – Entorno de ejecución de JavaScript
  https://nodejs.org/es/download/
- **Npm** – Manejador de paquetes de node, ejecutar: **npm -v** en la terminal
- **Git** – Sistema de control de versiones
  https://git-scm.com/download
  ejecutar: **git** en la terminal

Instalación

- **Visual Studio Code** – Editor ligero con soporte para JavaScript
  https://code.visualstudio.com/download
- **create-react-app** – Herramienta para crear una aplicación react. Ejecutar los siguientes comandos para instalarla:
  - **npm install -g create-react-app**

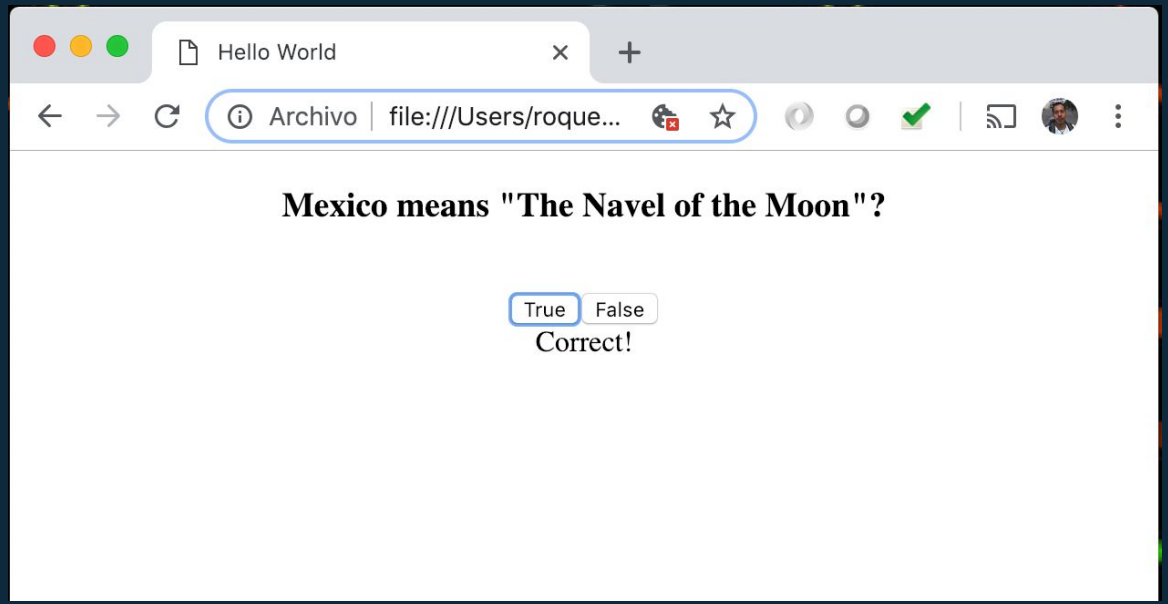
Trabajar con React

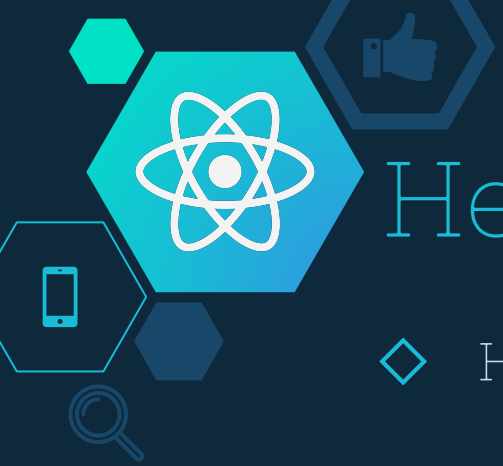
- **Agregar React a un sitio web:**
  - Adopción gradual de React

- **Crear aplicación React:**
  - Crea un aplicación a partir de una plantilla

HelloWorld

- Resultado final





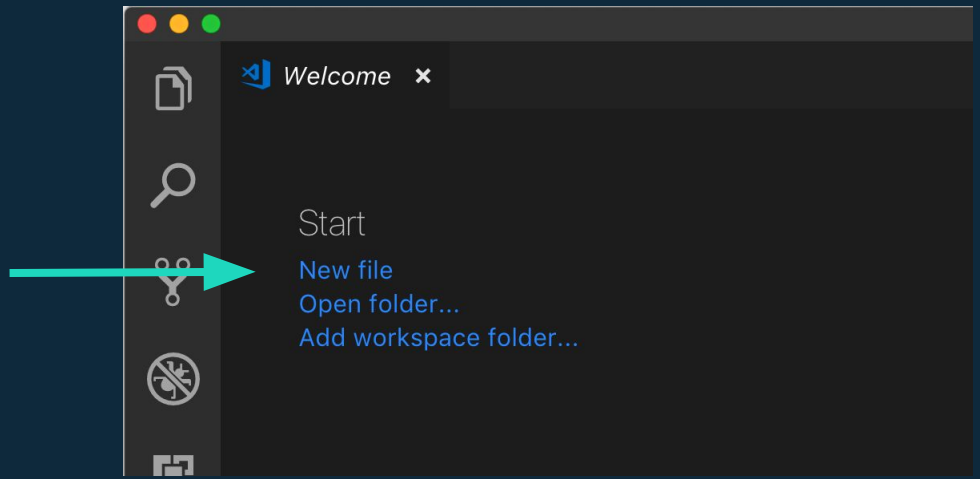
HelloWorld

- Herramientas
  - Visual Studio Code
  - HTML

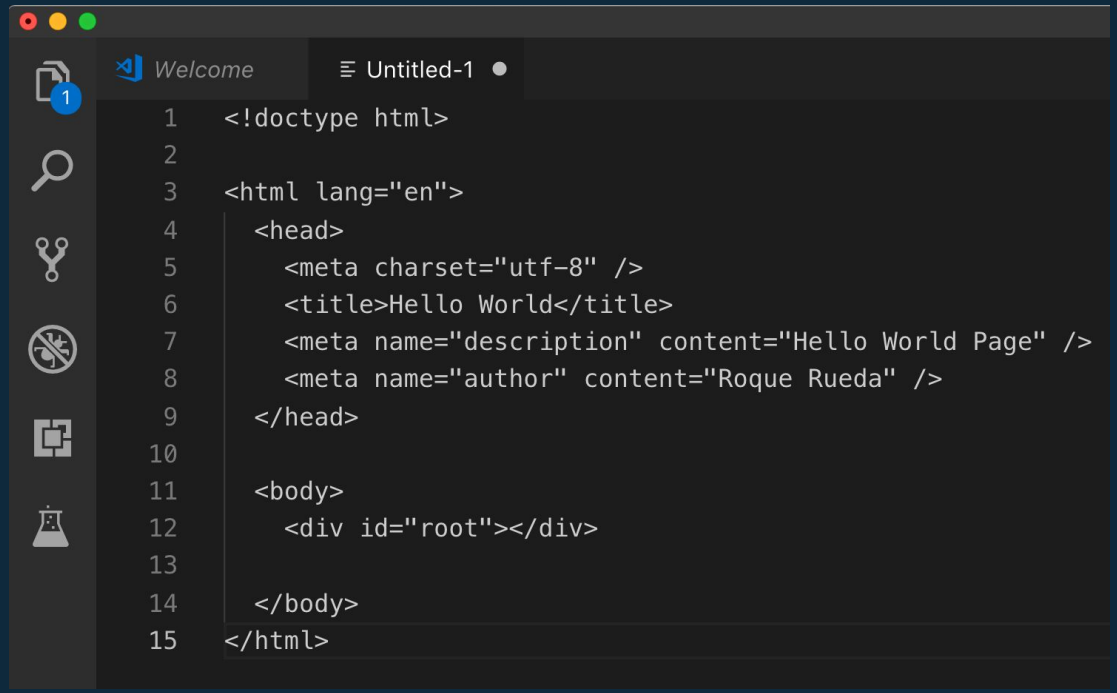
1. Crear un archivo HTML
2. Agregar React
3. Responder a interacción
4. Mostrar resultado

HelloWorld

- Crear nuevo archivo en VSCode
  - En la ventana de bienvenida seleccionar **New file**



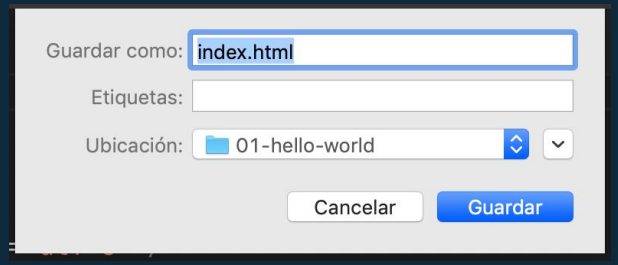
HelloWorld



HelloWorld

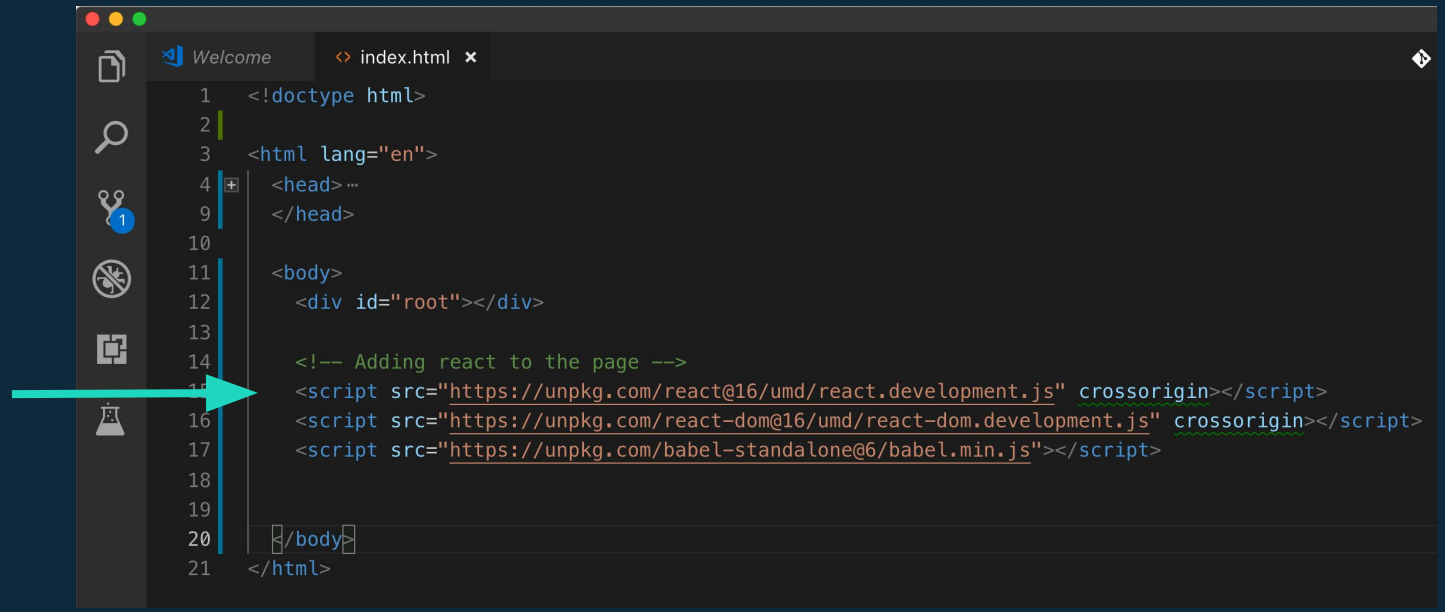
Guarda nuestro archivo

1. Ctrl + S o CMD + S
2. Menu File -> Save



HelloWorld

Agregar React



HelloWorld

Agregar una clase

```
19      <script type="text/babel">
20        class App extends React.Component {
21          constructor(props) {
22            super();
23          }
24
25          render() {
26            return (
27              <div>
28                <h3>Mexico means "The Navel of the Moon"?</h3>
29              </div>
30            );
31          }
32        }
33
34        ReactDOM.render(<App />, document.getElementById('root'));
```

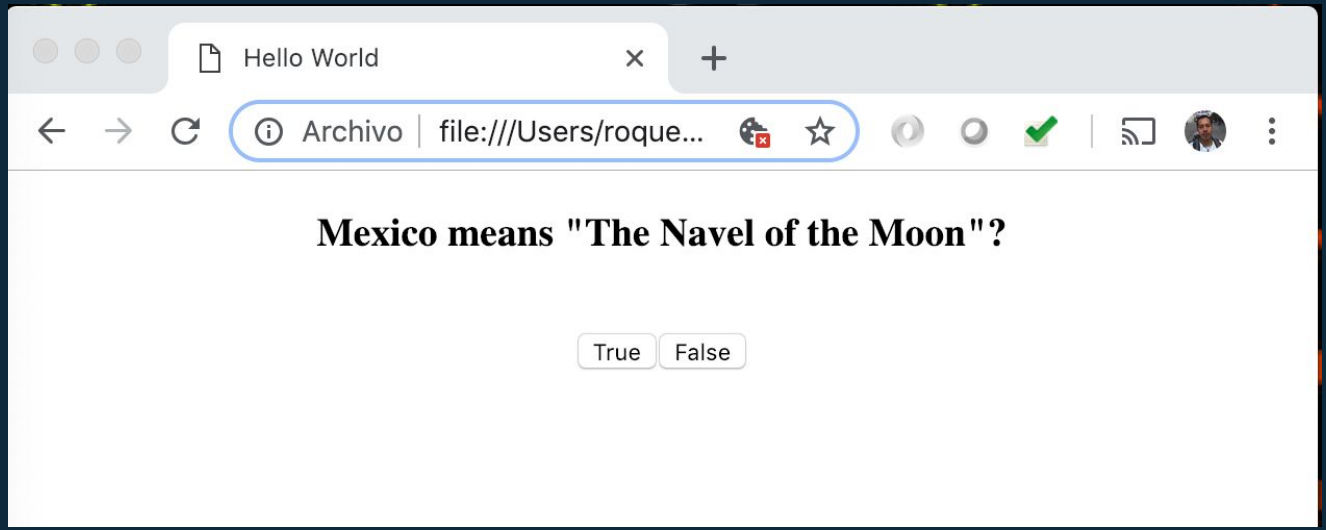
HelloWorld

HelloWorld

JSX – Metodo **render**

```
render() {
  return (
    <div>
      <center>
        <h3>Mexico means "The Navel of the Moon"?</h3>
        <br/>
        <button onClick={this.onTrueClick}>True</button>
        <button onClick={this.onFalseClick}>False</button>
        <br/>
      </center>
    </div>
  );
}
```

HelloWorld



HelloWorld

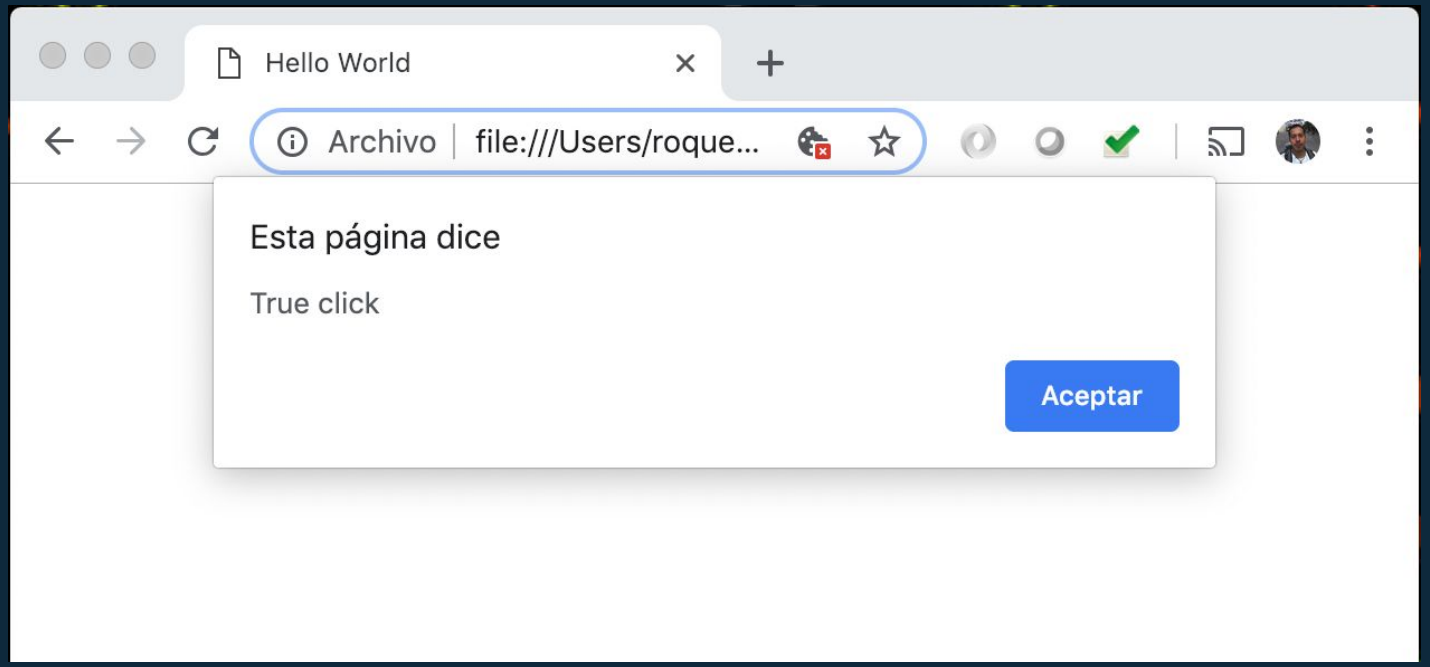
Funciones flecha

```
onTrueClick = (event) => {
  alert('True click');
}

onFalseClick = (event) => {
  alert('False click');
}

render() {
```

HelloWorld



HelloWorld

Estado del componente

```
class App extends React.Component {
  constructor(props) {
    super();
    this.state = { msg: '' };
  }

  onTrueClick = (event) => {
    this.setState({ msg: 'Correct!' });
  }

  onFalseClick = (event) => {
    this.setState({ msg: 'Incorrect!' });
  }
```

HelloWorld

Interpolacion JSX

```
render() {
  return (
    <div>
      <center>
        <h3>Mexico means "The Navel of the Moon"?</h3>
        <br/>
        <button onClick={this.onTrueClick}>True</button>
        <button onClick={this.onFalseClick}>False</button>
        <br/>
        {this.state.msg}
      </center>
    </div>
  );
}
```

HelloWorld

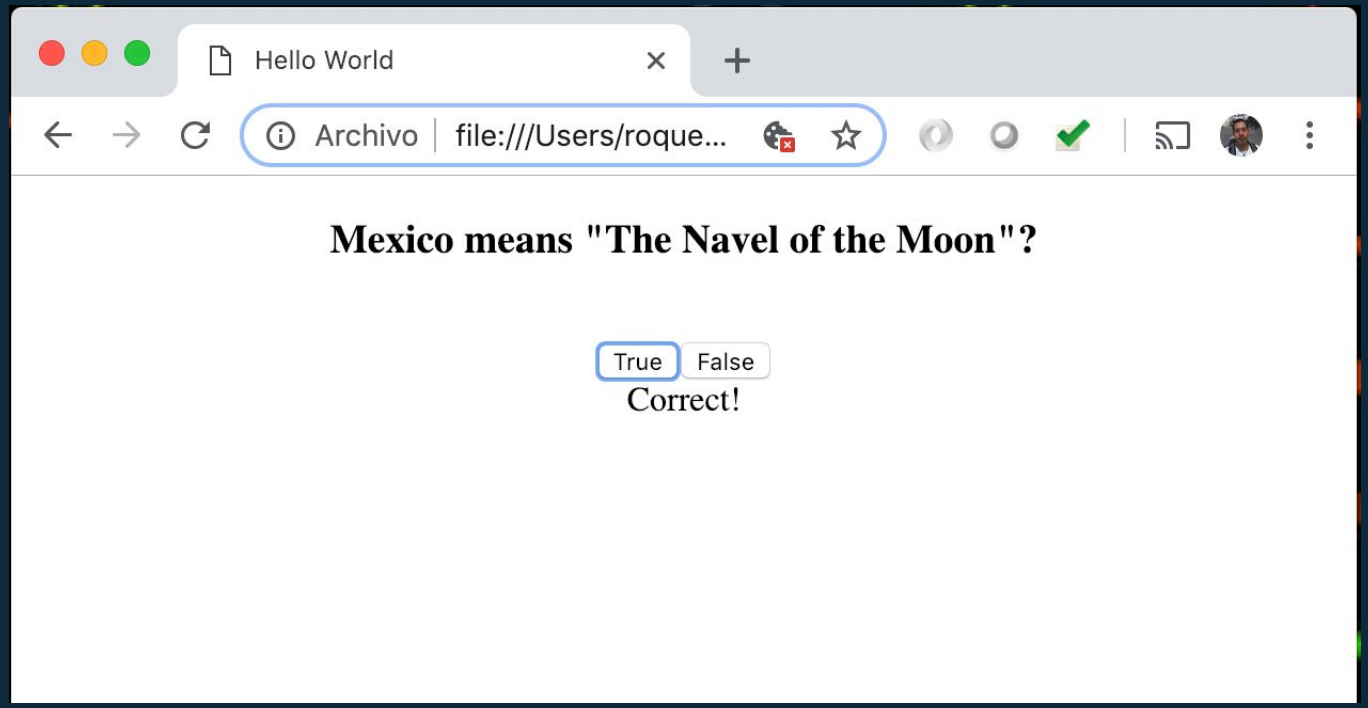
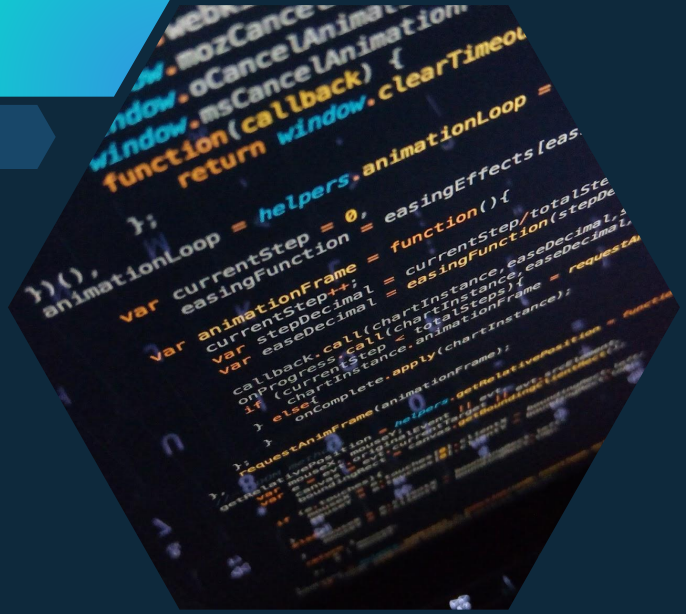


Photo by Irvan Smith on Unsplash



ES6, ES2015-6

Acrónimos para referirse a las versiones más recientes del Estándar de Especificación del Lenguaje ECMAScript.

ECMAScript

JavaScript

Es una implementación del estándar ECMAScript

ES6

La versión ES6 incluye muchas adiciones como: arrow functions, classes, template literals, sentencias let y const.

Photo by Fabian Grohs on Unsplash

Babel js

Compilador de JavaScript utilizado mayormente para convertir código ECMAScript a versiones anteriores de JavaScript.

Como funciona Babel

Gera el código que sea compatible con navegadores antiguos y actuales

```
// Código ES5
[1, 2, 3].map(
  (n) => n + 1
);
```

Babel

```
// Equivalente ES5
[1, 2, 3].map(
   function(n){
     return n + 1;
  });
```

JSX

Extension de sintaxis de JavaScript

JSX

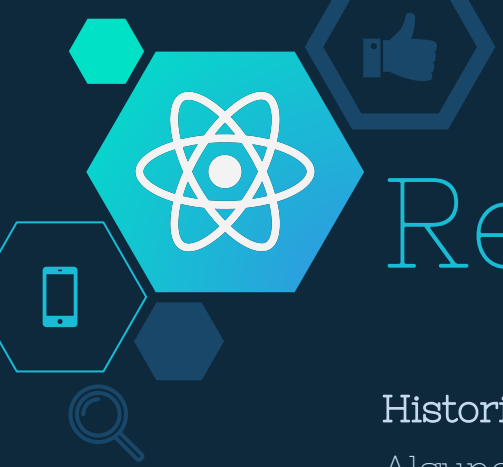
Creado para generar nuestra **Interfaz de Usuario (UI)** de una forma más familiar

- Separa cada responsabilidad con unidades llamadas componentes que contienen la lógica para presentar información junto con la lógica adicional como lo es eventos, cambios de estado y formateo de datos

JSX

React no requiere el uso de **JSX** pero:

- Brinda una ayuda visual
- Mejor manejo de errores
- Despliegue de mensajes de alerta



Revisión del módulo

Historia de React

Algunos de los datos históricos relevantes de react

Instalacción

Herramientas que se utilizaran en el curso

Agregar React

Hello World agregando react a un sitio web

ES6

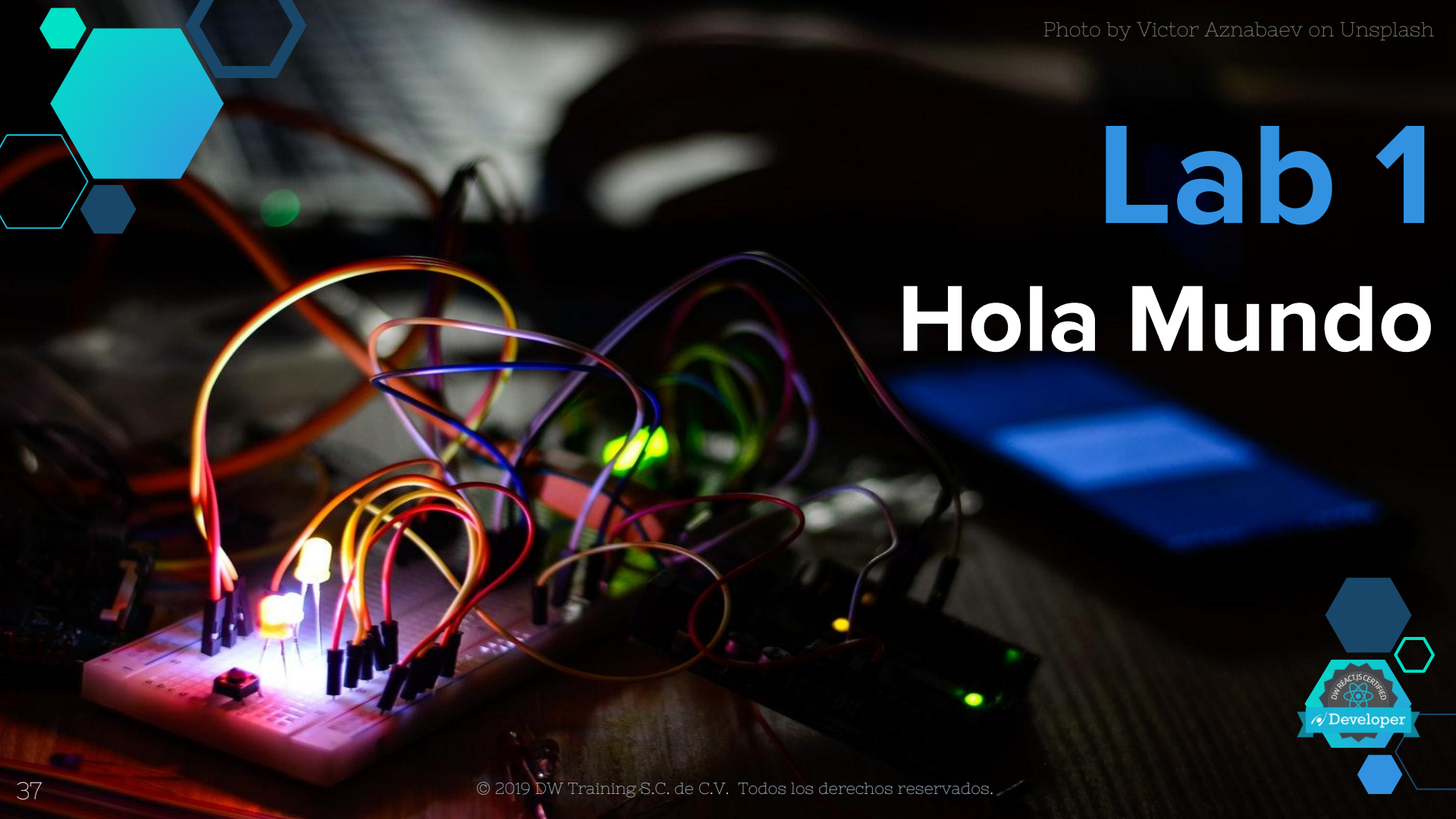
Sintaxis como arrow functions de ECMAScript

Babel

Compilador de JavaScript

JSX

Extension de JavaScript



Lab 1

Hola Mundo

2

Creación de elementos utilizando React API

Utilizando el API de React

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.” – Martin Fowler

Contenido

- Funcion **React.createElement**
- Composición de componentes
- Envío de **propiedades**
- Funcion **React.render**

Photo by Ashim D'Silva on Unsplash

React.createElement

Crea y retorna un elemento del tipo solicitado

React.createElement

React.createElement

- **type** – Se indica el nombre de la tipo que se desea generar puede ser el nombre de la etiqueta (div o span)
- **props** – Son los valores que se envían al constructor del componente
- **children** – Los elementos “hijos” que serán contenidos dentro del componente

React.createElement

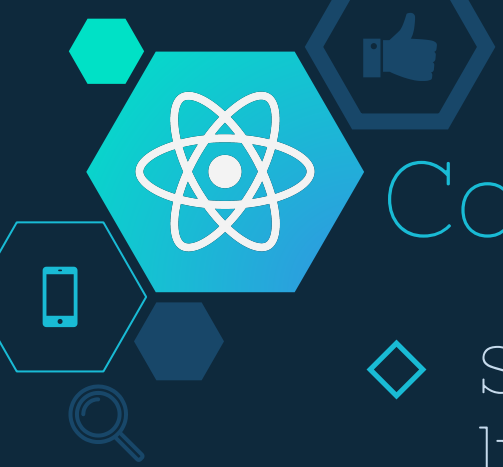
```
// No invocamos a createElement
React.createElement('div',
  { className:'title' }, 'Hello World');

// JSX se convierte a createElement
<div className="title">Hello World</div>
```

Photo by rawpixel on Unsplash

Composicion

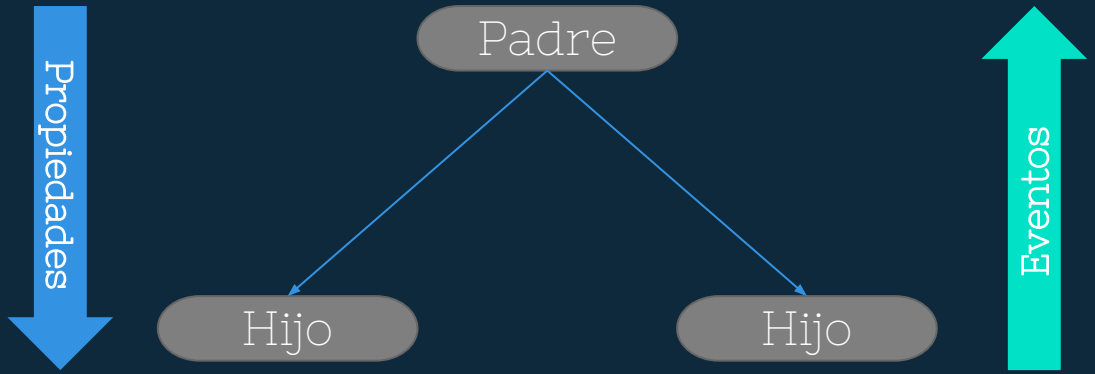
Elementos que contiene a su vez otros elementos que implementan funcionalidades deseadas



Composicion

- Se recomienda la **composición** en lugar de la herencia
- Jerarquía de componentes especializados
- Mediante las **propiedades** y los **hijos** del componente se obtiene la flexibilidad necesaria

Composicion



Composicion

```
<div>
  <center>
    <h3>Mexico means "The Navel of the Moon"? </h3>
    <br/>
    <button onClick={this.onTrueClick}>True</button>
    <button onClick={this.onFalseClick}>False</button>
    <br/>
    {this.state.msg}
  </center>
</div>
```

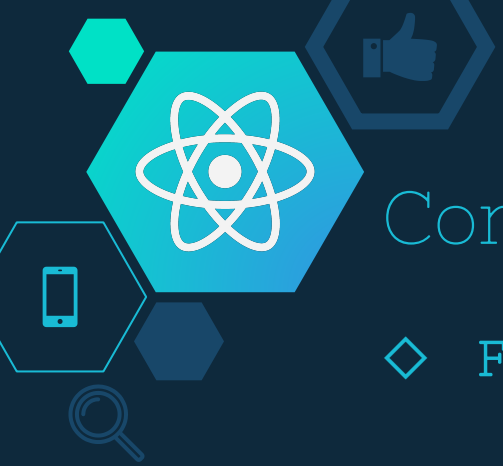
Photo by rawpixel on Unsplash

Propiedades

Los componentes reciben valores de entrada y retornan elementos React que describen lo que se muestra en pantalla

Propiedades

- Son de **solo lectura** es decir no se pueden modificar
- Se pueden definir como atributos en JSX
- Cada **atributo** JSX será una llave definida el objeto **props**



Componentes función y de clase

- Function components
  - Funcción de JavaScript:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- ES6 class
  - Clase que extiende de React.Component

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

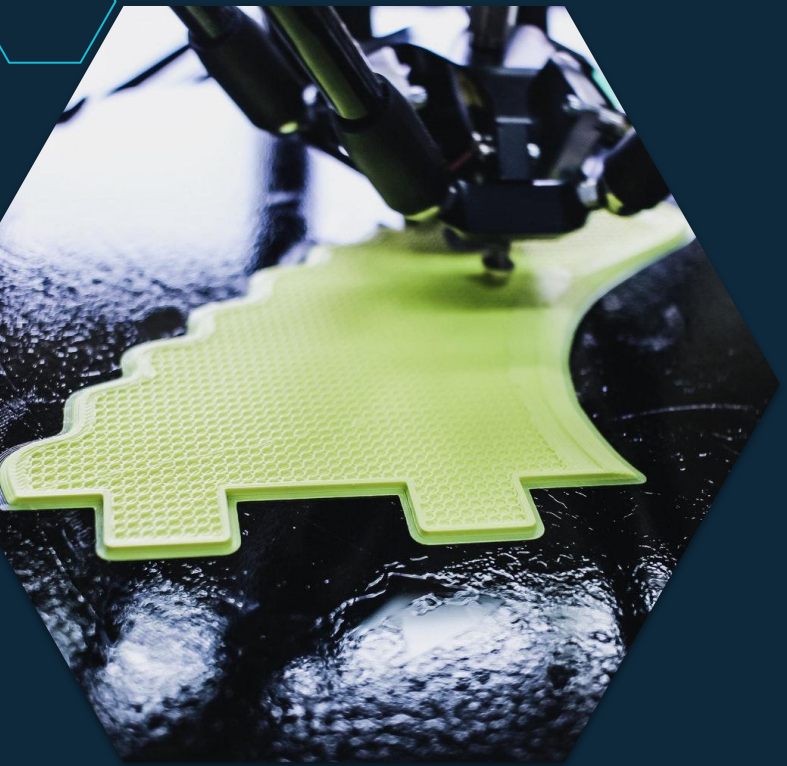
Componentes función y de clase

Enviar props a un componente

Para enviar las propiedades mediante JSX se utilizan los atributos los cuales se agregan como llaves al objeto props.

```
const element = <Welcome name="Sara" />;
```

Photo by Ines Álvarez Fdez on Unsplash



ReactDOM.render

Crea elementos en el Document Object Model y retorna una referencia hacia el componente

ReactDOM.render

ReactDOM.render

- Muestra el **elemento** en el **contenedor** especificado
- Si un **elemento** ya fue renderizado en el contenedor, solo se **actualizan** los elementos DOM necesarios
- Si se envía un **callback** opcional se ejecuta después de que el componente fue renderizado o actualizado

ReactDOM.render

- Los elementos contenidos en el contenedor son **reemplazados** la primer vez que se invoca la función
- No modifica el nodo **contenedor**
- Retorna la referencia al Componente React padre, sin embargo el uso de este valor no es **recomendado**

Revisión del módulo

createElement

Crea los elementos react que describen cómo lucirá la pantalla

propiedades

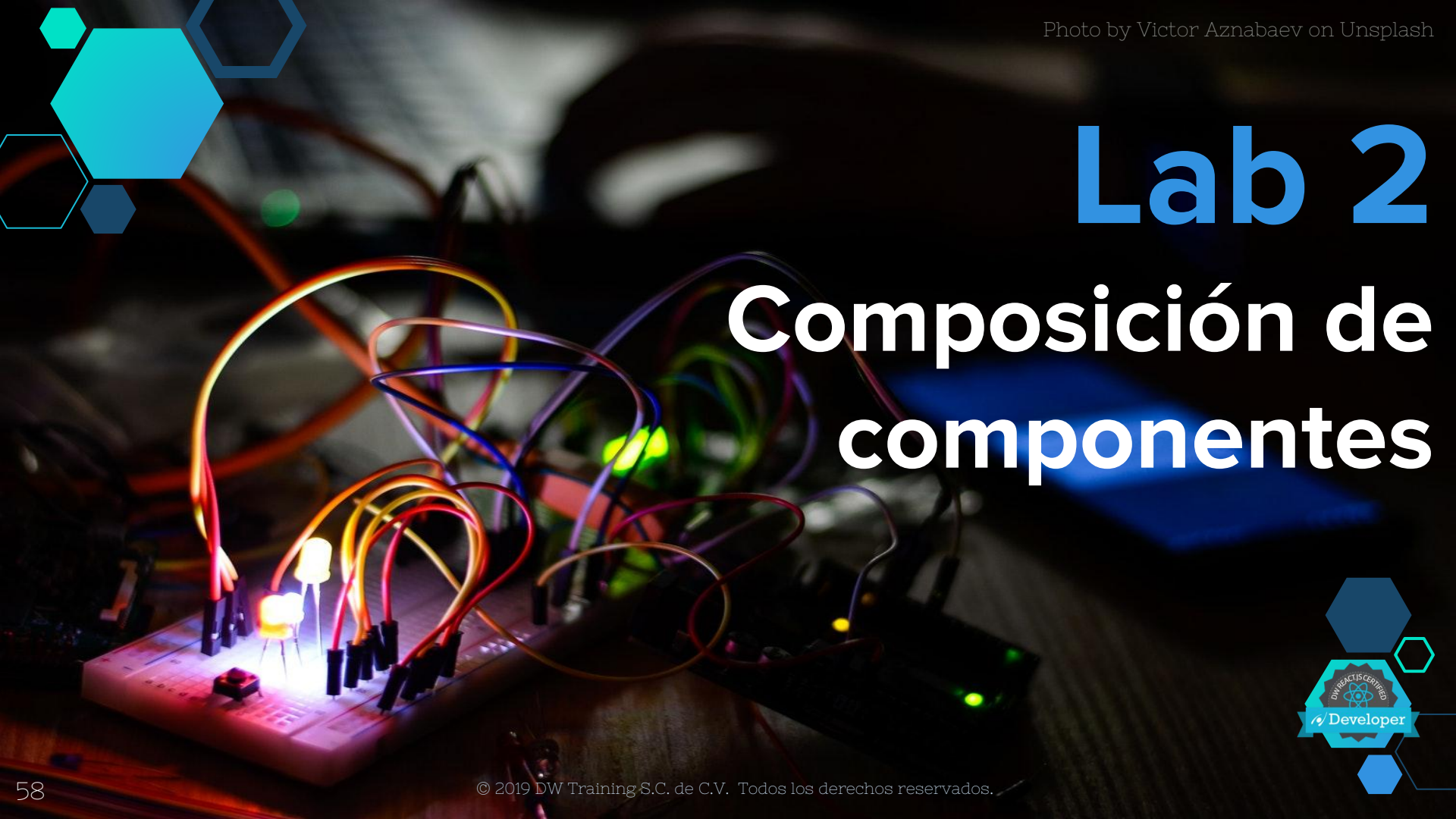
Datos de entrada que recibe un componente

Composicion

Anidar diferentes componentes para conformar un elemento

ReactDOM.render

Sintaxis como arrow functions de ECMAScript



Lab 2

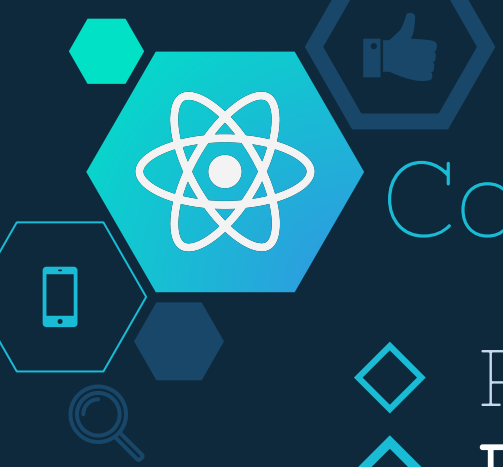
Composición de componentes

3

Composición vs Herencia

Reutilizar código utilizando composición

"The best error message is the one that never shows up." - Thomas Fuchs



Contenido

- Pensando en **React**
- **Propiedades y Estado**
- Jerarquía de componentes

Crear grandes aplicaciones Web de forma rápida

Pensando en React

Paso 1
Descomponer la UI en componentes

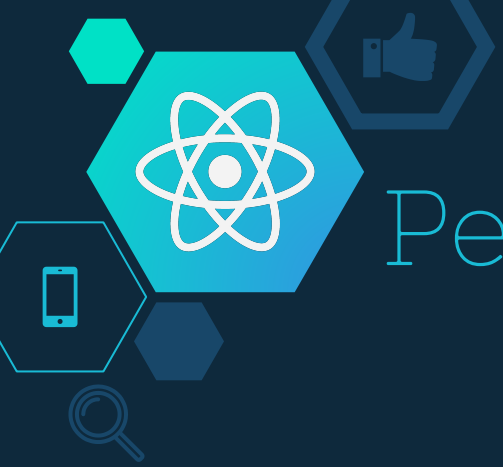
Paso 2
Crear una versión estática de la UI

Paso 3
Representar el Estado

Una sola responsabilidad
Idealmente cada componente debe realizar solo una acción

No requiere uso del estado
Top-down para proyectos pequeños
Button-up para proyectos grandes

No repetir
Determinar las piezas de datos con las que se podría trabajar

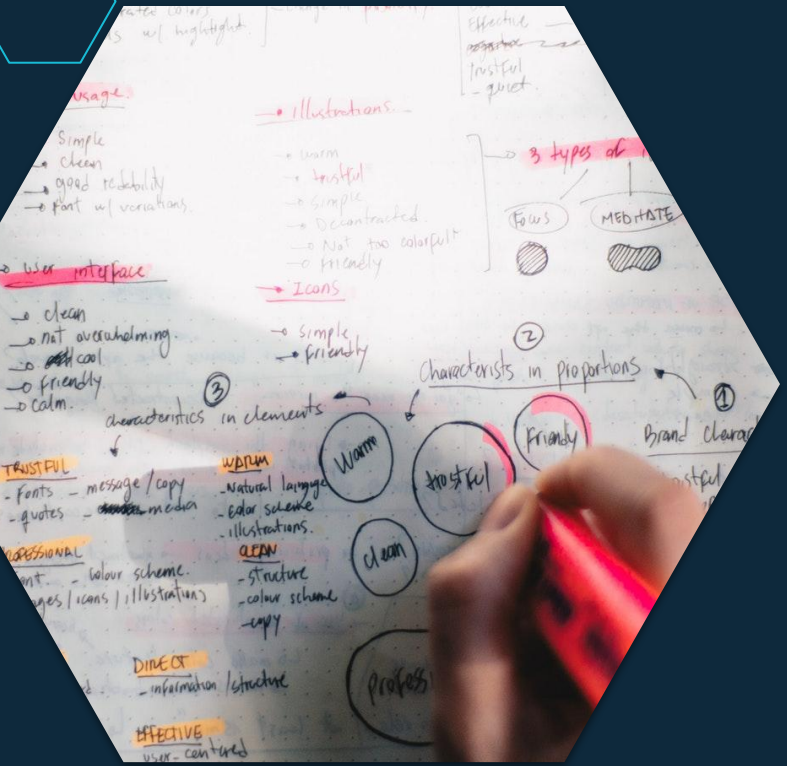


Pensando en React

Paso 4
Determinar el lugar donde vive el estado

El responsable del estado en la jerarquía
Que componente ser el responsable de mantener el estado y compartirlo en caso de ser necesario

Photo by Chris Knight on Unsplash



Propiedades y estado

Dos formas de modelo de datos en react **props** y **state**.

Propiedades

- Componentes puede aceptar valores de entrada llamados **props**
- Cuando React encuentra un componente definido por nosotros envía todos los **atributos** en JSX como un solo elemento llamado props
- Solo lectura y no deben intentar modificarse dentro del mismo componente.

Propiedades

- Pueden contener varios tipos de datos (array, object, number, function entre otros)
- Pueden ser validados utilizando **propTypes**
- Las propiedades de un componente son similares a los atributos en **HTML** de las etiquetas.

Propiedades

```
function Welcome(props) {
 return <h1>Hello, {props.name}</h1>;
}

class Welcome extends React.Component {
 render() {
   return <h1>Hello, {this.props.name}</h1>;
 }
}
```

Estado

- El **estado** es el corazón del componente y determina tanto su comportamiento como su visualización
- Estado representa la información del componente, si el estado de un componente **cambia** el componente se **renderiza** nuevamente
- El estado es administrado por el componente

Estado

- El estado **inicia** con un valor por defecto cuando el componente es montado
- El estado de un componente es **privado**
- El estado es **opcional**

Estado

```
class Button extends React.Component {
 constructor() {
   super();
   this.state = { count: 0 };
 }
 updateCount() {
   this.setState((prevState, props) => {
     return { count: prevState.count + 1 }
   });
 }
```

Estado

```
render() {
  return (
    <button onClick={() => this.updateCount()}>
      Clicked {this.state.count} times
    </button>);
 }
}
```

Propiedades y Estado

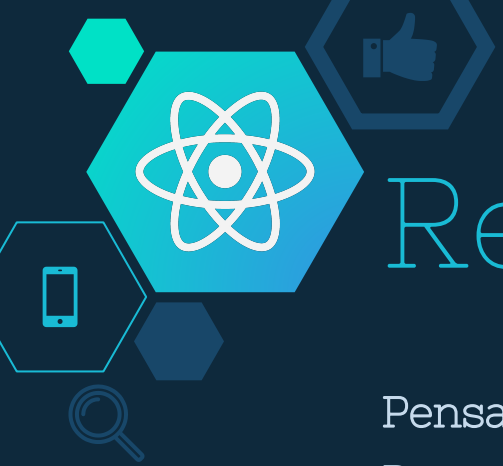
Stateless y Stateful

- **Stateless** – Solo recibe propiedades y no estado usualmente solo son encargados de la visualización y formateo de datos
- **Stateful** – Ambos propiedades y estado contiene procesamiento y responden a eventos del usuario

Jerarquía de componentes

Debido a que generamos jerarquía de componentes vale la pena identificar aquellos componentes que mantendrán un estado y aquellos que solo reciben propiedades

- ¿La información es asignada por el componente padre? => **props**
- ¿La información será creada y modificada por el componente? => **state**



Revisión del módulo

Pensando en react

Describe la forma de trabajo al crear una aplicación React

Propiedades y Estado

La forma en que React trata los datos en un componente

Jerarquia

Forma de construcción de elementos dentro de una aplicación

Lab 3

Pensando en react

4

Ciclo de vida de un componente

Eventos que ocurren en momentos específicos de la aplicación

“Good code is its own best documentation. As you're about to add a comment, ask yourself, "How can I improve the code so that this comment isn't needed?" Improve the code and then document it to make it even clearer.” – Steve McConnell

Contenido

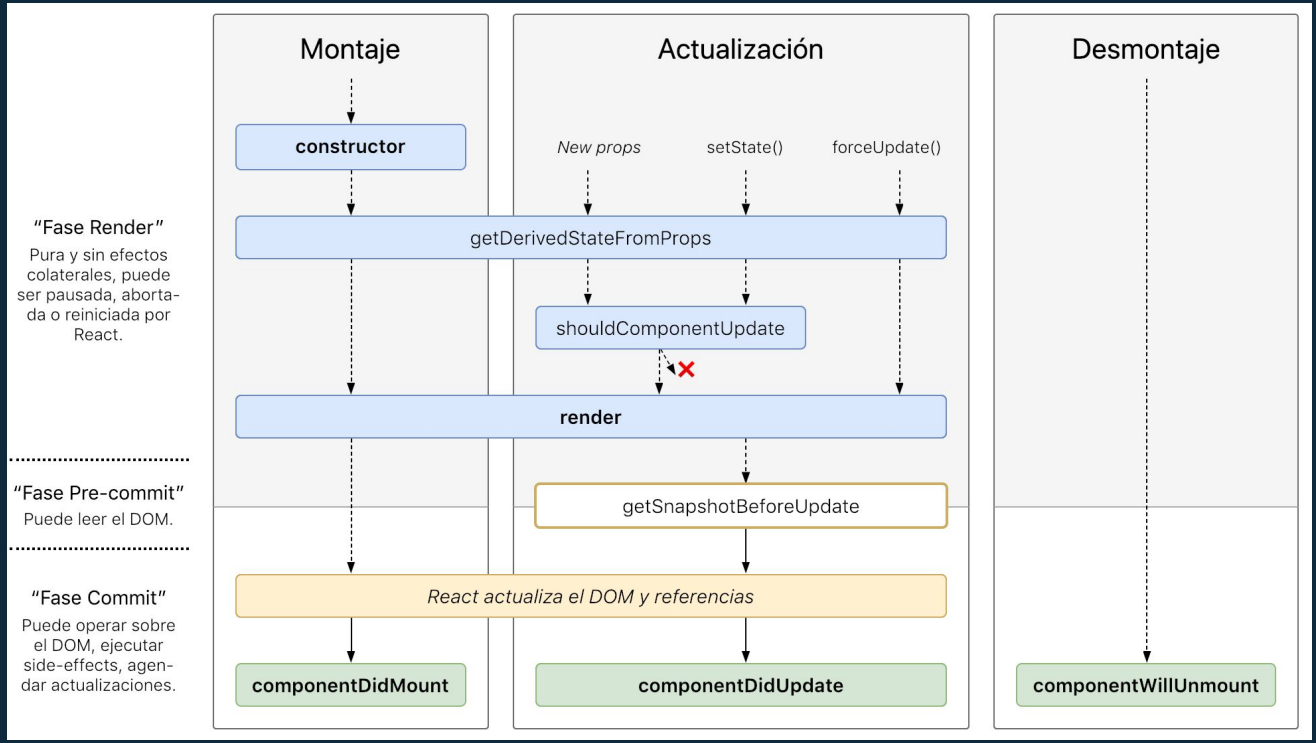
- Orden de eventos del **ciclo de vida**
- Class components y functional components
- Cambio de propiedades y estado
- Validación de **propiedades**

Photo by NEW DATA SERVICES on Unsplash

Orden de eventos del ciclo de vida

El orden de las fases en las que se ejecuta una aplicación React

Ciclo de vida



Ciclo de vida

Montaje

1. Construcción de componente
2. Se llama **getDerivedStateFromProps**
3. Se ejecuta el método **render**
4. Actualización de DOM y referencias
5. Se ejecuta **componentDidMount**

Ciclo de vida

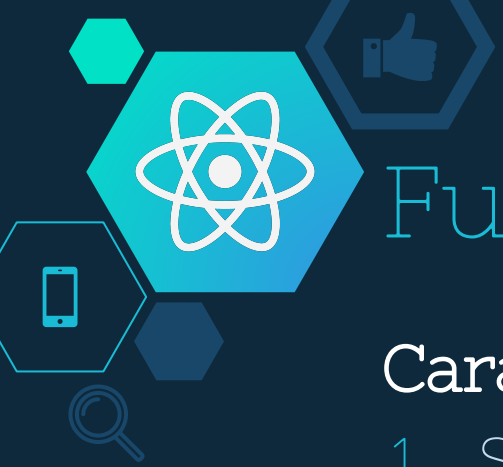
Actualización

1. Se invoca shouldComponentUpdate
2. Se ejecuta el método **render**
3. Se llama **getSnapshotBeforeUpdate**
4. Actualizar DOM y referencias
5. Se ejecuta **componentDidUpdate**

Ciclo de vida

Desmontaje

1. Se ejecuta **componentWillUnmount**



Functional components

Caracteristicas

1. Sencillos
2. Funciones puras
3. Representar y formatear datos

Class components

Caracteristicas

1. Tienen acceso al estado
2. Tienen acceso al ciclo de vida del componente
3. Facilitan la interacción a los eventos del usuario

Cambio de propiedades

Solo lectura

Un componente nunca debe modificar sus propiedades

Si un componente padre invoca a render las propiedades pueden ser enviadas nuevamente al componente hijo

Cambio de estado

El estado es privado

Debido a que el estado es privado, el componente está encargado de su actualización

La función setState es asíncrona por lo que sí requerimos de valores previos del estado debemos usar un callBack

Validación de propiedades

Para la validación de las propiedades el equipo de React nos ha brindado algunas funciones para realizarlo

Para utilizar dichas funcione se debe importar la dependencia ‘prop-types’

Se debe colocar la llave y su correspondiente tipo

Validación de propiedades

```
import PropTypes from 'prop-types';

CommentRow.propTypes = {
 avatar: PropTypes.string.isRequired,
 author: PropTypes.string.isRequired,
 time: PropTypes.string.isRequired,
 comment: PropTypes.string.isRequired
}
```

Validación de propiedades

Revisión del módulo

Eventos de ciclo de vida

Son puntos de ejecución de la aplicación

Class componentes y functional components

Diferencias en su uso

Propiedades y estado

Modelos de datos de React

Validación de propiedades

Permite asegurarnos que los valores de entrada sean correctos

Preguntas Modulo 4

Entrar a
kahoot.it

Game
PIN:
635437

Iniciar
el juego

5

Crear una aplicación React

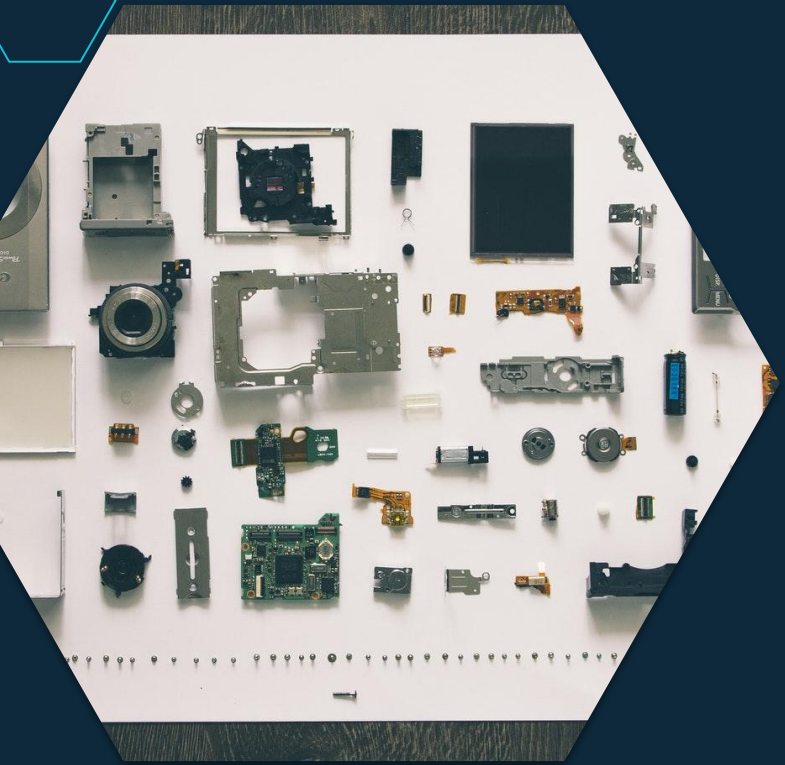
Crear una aplicación que utilice todas las características de React

"DRY – Don’t Repeat Yourself – Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." – Andy Hunt & Dave Thomas

Contenido

- Partes de una aplicación
- Creación de **formularios**
- Dando **estilo** a nuestros componentes
- Actualizando **estado** del componente

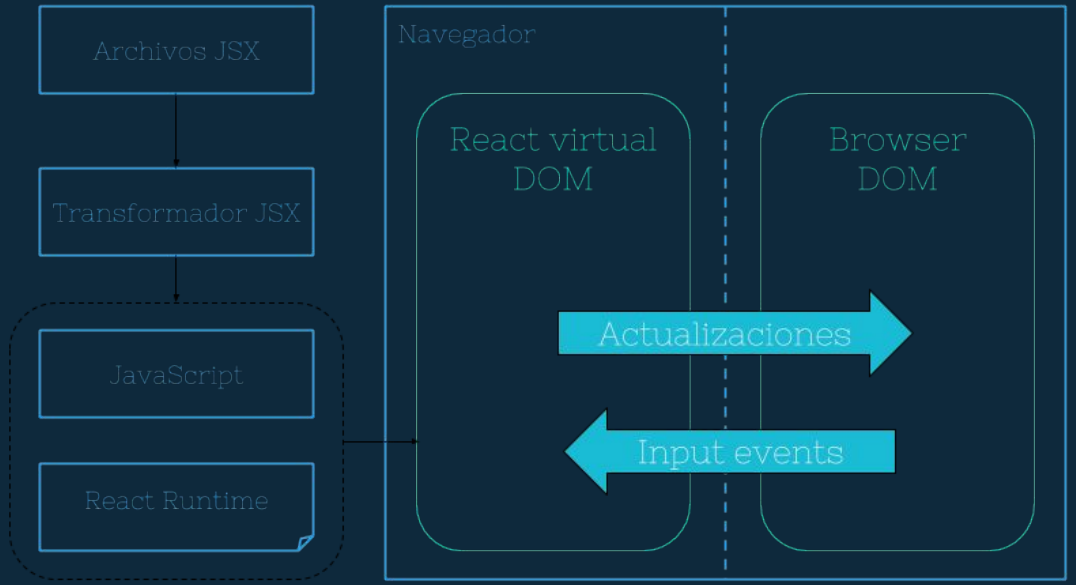
Photo by Vadim Sherbakov on Unsplash



Partes de una aplicación

Conjunto de elementos que conforman la aplicación

Partes de una aplicación



Partes de una aplicación

Componentes alrededor de los datos

En una aplicación React es importante entender que cada componente es un envoltorio de los datos que lo componen y representa los mismos en pantalla

Partes de una aplicación

Flujo de datos unidireccional

La información siendo una parte crucial de toda aplicación en React sigue un flujo unidireccional, de componentes padres a hijos.

Formularios

Formularios mantienen un estado interno

En React queremos manejar el estado de todos los componentes para tener una sola fuente de verdad (**controlled inputs**)

Formularios

```
<input
  value={this.state.term}
  onChange={event =>
    this.onInputChange(event.target.value)}
/>
```

Photo by Element5 Digital on Unsplash

Dando estilo a nuestros componentes

Los estilos en React no se definen como cadenas sino como objetos JavaScript

Estilos

Atributo style

El atributo style de JSX nos permite especificar estilos de nuestro componente mediante un objeto JavaScript usualmente en cada nombre del estilo será una llave del objeto en camelCase

Estilos

Ejemplo de estilos inline

```
const divStyle = {
  color: 'white',
  backgroundImage: 'url(' + imgUrl + ')'
};

<div style={divStyle}>Hello World!</div>
```

Estilos

Atributo className

El atributo para especificar la clase CSS de un elemento JSX

Se pueden tener elementos con **className** y con estilos **inline**

Estilos

Ejemplo de uso de CSS

```
App {
  text-align: center;
}
```

```
<div className="App">Hello World!</div>
```

Photo by Fredy Jacob on Unsplash

Actualizando el estado del componente

El estado debe ser actualizado siguiendo reglas en React

Actualizar el estado

No modificar el estado directamente

Al asignar el estado de forma directa React no renderiza nuevamente el componente

```
// Wrong
this.state.comment = 'Hello';

// Correct
this.setState({comment: 'Hello'});
```

Actualizar el estado

Puede ser asincrono

```
// Wrong
this.setState({
 counter: this.state.counter + this.props.increment,
});
// Correct
this.setState((state, props) => ({
 counter: state.counter + props.increment
}));
```

Actualizar el estado

Se pueden tener múltiples llaves y actualizar cada una de ellas por separado

```
this.state = { posts: [], comments: [] };
```

```
this.setState({ posts: response.posts });
```

Revisión del módulo

Partes de una aplicación

Los datos y su representación para formar una aplicación

Creación de formularios

Controlled components deben ser manejados por React

Dando estilo a nuestros componentes

Estilos inline y el uso de CSS

Actualizando estado de componente

El estado del componente debe ser afectado utilizando setState

Lab 5

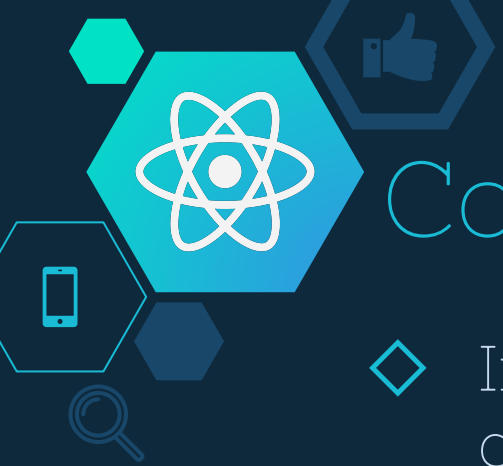
Crear una aplicación React

6

Estado en componente

Determinar los lugares de la aplicación donde el estado vivirá

"The most effective debugging tool is still careful thought, coupled with judiciously placed print statements." – Brian W. Kernighan



Contenido

- Inicializando estado en constructores
- Actualización de **estado** desde propiedades
- Manejo de **errores**
- Renderizado **condicional** de contenido

Photo by Fredy Jacob on Unsplash

Iniciar estado en constructores

Donde se puede iniciar el estado del componente

Formas de crear estado

Se puede utilizar el constructor ya que es el primer método que se llama del componente y donde se construye el componente

Es el único lugar donde es aceptable asignar directamente al estado

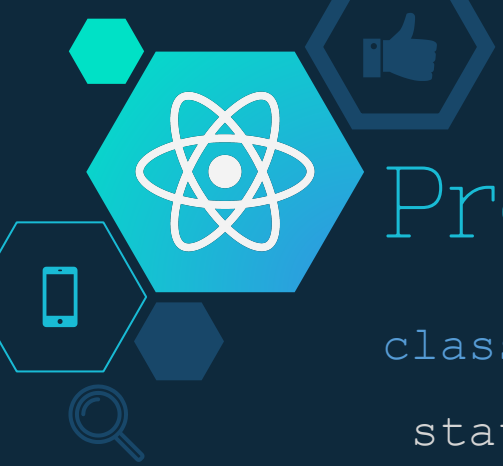
Usando constructor

```
class App extends React.Component {
 constructor(props) {
   super(props);
   this.state = { loggedIn: false }
 }
  render() {
   // whatever you like
 }
}
```

Formas de crear estado

Utilizar una propiedad de clase

La sintaxis aún se encuentra como propuesta



Propiedad de clase

```
class App extends React.Component {
 state = { loggedIn: false }

 render() {
   // whatever you like
 }
}
```

Estado desde propiedades

En muchos casos es un anti patrón

Los componentes se renderizan nuevamente cuando cambian sus propiedades

Estado desde propiedades

```
class BadExample extends Component {
 state = { data: props.data }

 componentDidUpdate(oldProps) {
   if(oldProps.data !== this.props.data) {
     // This triggers an unnecessary re-render
     this.setState({ data: this.props.data });
   }
 }
```

Estado desde propiedades

```
class GoodExample extends Component {
 render() {
   return (
     <div>The data: {this.props.data}</div>
   )
 }
}
```

Estado desde propiedades

Es válido cuando es solo un valor inicial del estado

Photo by Markus Spiske on Unsplash

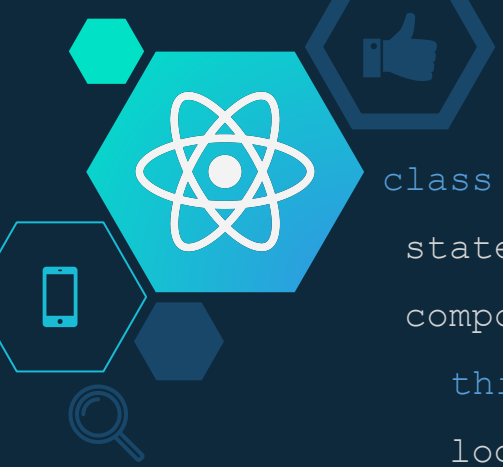
Manejo de errores

Manejar errores en la jerarquía de componentes

Errores de la UI

Usualmente un error en la interfaz de usuario no debe detener la aplicación

En la versión 16 de react se introduce el concepto de **Error Boundaries** definiendo el método de ciclo de vida **componentDidCatch(error, info)**



```
class ErrorBoundary extends React.Component {
 state = { hasError: false }
 componentDidCatch(error, info) {
   this.setState({ hasError: true });
   logErrorToMyService(error, info);
 }
 render() {
   if (this.state.hasError) {
     return <h1>Something went wrong.</h1>;
   }
   return this.props.children;
 }
}
```

Errores de la UI

componentDidCatch funciona como una sentencia catch pero en JSX

Solo class components pueden ser Error Boundaries

```
<ErrorBoundary>
 <MyWidget />
</ErrorBoundary>
```

Renderizado condicional

Renderizar elementos basados en el estado del componente

El renderizado condicional funciona de la misma manera que las condiciones en JavaScript o el operador condicional

Renderizado condicional

Interpolación condicional

```
{ this.state.value > 0 && <div>Hello</div> }
```

```
User is {this.state.value ? 'currently' :
'not'} logged in.
```

Renderizado condicional

En casos que se quiera ocultar un componente se puede retornar null

Si un componente ya fue renderizado y se invoca nuevamente un renderizado pero con un valor null, el componente se ocultara y no afecta los métodos del ciclo de vida

Revisión del módulo

Iniciando estado

Existen dos formas de iniciar estado es cuestión de gusto

Actualizar estado

El estado puede ser actualizado por las propiedades sin embargo hay que pensar cuidadosamente

Manejo de errores

componentDidCatch permite manejo de errores de forma declarativa

Renderizado condicional

Existen formas de renderizar contenido dependiendo del estado del componente

Lab 6

Estado del componente

7

Peticiones HTTP con react

Hacer llamadas a servicios REST para obtener información

"Don’t worry if it doesn’t work right. If everything did, you’d be out of a job." – Mosher’s Law of Software Engineering

Contenido

- API **Flickr** para manejo de imágenes
- Uso de **Axios** para invocar servicios **REST**

Photo by Fredy Jacob on Unsplash

API Flickr para manejo de imágenes

Obtener imágenes haciendo peticiones REST

API Flickr

Para nuestra aplicación vamos a obtener imágenes utilizando flickr

Podemos acceder al sitio de desarrolladores de flickr y seguir los pasos así como acceder a su documentación

https://www.flickr.com/services/developer/api/

flickr.photos.search

Para invocar la búsqueda debemos proporcionar los parámetros requeridos

- `api_key`

Biblioteca axios

Cliente HTTP basado en promesas

Axios es muy sencillo en su uso tanto como en el manejo de respuestas y errores.

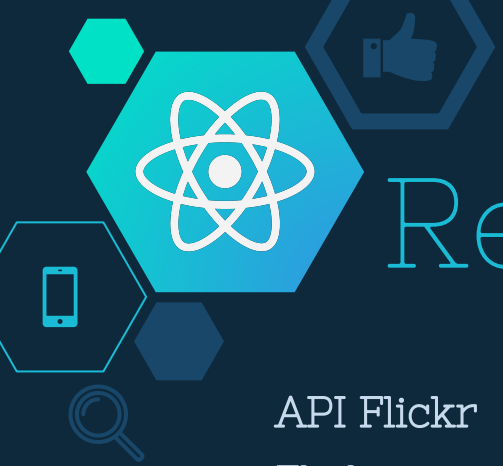
https://github.com/axios/axios

Biblioteca axios

Para instalar axios vamos a ejecutar en la terminal:

npm install --save axios

Axios cuenta con funciones para realizar los verbos más comunes en HTTP



Revisión del módulo

API Flickr

Flickr expone un API sencilla para usar las imágenes de su servicio

Axios

Permite un manejo sencillo de las peticiones HTTP

Lab 7

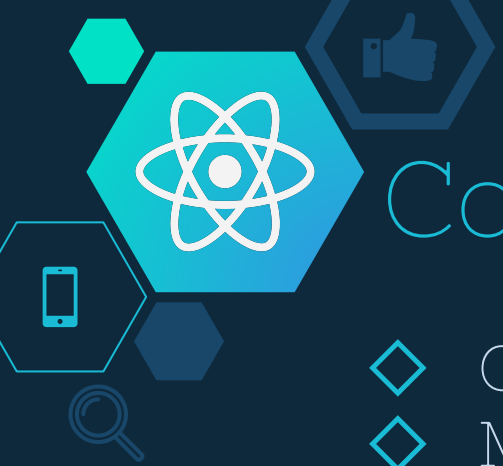
Peticiones HTTP

8

Introducción a Redux

Manejador de estado de componentes

"The sooner you start to code, the longer the program will take." – Roy Carlson



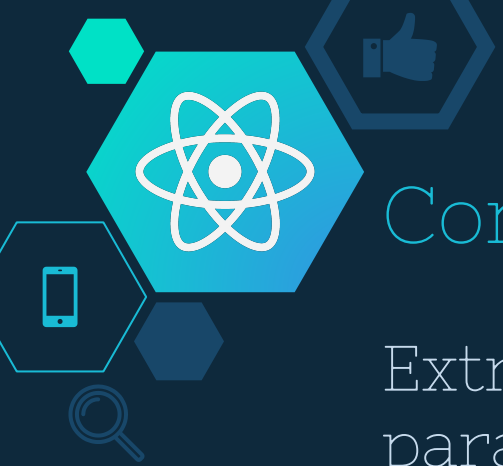
Contenido

- Conceptos básicos de **Redux**
- Mapeo con redux
- Creación de **reducers**
- Integración react con redux
- Funcion **connect**

Photo by Fredy Jacob on Unsplash

Conceptos básicos de Redux

Biblioteca para el manejo del estado



Conceptos Básicos

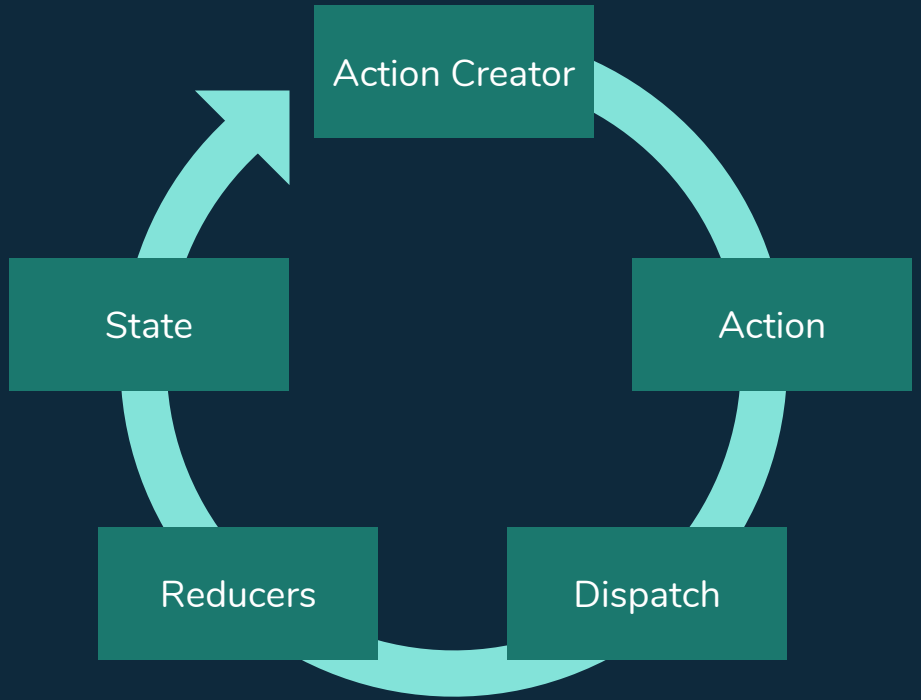
Extraer el **estado** de los componentes para que sean manejado por **redux**

Ayuda a hacer aplicaciones complejas más sencillas

No se requiere para trabajar con react

No fue creada para trabajar con React explícitamente

Ciclo de redux



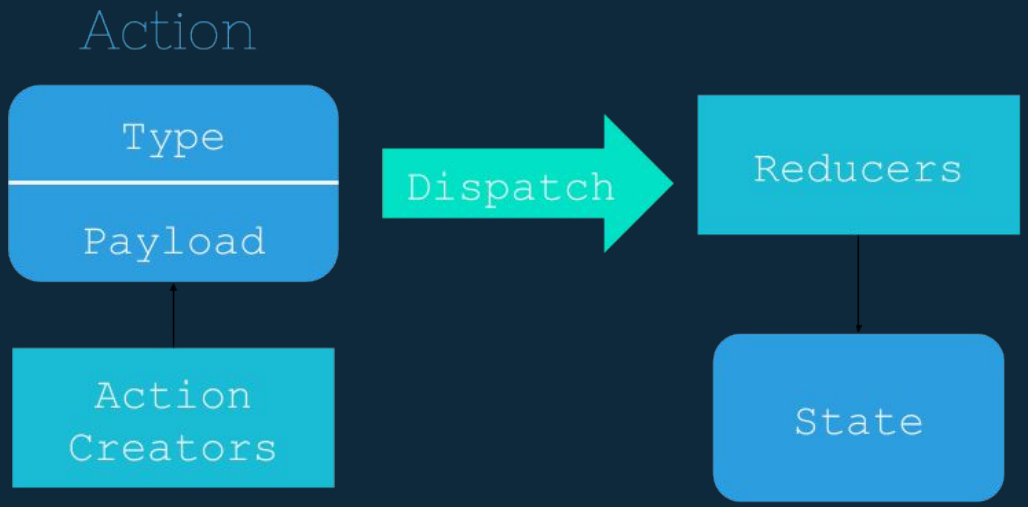
Conceptos Básicos

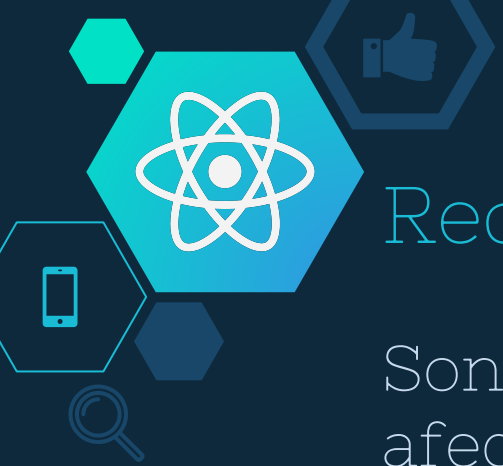
ActionCreators general Actions

Los Actions son enviados mediante el dispatch a cada Reducer

Cada reducer es encargado de mantener una porción de información de la aplicación

Mapeo con redux



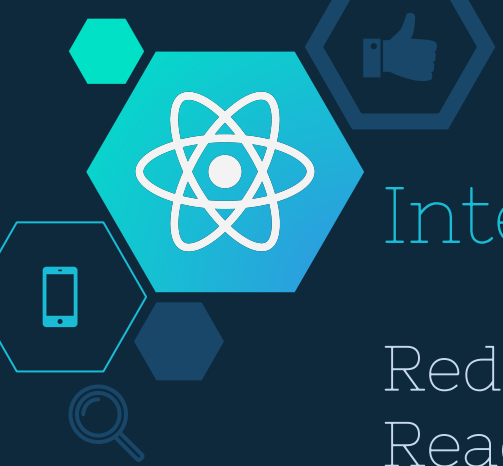


Reducers

Son todos aquellos encargados de afectar el estado global de la aplicación al recibir Actions pertinentes y generar un nuevo **estado**

Reducers

Máquinas de estado que a partir de un Action analizan el payload para generar un estado nuevo de la aplicación.



Integración React-Redux

Redux no está ligado directamente con React es por eso que existe una biblioteca que nos apoya para integrar ambas bibliotecas.

react-redux

Funcción connect

Esta función se usa para crear un envoltorio de nuestro componente el cual se utiliza para conectar nuestro componente con el estado global de **Redux**

Revisión del módulo

Conceptos básicos

ActionCreators Actions
Dispatch Reducer State

Reducers

Las funciones que modifican el estado global

react-redux

Biblioteca que nos ayuda a integrar React y Redux

Funcion connect

Enlaza nuestro componente y sus propiedades con el estado de Redux (Provider)

Lab 8

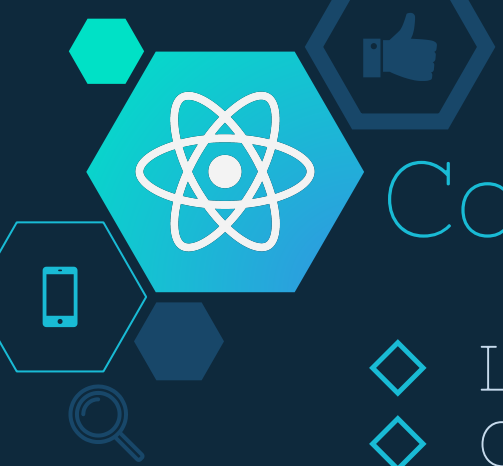
Introducción a Redux

9

Continuando con Redux

Entendiendo más sobre Redux y su funcionamiento

“The most important property of a program is whether it accomplishes the intention of its user.” – C.A.R. Hoare



Contenido

- Llamadas **asíncronas** con Redux
- Creacion de **actions**
- Creando **stores**
- Carga de información

Dependencias de Redux

react-redux

Capa de integración entre react y redux

redux-thunk

Permite realizar solicitudes en un aplicación redux

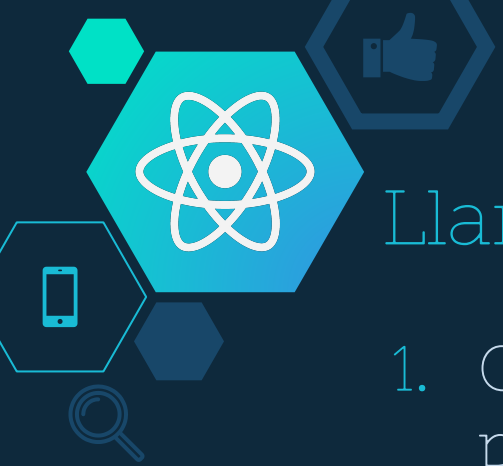
Dependencias de Redux

react-redux

Capa de integración entre react y redux

redux-thunk

Permite realizar solicitudes en un aplicación redux



Llamadas asíncronas con redux

1. Componente se renderiza en pantalla
2. **componentDidMount** y ejecutamos un actionCreator
3. El **actionCreator** ejecuta la llamada al API
4. El API responde y se retorna una acción con los datos como **payload**

Acciones en Redux

1. Actions son mensajes que indican a los stores que partes del estado modificar
2. Usualmente se generan dos propiedades **type y payload**
3. Type indica el tipo de acción
4. Payload los datos de dicha acción

Llamadas asíncronas con redux

redux-thunk es un middleware que se aplica a nuestro store y que maneja todas las acciones que sean funciones para ejecutarlas por nosotros.

¡Gracias!

¿Preguntas?

Me pueden contactar en:

- @RoqueManuel
- skaroque@gmail.com