UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

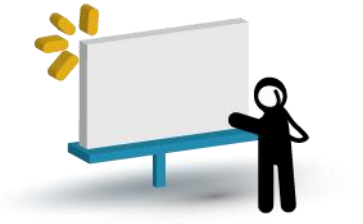
Desarrollo en React JS

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo II

Componentes y Virtual DOM

Unidad 1: Componentes. Parte 1



Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Presentación:

En esta unidad aprenderemos qué son las propiedades, estados, eventos y ciclos de vida de un componente.

A partir de ello podremos comenzar a desarrollar nuestros componentes y darle forma a nuestra aplicación web.

Objetivos:

Que los participantes*:

- Comprendan que son las propiedades, estados y eventos de un componente.
- Entiendan el marco teórico de los ciclos de vida de un componente.
- Aprendan a aplicar propiedades, eventos, estados y ciclos de vida en un componente.

Bloques temáticos*:

- Componentes
- Ciclos de vida
- Descripción de los métodos de los ciclos de vida

Consignas para el aprendizaje colaborativo

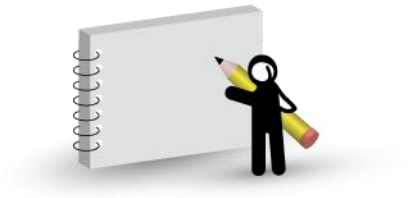
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.



Tomen nota*

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

Componentes

Propiedades

Un componente en React puede recibir propiedades como parámetros desde un componente padre para poder insertar valores y eventos en su HTML.

Imagina que tienes un componente que representa un menú con varias opciones, y éstas opciones las pasamos por parámetros como una propiedad llamada options:

```
import React from 'react'

class App extends React.Component {
  constructor () {
    super()
  }

  render () {
    let menuOptions = ['Opción 1', 'Opción 2', 'Opción 3']
    return <Menu options={menuOptions} />
  }
}
```

¿Cómo accedemos a estas propiedades en el componente hijo a la hora de renderizarlo? Por medio de las props. Veamos como con el código del componente <Menu />:

```
import React from 'react'

class Menu extends React.Component {
  constructor (props) {
    super(props)
  }

  render () {
    let options = this.props.options
    return (
      <ul>
       {options.map(option => <li>{option}</li>)}
      </ul>
    )
  }
}
```

En el método render creamos una variable options con el valor que tenga this.props.options. Éste options dentro de props es el mismo atributo options que tiene el componente <Menu />y es a través de props como le pasamos el valor desde el padre al componente hijo.

```
import React from 'react'
import ReactDOM from 'react-dom'

class App extends React.Component {
  constructor(props) {
    super(props)
  }

  render() {
    let menuOptions = ['Opción 1', 'Opción 2', 'Opción 3']
    return <Menu options={menuOptions} />
  }
}

class Menu extends React.Component {
  constructor(props) {
    super(props)
  }

  render() {
    let options = this.props.options
    return (
      <ul>
        {options.map(option => <li>{option}</li>)}
      </ul>
    )
  }
}

ReactDOM.render(<App />, document.getElementById('app')
```

Estados

Además de las props, los componentes en React pueden tener estado. Lo característico del estado es que si éste cambia, el componente se renderiza automáticamente. Veamos un ejemplo de esto.

Si tomamos el código anterior y en el componente <App />guardamos en su estado las opciones de menú, el código de App sería así:

```
class App extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      menuOptions: ['Opción 1', 'Opción 2', 'Opción 3']
    }
  }

  render() {
    return <Menu options={this.state.menuOptions} />
  }
}
```

De ésta manera, las "opciones" pertenecen al estado de la aplicación (App) y se pueden pasar a componentes hijos (Menú) a través de las props.

Ahora, si queremos cambiar el estado, añadiendo una nueva opción al menú tenemos a nuestra disposición la función setState, que nos permite modificarlo.

Aprovechando esto, vamos a modificar el estado a través de un evento disparado desde el componente hijo hacia el padre

Eventos

Si las propiedades pasan de padres a hijos, es decir hacia abajo, los eventos se disparan hacia arriba, es decir de hijos a padres. Un evento que dispare un componente, puede ser recogido por el padre.

Veámoslo con un ejemplo. El componente <Menu /> va a tener una nueva propiedad llamada onAddOption:

```
render() {
  return (
    <Menu
      options={this.state.menuOptions}
      onAddOption={this.handleAddOption.bind(this)}
    />
)
}
```

Esta propiedad va a llamar a la función handleAddOption en <App /> para poder modificar el estado:

```
handleAddOption () {
  this.setState({
    menuOptions: this.state.menuOptions.concat(['Nueva Opción'])
  })
}
```

ada vez que se llame a la función, añadirá al estado el ítem Nueva Opción. Cómo dijimos al inicio, cada vez que se modifique el estado, se "re-renderizará" el componente y veremos en el navegador la nueva opción añadida.

Para poder llamar a esa función, necesitamos disparar un evento desde el hijo. Voy a añadir un elemento <button> y utilizar el evento onClick de JSX, que simula al listener de click del ratón y ahí llamaré a la función "propiedad" onAddOption del padre.

El código de completo es:

```
class App extends React.Component {
  constructor(props) {
    super(props)
    this.state = {
      menuOptions: ['Opción 1', 'Opción 2', 'Opción 3']
    }
  }

  render() {
    return (
      <Menu
        options={this.state.menuOptions}
        onAddOption={this.handleAddOption.bind(this)}
      />
    )
  }

  handleAddOption () {
    this.setState({
      menuOptions: this.state.menuOptions.concat(['Nueva Opción'])
    })
  }
}
```

```
class Menu extends React.Component {
  constructor(props) {
    super(props)
  }

  render() {
    let options = this.props.options
    return (
      <div>
        <ul>
          {options.map(option => <li>{option}</li>)}
        </ul>
        <button onClick={this.props.onAddOption}>Nueva Opción</button>
      </div>
    )
  }
}

ReactDOM.render(<App />, document.getElementById('app'))
```

Cada vez que hagamos click en el botón, llamará a la función que le llega por props. Esta función, llama en el componente padre (App) a la función handleAddOption y la bindeamos con this, para que pueda llamar a this.setState dentro de la función. Éste setState modifica el estado y llama internamente a la función render lo que provoca que se vuelva a "pintar" todo de nuevo.

Ciclos de vida

El ciclo de vida no es más que una serie de estados por los cuales pasa todo componente a lo largo de su existencia. Esos estados tienen correspondencia en diversos métodos, que nosotros podemos implementar para realizar acciones cuando se van produciendo.

En React es fundamental el ciclo de vida, porque hay determinadas acciones que debemos necesariamente realizar en el momento correcto de ese ciclo. Ese es el motivo por el que hay que aprenderse muy bien cuáles son las distintas etapas por las que pasa la ejecución de un componente React.

Estas son las tres clasificaciones de los estados dentro de un ciclo de vida del componente.

- El montaje se produce la primera vez que un componente va a generarse, incluyéndose en el DOM.
- La actualización se produce cuando el componente ya generado se está actualizando.
- El desmontaje se produce cuando el componente se elimina del DOM.

Además, dependiendo del estado actual de un componente y lo que está ocurriendo con él, se producirán grupos diferentes de etapas del ciclo de vida. En la siguiente imagen puedes ver un resumen de esta diferenciación.

Primer renderizado	Cambios en las propiedades	Cambio en el estado	Componente se desmonta
getDefaultProps *	componentWillReceiveProps	shouldComponentUpdate	componentWillUnmount
getInitialState *	shouldComponentUpdate	componentWillUpdate	
componentWillMount	componentWillUpdate	render *	
render *	render *	componentDidUpdate	
componentDidMount	componentDidUpdate		

Descripción de los Métodos de ciclos de vida

componentWillMount()

Este método del ciclo de vida es de tipo montaje. Se ejecuta justo antes del primer renderizado del componente. Si dentro de este método seteas el estado del componente con setState(), el primer renderizado mostrará ya el dato actualizado y sólo se renderizará una vez el componente.

componentDidMount()

Método de montaje, que solo se ejecuta en el lado del cliente. Se produce inmediatamente después del primer renderizado. Una vez se invoca este método ya están disponibles los elementos asociados al componente en el DOM. Si se tiene que realizar llamadas Ajax, setIntervals, y cosas similares, éste es el sitio adecuado.

componentWillReceiveProps(nextProps)

Método de actualización que se invoca cuando las propiedades se van a actualizar, aunque no en el primer renderizado del componente, así que no se invocará antes de inicializar las propiedades por primera vez. Tiene como particularidad que recibe el valor futuro del objeto de propiedades que se va a tener. El valor anterior es el que está todavía en el componente, pues este método se invocará antes de que esos cambios se hayan producido.

shouldComponentUpdate(nextProps, nextState)

Es un método de actualización y tiene una particularidad especial con respecto a otros métodos del ciclo de vida, que consiste en que debe devolver un valor boleano. Si devuelve verdadero quiere decir que el componente se debe renderizar de nuevo y si devuelve falso quiere decir que el componente no se debe de renderizar de nuevo. Se invocará tanto cuando se producen cambios de propiedades o cambios de estado y es una oportunidad de comprobar si esos cambios deberían producir una actualización en la vista del componente. Por defecto (si no se definen) devuelve siempre true, para que los componentes se actualicen ante cualquier cambio de propiedades o estado. El motivo de su existencia es la optimización, puesto que el proceso de renderizado tiene un coste y podemos evitarlo si realmente no es necesario de realizar.

componentWillUpdate(nextProps, nextState)

Este método de actualización se invocará justo antes de que el componente vaya a actualizar su vista. Es indicado para tareas de preparación de esa inminente renderización causada por una actualización de propiedades o estado.

componentDidUpdate(prevProps, prevState)

Método de actualización que se ejecuta justamente después de haberse producido la actualización del componente. En este paso los cambios ya están trasladados al DOM del navegador, así que podríamos operar con el DOM para hacer nuevos cambios. Como parámetros en este caso recibes el valor anterior de las propiedades y el estado.

componentWillUnmount()

Este es el único método de desmontado y se ejecuta en el momento que el componente se va a retirar del DOM. Este método es muy importante, porque es el momento en el que se debe realizar una limpieza de cualquier cosa que tuviese el componente y que no deba seguir existiendo cuando se retire de la página. Por ejemplo, temporizadores o manejadores de eventos que se hayan generado sobre partes del navegador que no dependen de este componente.

Ver aplicación de ejemplo con la utilización de los ciclos de vida.

Bibliografía utilizada y sugerida

Fedosejev, A. (2015). React.js Essentials (1 ed.). EEUU, Packt.

Amler, . (2016). ReactJS by Example (1 ed.). EEUU, Packt.

Stein, J. (2016). ReactJS Cookbook (1 ed.). EEUU, Packt.

http://www.enrique7mc.com/2016/07/tipos-de-componentes-en-react-js/

https://desarrolloweb.com/articulos/ciclo-vida-componentes-react.html

https://carlosazaustre.es/estructura-de-un-componente-en-react/

Lo que vimos:

En esta unidad aprendimos que son las propiedades, estados, eventos y ciclos de vida de un componente.

Lo que viene:

En la próxima unidad aprenderemos acerca del virtual DOM, ajax y como darle estilos a nuestros componentes.