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 I

Nivelación, Javascript y configuración del entorno

Unidad 4: Componentes. Parte 1

Presentación:

En esta unidad aprenderemos un concepto fundamental en el desarrollo con react js, los componentes.

Comprenderemos que son, qué ventajas tiene este enfoque, como crearlos y cómo utilizarlos.

Objetivos:

Que los participantes*:

- Comprendan las ventajas del enfoque
- Entiendan que es un componente
- Aprendan cómo crear y utilizar los mismos

Bloques temáticos*:

- Componentes
- Tipos de componentes
- Más sobre componentes
- Desarrollando sobre nuestra aplicación

Consignas para el aprendizaje colaborativo

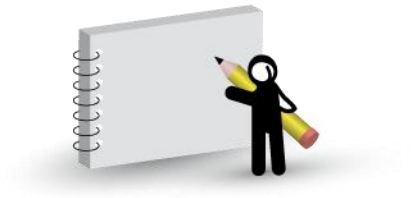
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- 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

En pocas palabras

Comenzamos un tema importante en React, ya que en esta librería realizamos un desarrollo basado en componentes. Básicamente quiere decir que,mediante anidación de componentes podremos crear aplicaciones completas de una manera modular, de fácil mantenimiento a pesar de pòder ser complejas.

En React JS existen dos categorías recomendadas para los componentes: los componentes de presentación y los componentes contenedores.

Los **componentes de presentación** son aquellos que simplemente se limitan a mostrar datos y tienen poca o nula lógica asociada a manipulación del estado, es preferible que la mayoría de los componentes de una aplicación sean de este tipo porque son más fáciles de entender y analizar.

Los **componentes contenedores** tienen como propósito encapsular a otros componentes y proporcionarles las propiedades que necesitan, además se encargan de modificar el estado de la aplicación por ejemplo usando Flux o Redux para despachar alguna acción y que el usuario vea el cambio en los datos.

Ventajas del enfoque

- Favorece la separación de responsabilidades, cada componente debe tener una única tarea.
- Al tener la lógica de estado y los elementos visuales por separado es más fácil reutilizar los componentes.
- Se simplifica la tarea de hacer pruebas unitarias.
- Puede mejorar el rendimiento de la aplicación.
- La aplicación es más fácil de entender.

Composición de componentes

Así como en programación funcional se pasan funciones como parámetros para resolver problemas más complejos, creando lo que se conoce como composición funcional, en ReactJS podemos aplicar este mismo patrón mediante la composición de componentes

Las aplicaciones se realizan con la composición de varios componentes. Estos componentes encapsulan un comportamiento, una vista y un estado. Pueden ser muy complejos, pero es algo de lo que no necesitamos preocuparnos cuando estamos desarrollando la aplicación, porque el comportamiento queda dentro del componente y no necesitamos complicarnos por él una vez se ha realizado.

En resumen, al desarrollar crearemos componentes para resolver pequeños problemas, que por ser pequeños son más fáciles de resolver y en adelante son más fáciles de visualizar y comprender. Luego, unos componentes se apoyarán en otros para resolver problemas mayores y al final la aplicación será un conjunto de componentes que trabajan entre sí. Este modelo de trabajo tiene varias ventajas, como la facilidad de mantenimiento, depuración, escalabilidad, etc.

Tipos de componentes

Características de los componentes de presentación

- Orientados al aspecto visual
- No tienen dependencia con fuentes de datos (e.g. Flux)
- Reciben callbacks por medio de props
- Pueden ser descritos como componentes funcionales.
- Normalmente no tienen estado

Ejemplo de componente de presentación

```
// Componentes de presentación
class Item extends React.Component {
  render () {
      return (
          <li><a href='#'>{ this.props.valor }</a></li>
      );
  }
}

class Input extends React.Component {
  render () {
      return (
          <input type='text' placeholder={ this.props.placeholder } />
      );
  }
}

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

En este fragmento de código definimos algunos componentes de presentación (Header, Item e Input) que son mostrados en la página dentro de un componente contenedor.

Los componentes de presentación no tienen estado y sólo contienen código para mostrar información, y para hacer el código más simple dichos componentes se crean con una función en vez de una clase. Además reciben de su componente padre las propiedades necesarias.

También es posible escribir estos mismos componentes de forma funcional, es decir, en vez de que sean definidos con la palabra class, serán creados como simples funciones que regresan el contenido que se mostrará:

```
// Componentes de presentación de forma funcional
const Titulo = ({nombre} = props) => (
  <h1>{ nombre }</h1>
);

const Item = (props) => (
  <li><a href='#'>{ props.valor }</a></li>
);

const Input = (props) => (
  <input type='text' placeholder={ props.placeholder } />
);
```

Usando esta sintaxis las propiedades se reciben como parámetros de la función e incluso es posible aplicar destructuring para obtener las variables que nos interesan por separado.

Características de los componentes contenedores

- Orientados al funcionamiento de la aplicación
- Contienen componentes de presentación y/o otros contenedores
- Se comunican con las fuentes de datos
- Usualmente tienen estado para representar el cambio en los datos

```
// Contenedor
class AppContainer extends React.Component {
  constructor (props) {
    super(props);
    this.state = {
      temas: ['JavaScript', 'React JS', 'Componentes']
    };
  }

  render () {
    const items = this.state.temas.map(t => (
      <Item valor={ t } />
    ));
    return (
      <div>
       <Titulo nombre='List Items'/>
       <ul>{ items }</ul>
       <Titulo nombre='Inputs'/>
       <div>
        <Input placeholder='Nombre' /><br/>
        <Input placeholder='Apellido' />
       </div>
      </div>
    );
  }
}
```

Por otra parte el componente contenedor define los datos contenidos en la aplicación y también los manipula, después crea los componentes hijos y los muestra en el método render.

Más sobre componentes

Crear un componente

Tenemos 2 formas de crear un componente, utilizando el método **createClass:**

```
const React = require('react')

const MyComponent = React.createClass({
...
})
```

O bien utilizando la notación de clases de ES 6

```
import React form 'react'

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

Render

Todo componente de React, tiene un método Render que es el que se encarga de renderizar en el navegador el HTML correspondiente al componente.

Este método se llama automáticamente cuando se crea un componente y cuando el estado del componente se actualiza.

En este método es donde usamos JSX para facilitar el desarrollo y creación de elementos HTML. Veamos un ejemplo:

```
import React from 'react'

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

  render () {
    return (
      <div>
        <span>Hola!, soy un componente</span>
      </div>
    )
  }
}
```

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>
    )
  }
}
```

El código final de este ejemplo sería:

```
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')
```

Y esto muestra en el navegador algo como lo siguiente:

- Opción 1
- Opción 2
- Opción 3

Desarrollando sobre nuestra aplicación

En unidades anteriores hemos creado nuestra aplicación con el CLI de react.

Ahora desarrollaremos la home de nuestra aplicación utilizando componentes

Entendiendo mas el codigo

Si observamos el código de la aplicación creada veremos lo siguiente en el archivo **index.js:**

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

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

Aca podemos observar que estamos renderizando el componente **<App />** sobre el elemento con id **root** del DOM. ¿Que tiene el componente **<App />**?

Para responder a esta pregunta debemos ir al archivo **App.js**

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    var app = <div className="App">
    <header className="App-header">
      <img src={logo} className="App-logo" alt="logo" />
      <h1 className="App-title">Welcome to React</h1>
    </header>
    <p className="App-intro">
      To get started, edit <code>src/App.js</code> and save to reload.
    </p>
  </div>;
    return (
      app
    );
  }
}

export default App;
```

Aca podemos observar como el componente **App** renderiza una imagen un párrafo.

También podemos ver como el src de la imagen lo obtiene utilizando el **import** de javascipr y no definiendo su ruta absoluta y/o relativa en el código.

Volviendo al archivo **index.js**, ¿en donde está el elemento del DOM con id **root**?

Veamos el archivo **index.html**

```
<body>
  <noscript>
    You need to enable JavaScript to run this app.
  </noscript>
  <div id="root"></div>
  <!--
    This HTML file is a template.
    If you open it directly in the browser, you will see an empty page.

    You can add webfonts, meta tags, or analytics to this file.
    The build step will place the bundled scripts into the <body> tag.

    To begin the development, run `npm start` or `yarn start`.
    To create a production bundle, use `npm run build` or `yarn build`.
  -->
</body>
</html>
```

Este será el html que se renderiza en nuestro navegador. Luego utilizando react se renderizada el contenido del componente **App** sobre el elemento con id **root**

En caso de querer modificar los metas de nuestra página debemos hacerlo sobre el archivo **index.html**

Modificando nuestra aplicación

Index.html:

Modificamos el title de nuestra página:

```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <!--
      manifest.json provides metadata used when your web app is added to the
      homescreen on Android. See https://developers.google.com/web/fundamentals/engage-and-r
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>Red Social UTN FRBA</title>
  </head>
```

App.js

Modificamos el contenido de nuestro componente principal:

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
import Login from './Login';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {

    return (
      <div className="App">
        <h1>Bienvenido a Red Social UTN FRBA</h1>
        <Login />
      </div>
    );

  }
}

export default App;
```

Importamos el componente **Login** y luego lo instanciamos con **<Login />**

Para que esto funcione debemos crear nuestro componente **Login:**

Cuyo contenido será el siguiente:

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';


class Login extends Component {
  render() {

    return (
      <div className="Login">
        <div>
            <label>Email</label>
            <input type="text" />
        </div>
        <div>
            <label>Contraseña</label>
            <input type="password" />
        </div>
      </div>
    );

  }
}
export default Login;
```

De esta forma podremos desarrollar toda nuestra aplicación utilizando componentes

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/caracteristicas-react.html

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

Lo que vimos:

En esta unidad aprendimos que es un componente, que ventajas tiene su utilización, como crearlos y cómo se instancian entre sí.

Lo que viene:

En la próxima unidad aprenderemos más acerca de componentes, sus propiedades, estados y ciclos de vida.