# APLICACIONES MÓVILES MULTIPLATAFORMA

## LABORATORIO N° 02

# Componentes en React JS

| | | | |
|---|---|---|---|
| ***Alumno(s):*** | | ***Nota*** | |
| ***Grupo:*** | | ***Ciclo:V*** | |

| *Criterio de Evaluación* | *Excelente (4pts)* | *Bueno (3pts)* | *Requiere mejora (2pts)* | *No acept. (0pts)* | *Puntaje Logrado* |
|---|---|---|---|---|---|
| Entiende cómo funciona los componentes | | | | | |
| Utiliza Next-Gen Javascript | | | | | |
| Desarrolla aplicaciones web con ReactJS | | | | | |
| Realiza con éxito lo propuesto en la tarea. | | | | | |
| Es puntual y redacta el informe adecuadamente | | | | | |

# Laboratorio 02:
# Componentes en React JS

## Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:

- Entender el funcionamiento de React JS
- Desarrollar aplicaciones web enfocadas a componentes
- Implementación correcta de Next-Gen Javascript

## Seguridad:

- Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
- No ingresar con líquidos, ni comida al aula de Laboratorio.
- Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

## Equipos y Materiales:

- Una computadora con:
  - Windows 7 o superior
  - VMware Workstation 10+ o VMware Player 7+
  - Conexión a la red del laboratorio
- Máquinas virtuales:
  - Windows 7 Pro 64bits Español - Plantilla
- Instalador de node.js

## Procedimiento:

**Lab Setup**

1. **Instalación y configuración de ReactJS**
   1.1. Ejecuta el siguiente comando para preparar la instancia de ReactJS del presente laboratorio

```
$ create-react-app lab02
```

   1.2. Esto dejará listo nuestro entorno de trabajo, por lo que nos situaremos en la carpeta generada y posteriormente iniciaremos el proyecto con **npm start**

```
$ cd lab02
$ npm start
```

2. Componentes en ReactJS

   2.1. Cree la carpeta **components** dentro de la carpeta src. Luego cree la carpeta **Contador** dentro de componentes y finalmente el archivo **Contador.js** dentro de la misma con el siguiente contenido.

```
import React, {Component} from 'react';

export default class Contador extends Component{
    render(){
        return (<div>
            <h1>Este es mi componente de contador</h1>
        </div>)
    }
}
```

2.2. Modifique el contenido del archivo App.js dentro de la carpeta src para que tenga esta apariencia.

```
import React, { Component } from 'react';

import Contador from './components/Contador/Contador';

class App extends Component {
  render() {
    return (
      <Contador />
    );
  }
}

export default App;
```

2.3. Tome una captura del componente obtenido

3. Uso de props en componentes

3.1. Modifique la función render del componente contador para que tenga el siguiente contenido.

```
render(){
    return (<div>
        <h1>Este es mi componente de contador</h1>
        <p>Este contador iniciará en: {this.props.valor}</p>
    </div>)
}
```

3.2. Adjunte una captura del resultado obtenido

3.3. Modifique la función render del archivo App.js y vea cómo afecta al componente antes modificado.

```
render() {
  return (
    <Contador valor={5} />
  );
}
```

3.4. La propiedad **props** recibe todo argumento pasado al componente. Cada vez que este varía se genera una actualización del mismo. Podemos utilizar los props (y debemos, de hecho) para mostrar el mismo componente con uno u otro comportamiento que lo distinga.

3.5. Modifique la función render de contador.

```
render(){
    let advertencia = (<p>Iniciamos con un número mayor a 5!</p>);
    if(this.props.valor<=5) advertencia = null;
    return (<div>
        <h1>Este es mi componente de contador</h1>
        <p>Este contador iniciará en: {this.props.valor}</p>
        {advertencia}
    </div>)
}
```

3.6. Después de haber adjuntado una captura del resultado obtenido, modifiquemos el componente principal de App.js para demostrar el comportamiento del mismo componente siendo reutilizado con distintos valores.

```
class App extends Component {
  render() {
    return (<div>
      <Contador valor={6} />
      <hr />
      <Contador valor={0} />
    </div>);
  }
}
```

4. Componentes CSS

4.1. Vamos a configurar nuestro proyecto para que genere componentes CSS.

4.2. Ejecutamos el siguiente comando en la consola.

```
$ npm run eject
```

4.3. Este comando procede a expulsar toda la configuración implícita de React y ahora podemos modificarla. Confirmamos que estamos seguros.

```
? Are you sure you want to eject? This action is permanent. Yes
Ejecting...
```

4.4. Antes de volver a iniciar nuestro proyecto, buscaremos el siguiente texto en el archivo webpack.config.dev.js ubicado en la carpeta config.

```
test: /\.css$/,
use: [
  require.resolve('style-loader'),
  {
    loader: require.resolve('css-loader'),
    options: {
      importLoaders: 1,
    },
  },
```

4.5. Lo modificaremos para que luzca así:

```
test: /\.css$/,
use: [
  require.resolve('style-loader'),
  {
    loader: require.resolve('css-loader'),
    options: {
      importLoaders: 1,
      // agregar estas lineas para configurar modulos css
      modules: true,
      localIdentName: '[name]__[local]__[hash:base64:5]'
    },
  },
```

4.6. Ahora buscaremos el siguiente texto en el archivo webpack.config.prod.js ubicado en la carpeta config.

```
use: [
  {
    loader: require.resolve('css-loader'),
    options: {
      importLoaders: 1,
      minimize: true,
      sourceMap: shouldUseSourceMap,
    },
  },
```

4.7. Lo modificaremos para que luzca así:

```
use: [
  {
    loader: require.resolve('css-loader'),
    options: {
      importLoaders: 1,
      minimize: true,
      sourceMap: shouldUseSourceMap,
      //usar esta configuración para usar los modulos css
      modules: true,
      localIdentName: '[name]__[local]__[hash:base64:5]'
    },
  },
```

4.8. Iniciamos nuestro proyecto

```
npm start
```

4.9. Crearemos ahora el archivo Contador.css dentro de la carpeta Contador, con el siguiente contenido

```
.Contador {
    font-size: 14px;
    padding: 5px;
    color: green;
}
```

4.10. Procedemos a importar las clases creadas dentro de dicho componente y la agregamos a nuestro componente contador.

```
import React, {Component} from 'react';

import classes from './Contador.css';

export default class Contador extends Component{
    render(){
        let advertencia = (<p>Iniciamos con un número mayor a 5!</p>);
        if(this.props.valor<=5) advertencia = null;
        return (<div className={classes.Contador}>
            <h1>Este es mi componente de contador</h1>
            <p>Este contador iniciará en: {this.props.valor}</p>
            {advertencia}
        </div>)
    }
}
```

5. States del componente

5.1. Otra forma común de manipular un componente es a través de una variable interna llamada state (estado). Gracias a esta, el componente puede tener todo un ciclo de vida.

5.2. Modificaremos al componente contador para que cuente con las siguientes líneas adicionales.

```
export default class Contador extends Component{
    state = {
        cont: this.props.valor
    }
    render(){
        let advertencia = (<p>Iniciamos con un número mayor a 5!</p>);
        if(this.props.valor<=5) advertencia = null;
        return (<div className={classes.Contador}>
            <h1>Este es mi componente de contador</h1>
            <p>Este contador iniciará en: {this.props.valor}</p>
            {advertencia}
            <p>Número actual: {this.state.cont}</p>
        </div>)
    }
}
```

5.3. Esta es la forma de declarar un state cuando el componente se carga. Podemos ponerle un valor estático o alimentar nuestra data de la misma propiedad props.

5.4. Modifiquemos parte del render para tener botones que alteren el estado de nuestro componente.

```
<p>Número actual: {this.state.cont}</p>
<button onClick={this.disminuirHandler}>Disminuir</button>
<button onClick={this.resetHandler}>Resetear</button>
<button onClick={this.aumentarHandler}>Aumentar</button>
```

5.5. Procedemos a crear las funciones señaladas en estos componentes. Las declararemos después de la declaración de nuestro state y antes del render solamente por un tema de orden, no es un requisito de React.

```
state = {
    cont: this.props.valor
}
disminuirHandler = () => {
    this.setState({
        cont: this.state.cont - 1
    });
}
aumentarHandler = () => {
    this.setState({
        cont: this.state.cont + 1
    });
}
resetHandler = () => {
    this.setState({
        cont: this.props.valor
    });
}
```

5.6. Adjunte captura del comportamiento del componente

6. **Finalizar la sesión**
   6.1. Apagar el equipo virtual
   6.2. Apagar el equipo

## Tarea:

Realizar la creación de un componente calculadora que pueda hacer las funciones básicas:

- Suma
- Resta
- Multiplicación
- División
- Porcentaje

## Conclusiones:

Indicar las conclusiones que llegó después de los temas tratados de manera práctica en este laboratorio.