UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

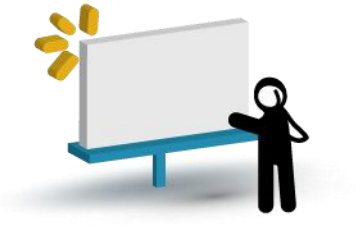
Desarrollo en React JS

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 2: 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

Presentación:

En esta unidad aprenderemos qué es una api rest y un promise y cómo combinar ambas para consultar datos a un sistema externo.

Luego haremos un ejemplo completo de desarrollo con los conceptos vistos hasta el momento.

Objetivos:

Que los participantes*:

- Comprendan que es una api rest
- Conozcan la técnica de promise en ES 6
- Implementen los conocimientos vistos hasta el momento

Bloques temáticos*:

- Map
- Trabajando con nuestra aplicación
- API REST
- Promise
- Aplicando una consulta API a nuestro código

Consignas para el aprendizaje colaborativo

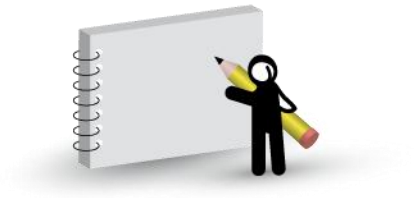
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- 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.

Map

Con el método map, podremos recorrer una colección de objetos.

Por ejemplo:

```
{this.props.cervezas.map(cerveza=><li>{cerveza}</li>)}
```

Cervezas es un array enviado como propiedad desde el llamado al componente:

```
class App extends Component {
  render() {

    return (
      <div className="App">
        <h1>Bienvenido a Red Social UTN FRBA</h1>
        <Productos cervezas={['Quilmes','Brahma']} />
      </div>
    );

  }
}
```

Podemos ver que el primer parámetro recibido por la función dentro de map hace referencia a cada objeto devuelto por la iteración del array.

Trabajando con nuestra aplicación

Vamos a comenzar a desarrollar con nuestra aplicación, utilizando los conceptos vistos hasta el momento.

Comencemos creando nuestro componente Home.

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


class Home extends Component {
  constructor(props) {
    super(props)

  }

  render() {
    return (
      <div>

      </div>
    )
  }
}
export default Home;
```

Luego realizaremos el llamado a este componente desde app.js

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';
import Login from './Login';
import Productos from './Productos';
import Home from './Home';
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>
        <Home />
      </div>
    );

  }
}

export default App;
```

Como vemos debemos incluir el componente **Home** con el import y luego realizar la instancia del mismo con la etiqueta html **<Home />**

Nuestra Home deberá mostrar distintos perfiles de nuestra red social, por lo cual crearemos un nuevo componente llamado perfil:

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


class Perfil extends Component {
  constructor(props) {
    super(props)

  }

  render() {
    return (
      <div>
        Perfil
      </div>
    )
  }
}
export default Perfil;
```

Ahora haremos la instancia de este componente desde **Home**:

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

class Home extends Component {
  constructor(props) {
    super(props)

  }

  render() {
    return (
      <div>
        Home
        <Perfil />
      </div>

    )
  }
}
export default Home;
```

Hasta ahora hemos creado 2 componentes y desde home podemos llamar a cada perfil del usuario.

Ahora lo que haremos es definir un array que contendrá los datos de los perfiles de nuestra red social. Este array lo inicializamos antes de renderizar el componente home:

```
class Home extends Component {
  perfiles;
  constructor(props) {
    super(props)

  }
  componentWillMount(){
    this.perfiles = [
        {
            id:1,
            nombre: "Leandro",
            apellido: "Gil Carrano",
            foto: "../img/f1.png"

        },
        {
            id:2,
            nombre: "Matias",
            apellido: "Perez",
            foto: "../img/f1.png"

        },
        {
            id:3,
            nombre: "Cristian",
            apellido: "Sanchez",
            foto: "../img/f1.png"

        }
    ];
}
```

Como vemos declaramos la variable de clase **perfiles** y luego inicializamos la misma en el método **componentWillMount**

Ahora lo que haremos es pasar este array de datos al componente perfil para que él mismo lo renderize (nuevamente esto en el componente home)

```
render() {
  return (
    <div>
      Home
      {this.perfiles.map(
          perfil=><Perfil datos={perfil}  />)
      }

    </div>

  )
}
```

Como vemos en este caso utilizamos nuevamente la función **map**

Estos datos los recibimos como propiedades desde el componente **Perfil**

```
class Perfil extends Component {
  constructor(props) {
    super(props)
    console.log("props",this.props.datos)
  }

  render() {
    return (
      <div>
        {this.props.datos.nombre}
        {this.props.datos.apellido}
        <img src={this.props.datos.foto} />
      </div>
    )
  }
}
export default Perfil;
```

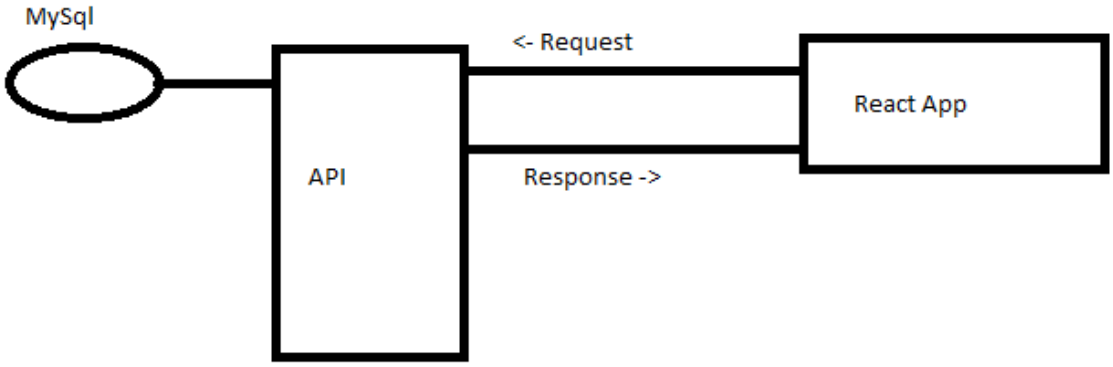
Podemos observar que debemos utilizar la función **super(props)** y luego mostrar las propiedades recibidas en el componente desde el componente **Home**

Por último debemos agregar una carpeta en **public** en la cual incluimos la imagen a renderizar:

El resultado obtenido será el siguiente:

API REST

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST.



Ventajas

- Separación entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
- Visibilidad, fiabilidad y escalabilidad. La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.
- La API REST siempre es independiente del tipo de plataformas o lenguajes: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

Promise

Una Promise (promesa en castellano) es un objeto que representa la terminación o el fracaso eventual de una operación asíncrona. Una promesa puede ser creada usando su constructor. Sin embargo, la mayoría de la gente son consumidores de promesas ya creadas devueltas desde funciones.

Esencialmente, una promesa es un objeto devuelto al cual enganchas las funciones callback, en vez de pasar funciones callback a una función.

Por ejemplo, en vez de una función del viejo estilo que espera dos funciones callback, y llama a una de ellas en caso de terminación o fallo:

```
function exitoCallback(resultado) {
  console.log("Tuvo éxito con " + resultado);
}

function falloCallback(error) {
  console.log("Falló con " + error);
}

hazAlgo(exitoCallback, falloCallback);
```

las funciones modernas devuelven una promesa a la que puedes enganchar tus funciones de retorno

```
hazAlgo().then(exitoCallback, falloCallback);
```

A diferencia de las funciones callback pasadas al viejo estilo, una promesa viene con algunas garantías:

- Las funciones callback nunca serán llamadas antes de la terminación de la ejecución actual del bucle de eventos de JavaScript.
- Las funciones callback añadidas con .then serán llamadas después del éxito o fracaso de la operación asíncrona, como arriba.
- Pueden ser añadidas múltiples funciones callback llamando a .then varias veces, para ser ejecutadas independientemente en el orden de inserción.
- Pero el beneficio más inmediato de las promesas es el encadenamiento.

```
hazAlgo().then(function(resultado) {
  return hazAlgoMas(resultado);
})
.then(function(nuevoResultado) {
  return hazLaTerceraCosa(nuevoResultado);
})
.then(function(resultadoFinal) {
  console.log('Obtenido el resultado final: ' + resultadoFinal);
})
.catch(falloCallback);
```

Aplicando una consulta API a nuestro código

Modificaremos nuestro código anterior para consultar los datos de nuestros usuarios a una API REST

En el componente **Home** realizaremos lo siguiente:

```
constructor(props) {
  super(props)
  this.state = {
      error: null,
      isLoaded: false,
      perfiles: []
    };
}
```

Definimos en el estado del componente las variables error, isLoaded, perfiles

Luego modificamos el ciclo de vida utilizado anteriormente por **componentDidMount**

```
componentDidMount(){

    fetch("https://jsonplaceholder.typicode.com/users")
    .then(res => res.json())
    .then(
      (result) => {
          console.log(result)
        this.setState({
          isLoaded: true,
          perfiles: result
        });
      },
      // Note: it's important to handle errors here
      // instead of a catch() block so that we don't swallow
      // exceptions from actual bugs in components.
      (error) => {
          console.log(error)
        this.setState({
          isLoaded: true,
          error
        });
      }
    )
}
```

Dentro de este método incluimos el llamado a la API REST, como vemos debemos trabajar la respuesta con el promise (then)

```
render() {
  const { error, isLoaded, perfiles } = this.state;
  if (error) {
    return <div>Error: {error.message}</div>;
  } else if (!isLoaded) {
    return <div>Loading...</div>;
  } else {
    return (
      <ul>
        {perfiles.map(
          perfil=><Perfil datos={perfil}  />)
        }
      </ul>
    );
  }

}
```

En el render agregamos condiciones para mostrar el listado de usuarios solo en el caso de que se haya completado la respuesta desde la api. En caso contrario mostramos el loader o bien un error.

Este es el resultado obtenido:

Bibliografía y Webgrafía 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.

https://reactjs.org/docs/faq-ajax.html

Lo que vimos:

En esta unidad aprendimos que es una api rest y un promise y cómo combinar ambos para consultar datos a un sistema externo.

Lo que viene:

En la próxima unidad aprenderemos cómo generar las rutas para nuestra aplicación reac