Xseed

Curso de React _ Technisys

Xseed

Clase 01 -

Introduccion a React, JSX y Componentes

---

by Diego Cáceres

Antes de comenzar...

Antes de comenzar...

- Instalar un editor de texto
  - Mi recomendacion es Microsoft **Visual Studio Code** (https://code.visualstudio.com/).
  - Es posible que la instalación requiera que se instale el programa Git.
  - Otra opción: Atom (https://atom.io/).
- Instalar **Node.js** (https://nodejs.org/en/).
- Unirse al **Slack** (https://slack.com/).
  - Por Slack podremos compartir material, dudas y soluciones a errores.

Antes de comenzar… ¿qué deberían saber?

- Tipos de datos.
- Operadores.
- If/Else.
- For/While Loops.
- Funciones.
  - Parámetros por referencia y por valor.
- Arrays (Arreglos).
- Objetos.

¿Qué es React?

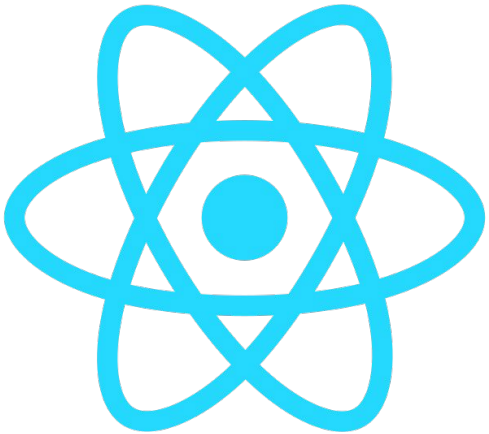
¿Qué es React? (1)

React es simplemente una **librería Javascript** para construir interfaces de usuario (UI), desarrollada y mantenida por Facebook, pero Open Source.

Cuenta con una gran comunidad de desarrolladores.

React está basado en el desarrollo de componentes, pequeños bloques que se pueden componer para poder lograr interfaces de usuario más complejas.

Documentacion: link.



¿Qué es React? (2)

React está diseñado de tal forma que, aprendiendo sus conceptos, sea muy simple desarrollar para diferentes ambientes, como web mediante **ReactDOM** o para plataformas móviles mediante **React Native**.

React Native cuenta con la ventaja de poder escribir una aplicación tanto como para **Android** como para **iOS** con básicamente el mismo código (JavaScript), pero a diferencia de otros desarrollos “Cross-Platform”, el resultado final son aplicaciones nativas.

¿Por qué React?

React es Declarativo, lo que lo hace más sencillo de desarrollar y que el código que escribimos sea más sencillo de entender. Podríamos tener el siguiente árbol de componentes, y fácilmente podemos leerlo y entender que realiza nuestra aplicación:

```
<MyApp>
  <Header>
      <Menu />
  </Header>
  <Main>
      <Products />
  </Main>
  <Footer />
</MyApp>
```

¿Por qué React?

React está basado en componentes, por lo construir una aplicación en React se puede comparar a jugar con Legos, donde cada pieza de Lego es un componente (desarrollado por nosotros, otro del equipo, o de una librería de terceros).

Cada componente debería ser lo más independiente posible, para favorecer la reutilización, pero también fácil de conectar a los demás. Cuanto más chicos, más fácil de mantenerlos, y entenderlos.

¿Por qué React?

- Se puede agregar a aplicaciones existentes de forma sencilla.
- Creada por Facebook, y utilizada en sus propios productos.
- Tiene un gran ecosistema, que brinda muchas librerías y frameworks auxiliares para utilizar en conjunto con React, como Redux, React Router, NextJS, y otras.
- Grandes jugadores la utilizan, y aportan a la comunidad y el ecosistema: Airbnb, Netflix, Apple, Instagram, Paypal, etc.
- Permite la creación de aplicaciones complejas, con UIs dinámicas y que manejan muchos datos.

Un componente en React

En React generalmente se utiliza una sintaxis llamada **JSX** (de la que se hablará posteriormente). Los componentes en React deben tener sí o sí una función render, que es donde se define cómo se dibuja el componente en la UI.

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

ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

JSX le llamamos a poder usar tags de HTML en el código Javascript.

Xseed

DOM y Virtual DOM en React

Manejo del DOM

El HTML **DOM** (Document Object Model) es el árbol de elementos que toda página web tiene. En el desarrollo web es común trabajar creando, modificando o borrando elementos del DOM (utilizando jQuery por ejemplo).

En React el DOM es manejado internamente, y nosotros sólo tenemos que actualizar ***Componentes***. React es luego el encargado de interpretar qué elementos del DOM fueron modificados, para actualizar **sólo** esos elementos.

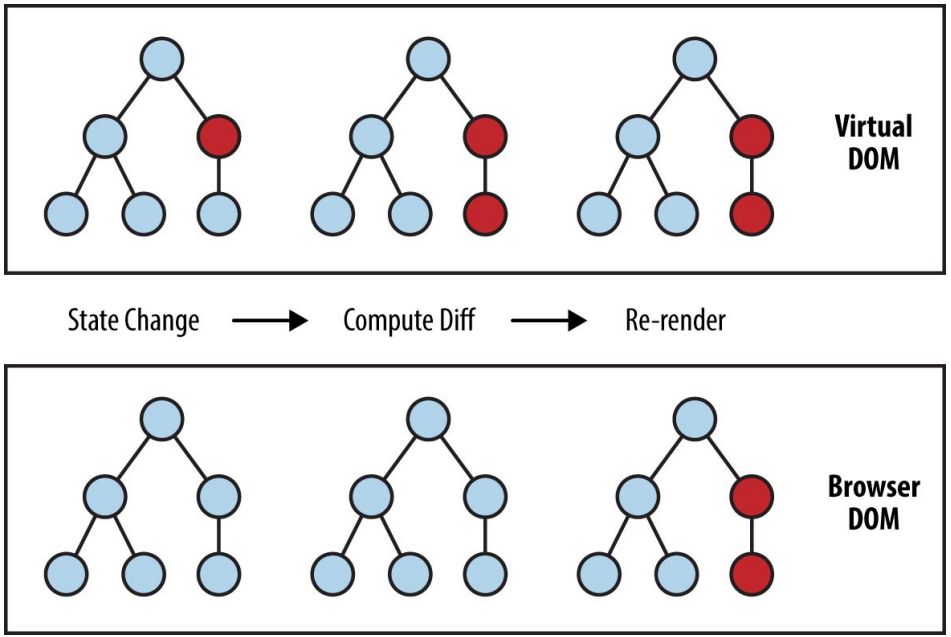
Para esto utiliza algo llamado **Virtual DOM**, propio de React, que es una copia del DOM pero mucho más liviana y rápida.

Virtual DOM

Esta es una de las grandes ventajas de React.

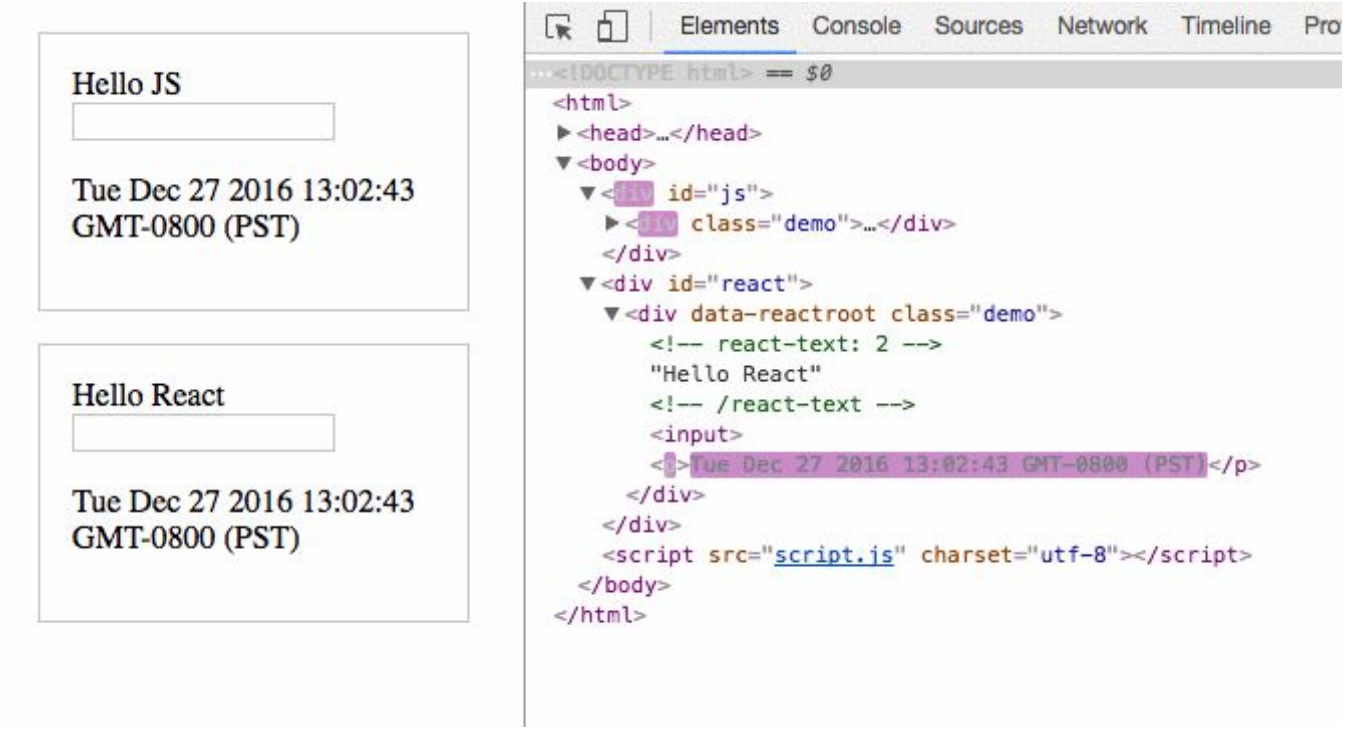
Lo hacen muy **rápido** y atractivo, ya que los elementos del DOM son pesados y caros de dibujar para los navegadores.

El proceso de actualización del DOM utiliza un algoritmo llamado Reconciliation, se puede leer más en la documentación.



Virtual DOM

Esta demo la podemos encontrar en este link.



Introduciendo JSX

JSX

```
var element = <h1>Hello, world!</h1>;
```

Esto no es HTML ni un string. Es JSX, que es una **extensión a la sintaxis de JavaScript**.
Se utilizará en React para definir cómo se debe ver la UI de un componente.

JSX produce elementos de React, que en realidad son simplemente objetos en JavaScript!

```
var element = <h1>Hello, world!</h1>;
```

Se traduce en

```
var element = React.createElement(
    "h1",
    null,
    "Hello, world!"
);
```

Pueden probar traducir cualquier código JSX a JavaScript normal online en Babel.

Expresiones en JSX

Dentro de los tags de JSX es posible utilizar cualquier expresión JavaScript, ya sea una cuenta matemática, o evaluar una función.

```
function formatName(user) {
    return user.firstName + ' ' + user.lastName;
}

var user = {
    firstName: 'José',
    lastName: 'Pérez'
};

var element = (
    <h1>
        Hello, {formatName(user)}!
    </h1>
);
```

Cuando este elemento se muestre al usuario, será con “Hello, José Pérez”.

JSX también es una expresión

Como se mencionó antes, cuando JSX es compilado pasa a ser un objeto JavaScript, por lo que puede utilizarse dentro de `if`'s, loops `for`, asignarlo a variables, aceptarlo como argumentos y retornarlo en funciones.

```
function getGreeting(user) {
    if (user) {
        return <h1>Hello, {formatName(user)}!</h1>;
    }
    return <h1>Hello, Stranger.</h1>;
}
```

Atributos en JSX

A los tags de JSX se les puede especificar atributos, al igual que sus tags equivalentes en HTML. La diferencia es que en JSX se escriben utilizando nomenclatura camelCase:

- `tabindex` se convierte en `tabIndex`
- `onclick` se convierte en `onClick`

```
const element = <div tabIndex="0"></div>;
```

Un caso a tener en cuenta: En Javascript `class` es una palabra reservada, por lo que para proporcionar una clase, en JSX usamos `className`.

```
const element = (
    <h1 className="greeting">
        Hello, world!
    </h1>
);
```

La clase greeting está definida en otro archivo CSS. Por ejemplo:

```
.greeting {
   color: "red"
}
```

JSX y ataques de inyección de código

En JSX es seguro embeber texto introducido por los usuarios, ya que JSX es seguro contra los ataques de inyección de código.

```
const title = response.potentiallyMaliciousInput;
// This is safe:
const element = <h1>{title}</h1>;
```

De forma predeterminada, React DOM “escapa” cualquier valor embebido en JSX antes de representarlos. Por lo tanto, garantiza que nunca se podrá inyectar algo que no esté escrito explícitamente en su aplicación. Todo se convierte en una cadena antes de ser renderizado. Esto ayuda a prevenir ataques XSS (cross-site-scripting).

Xseed

Componentes en React

Utilizando Componentes

Un elemento es el bloque más chico en React, y describe qué mostrar en pantalla.

```
const element = <h1>Hello, world!</h1>;
```

Para mostrar un elemento en pantalla, se necesita un nodo del DOM que se utilizará como "raíz". En general, una aplicación construida en React sólo tendrá un nodo, pero si se agrega React a una aplicación existente, puede haber varios nodos “raíz”. Utilizando `ReactDOM.render` se podrá indicarle a React dónde renderizar el componente raíz.

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(
    element,
    document.getElementById('root')
);
```

En nuestro proyecto, debe haber un html que dentro tenga un `div` con id “root” para que esto funcione.

Componentes y Props (1)

Los componentes permiten separar la UI en pedazos chicos, independientes y reutilizables.
Conceptualmente, los componentes son como **funciones** JavaScript; reciben parámetros (*props*) y retornan lo que se debe mostrar en pantalla (en JSX).

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

Esta función es un componente válido en React, y es llamado “functional component”

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

Esto es llamado “class component”, y está utilizando las clases de Javascript introducidas en ES6.

Estos dos componentes son equivalentes desde el punto de vista de React, aunque el class component tiene algunas características diferentes que veremos más adelante.

Componentes y Props (2)

Hasta ahora se vieron elementos de React que utilizan sólo tags de HTML, pero se pueden crear elementos a partir de los Componentes que nosotros mismos definimos.

```
// Definimos el Componente
function WelcomeMessage(props) {
    return <h1>Hello, {props.name}</h1>;
}

// Utilizamos el Componente
var element = <WelcomeMessage name="Anne" />;
ReactDOM.render(
    element,
    document.getElementById('root')
);
```

Cuando React encuentra un elemento definido por nosotros, le pasa los atributos definidos en JSX al componente como ***props***.

Componentes y Props (3)

Lo que está sucediendo en el bloque de código anterior es:

- Se llama a `ReactDOM.render()` con el elemento `<WelcomeMessage name="Anne" />`
- `React` llama al componente `WelcomeMessage` con las props `{ name: "Anne" }`
- Nuestro componente `WelcomeMessage` retorna `<h1>Hello, Anne</h1>`
- React DOM eficientemente actualiza el DOM para que coincida con `<h1>Hello, Anne</h1>` dentro del `div` con `id` "root".

Componer Componentes

Como se mencionó, la idea de los *componentes* en React es poder componerlos y crear UIs complejas de forma sencilla.

```
function WelcomeMessage(props) {
    return <h1>Hello, {props.name}</h1>;
}
function App() {
    return (
        <div>
            <WelcomeMessage name="Anne" />
            <WelcomeMessage name="Marta" />
            <WelcomeMessage name="Lucía" />
        </div>
    );
}
ReactDOM.render(<App />, document.getElementById('root'));
```

Como regla, debemos nombrar nuestros componentes comenzando con mayúscula, para que React los distinga con los tags propios de HTML.

Como queremos devolver más de un elemento, una opción es envolverlos en un `<div>`, pero desde React v16 también podemos devolver un array de elementos separados por coma.

Componer Componentes

Estas dos formas también son válidas para retornar una lista de elementos, sin necesidad de crear otro nodo del DOM.

```
function WelcomeMessage(props) {
    return <h1>Hello, {props.name}</h1>;
}
function App() {
    return [
         <WelcomeMessage name="Anne" />,
         <WelcomeMessage name="Marta" />,
         <WelcomeMessage name="Lucía" />
    ];
}
ReactDOM.render(<App />,
     document.getElementById('root'));
```

```
function WelcomeMessage(props) {
    return <h1>Hello, {props.name}</h1>;
}
function App() {
    return (
      <>
         <WelcomeMessage name="Anne" />
         <WelcomeMessage name="Marta" />
         <WelcomeMessage name="Lucía" />
      </>
    );
}
ReactDOM.render(<App />,
     document.getElementById('root'));
```

Ejercicios

Instalaciones Previas

- Para trabajar con React es necesario tener instalado **Node.js**: https://nodejs.org/en/
  - Esto también instalará `npm` que se utilizará constantemente en la consola para instalar paquetes a nuestros proyectos.

- Lo otro que se necesita es un editor de texto. Se podrá utilizar Atom, Sublime, **Visual Studio Code** o algún otro.
  - En el caso de Sublime es necesario instalar además un plugin para que interprete JSX. Atom y Visual Studio Code lo traen incluido.

Proyecto base

Para arrancar a trabajar en un proyecto de React de forma rápida se utilizará un proyecto llamado `create-react-app` el cual creará un proyecto ya configurado y listo para utilizar.

1. Abrir una consola e instalar, por única vez, el proyecto con el comando:
   ```
   $ npm install -g create-react-app
   ```

2. Luego de que la instalación termine, ejecutar el siguiente comando para crear un proyecto nuevo. Este comando se usará cada vez que se cree un nuevo proyecto:
   ```
   $ create-react-app my-first-react-app
   ```

3. Luego, desde la consola, "pararse" en la carpeta del nuevo proyecto e iniciarlo.
   ```
   $ cd my-first-react-app
   $ npm start
   ```
   En caso de que el browser no se abra de forma automática, entrar a http://localhost:3000/.

Ejercicio 1

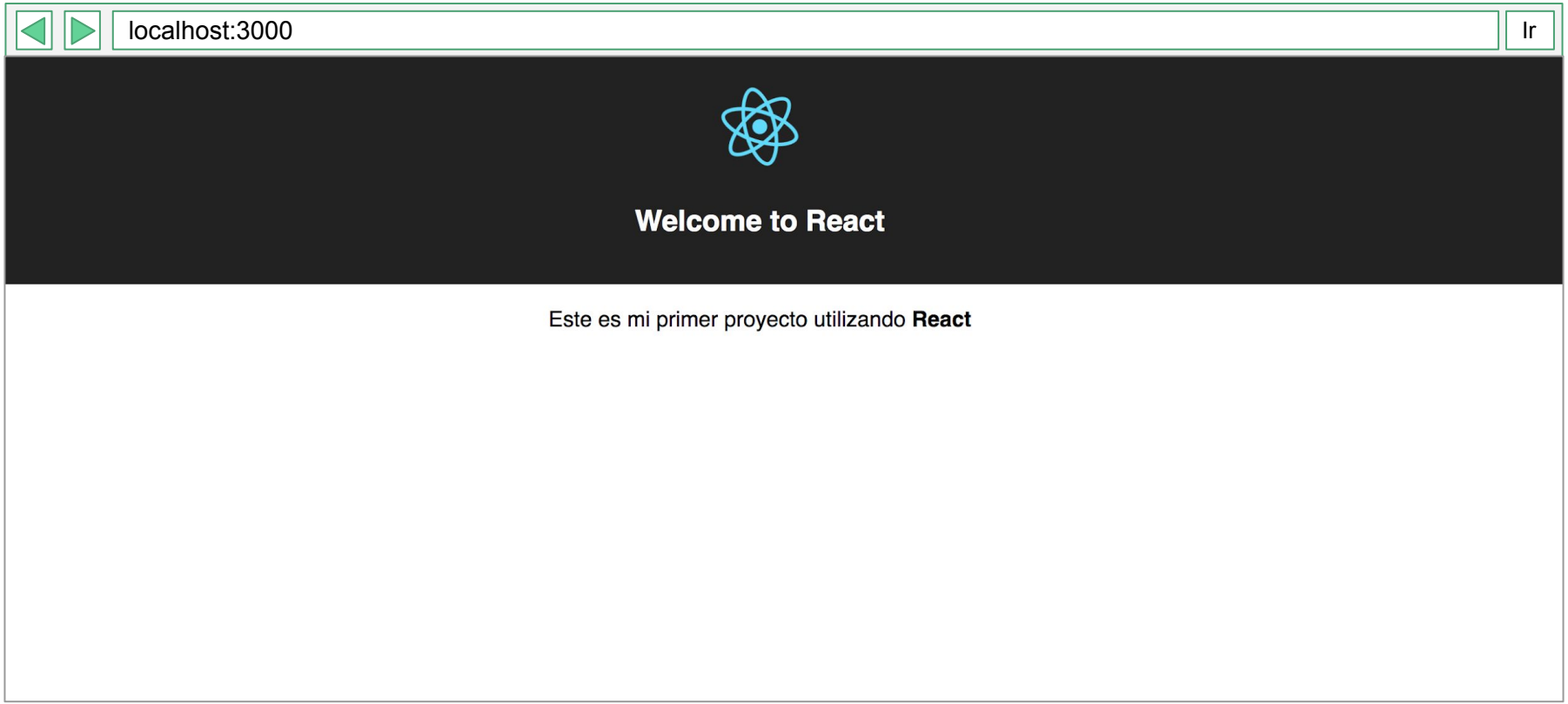
1. Crear un nuevo proyecto React llamado **`my-first-react-app`**.

2. Abrir dicha carpeta en VSC o Atom (como un **proyecto**).

3. Desde la consola, parados en la carpeta del proyecto, iniciar el mismo (`npm start`).

4. Revisar el archivo `src/App.js`, probar hacer algunos cambios en los textos. Guardar y revisar en el navegador los cambios. (Notar que no es necesario recargar el navegador para ver los cambios, esto es gracias a un paquete llamado “Hot-Reloading” que ya viene instalado por `create-react-app` que hace nuestro desarrollo mucho más rápido)

Ejercicio 1 (cont)



Ejercicios

Descargar el **zip** `EjerciciosClase01`. Cada carpeta dentro corresponde a un ejercicio, y es un proyecto creado con Create React App, por lo que para ejecutarlo es necesario:

- Si usan yarn:
  - `$ yarn install`
  - `$ yarn start`
- Si usan npm:
  - `$ npm install`
  - `$ npm start`

Se debería abrir automáticamente el explorador con el ejercicio corriendo en http://localhost:3000/.

Ejercicio 2

1. Modificar `index.js` para que el componente `App` lo importe desde el archivo `AppEjercicio2.js`.

```
import App from './AppEjercicio2';
```

2. Completar el componente `Badge` de forma que muestre la imagen, el nombre y el apodo del usuario Luis Suárez (desde el componente `App` le pueden pasar por *props* los valores a `Badge`).

Ejercicio 3

1. Modificar `index.js` para que el componente `App` lo importe desde el archivo `AppEjercicio3.js`.

```
import App from './AppEjercicio3';
```

2. Completar el componente `Badge`, `Avatar`, `Name` y `NickName` de forma que muestre la imagen, el nombre y el apodo del usuario Luis Suárez logrando el mismo resultado final que el ejercicio anterior (tener cuidado con el componente `NickName`, ya que es un *functional component*, o sea que accede a los valores de *props* un poco diferente).

Ejercicio 4

1. Modificar `index.js` para que el componente `App` lo importe desde el archivo `AppEjercicio4.js`.

```
import App from './AppEjercicio4';
```

2. El componente `Comment` se encarga de bastantes cosas y es poco reutilizable, por lo que se debe separar en componentes más chicos:

   a. Crear un nuevo componente llamado `Avatar` que sólo se encargue de mostrar la imagen.

   b. Luego crear otro componente llamado `UserInfo` encargado de mostrar el texto junto al `Avatar` (`Comment` usará `UserInfo`, que dentro utilizará `Avatar`).

   c. Asegurarse que el resultado final sea igual al inicial.