Xseed

Curso de React _ Technisys

Xseed

Clase 07 -

Routing en React, High Order Components

by Diego Cáceres

Repaso

Lifecycle Methods

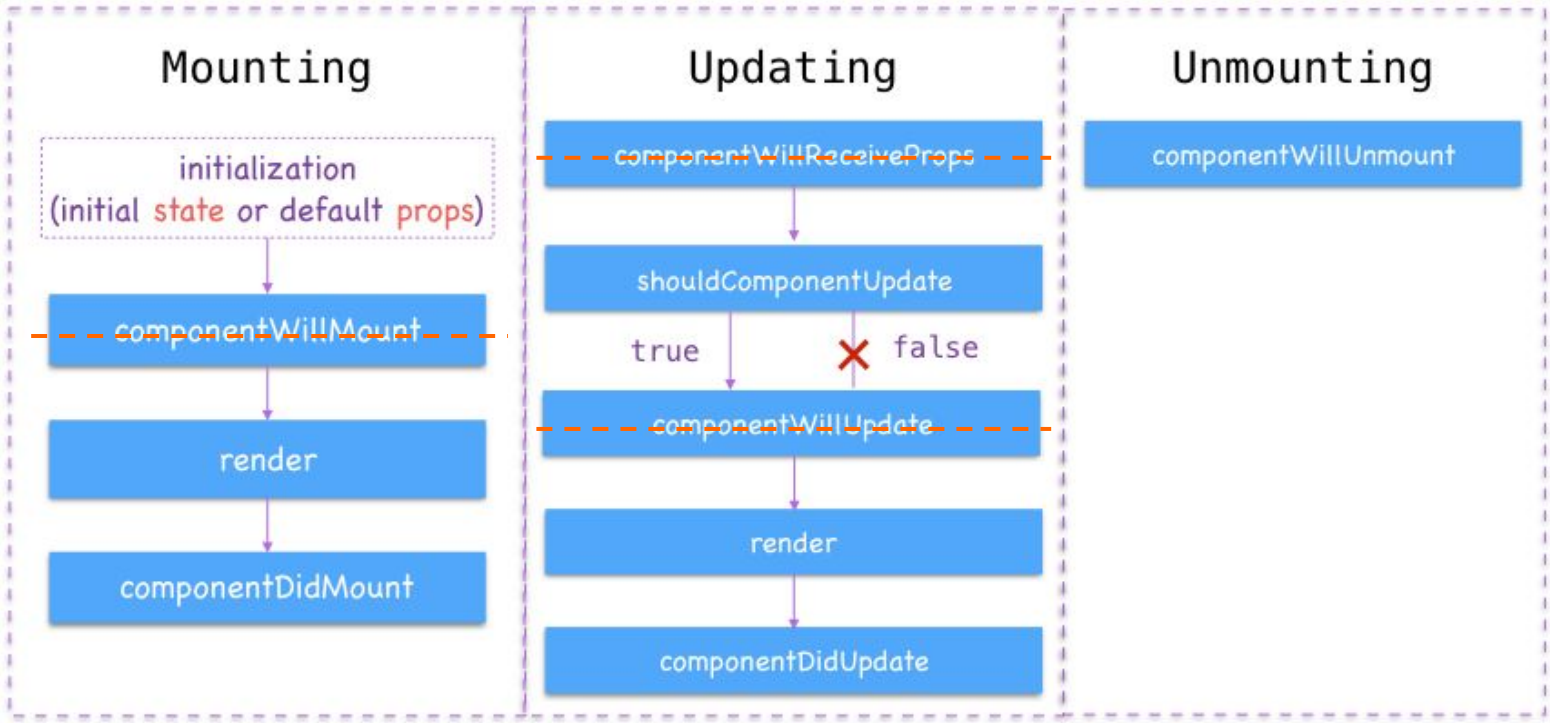
Todos los componentes en React tienen un ciclo de vida, empezando cuando son montados en pantalla, cuando son actualizados, y cuando se desmontan.

Una de las diferencias entre los Class Components y los Functional Components es que en Class Components se puede sobreescribir los métodos de ciclo de vida para programar acciones que suceden en cada uno de ellos.

Esto permite, por ejemplo, ir a buscar datos a una API exactamente después de que se dibuja por primera vez el componente.

Lifecycle Methods

Se muestran a continuación algunos diagramas para entender esto.



“Render Phase”

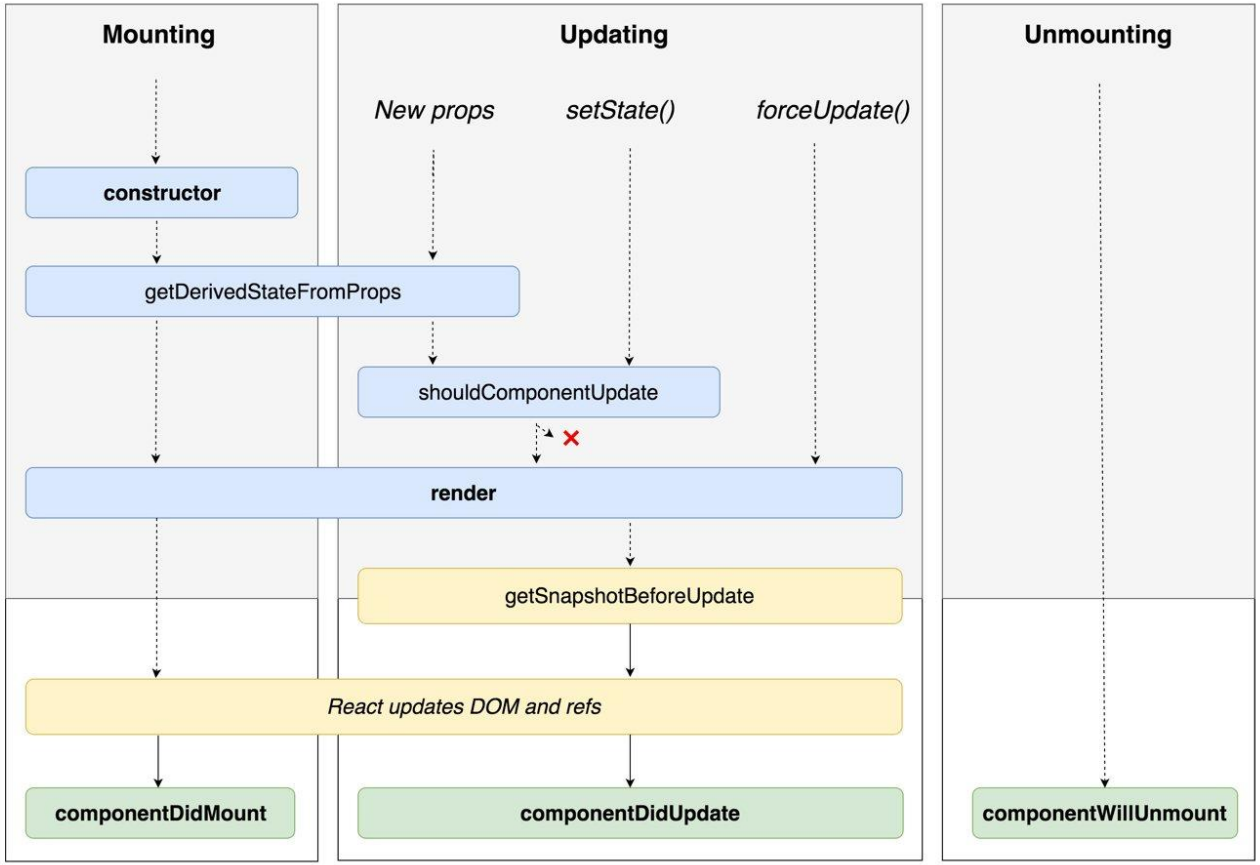
Pure and has no side effects. May be paused, aborted or restarted by React.

“Pre-Commit Phase”

Can read the DOM.

“Commit Phase”

Can work with DOM, run side effects, schedule updates.



Lifecycle Methods – Ejemplo

Xseed
REPASO

```
class MyComponent extends Component {
    constructor(props) {
        super(props);
    }
    componentDidMount() {
        console.log("Se acaba de montar el componente");
    }
    componentWillUnmount() {
        console.log("Se acaba de des-montar el componente");
    }
    render() {
        return (
            <div>
                <h1>Hello, world!</h1>
            </div>
        );
    }
}
```

External Data Access – Axios GET

Como Axios está basado en Promesas, se pueden realizar llamadas `GET` y programar qué hacer con el resultado en `then`, sin necesidad de utilizar callbacks.

```
axios.get('/user?ID=12345')
.then(function (response) {
    return axios.get('/comment)
})
.catch(function (error) {
    console.log(error);
});
```

```
axios.get('/user', {
    params: {
        ID: 12345
    }
})
.then(function (response) {
    console.log(response);
})
.catch(function (error) {
    console.log(error);
});
```

También se pueden pasar los parámetros como un objeto.

Xseed

REPASO

External Data Access – Axios POST

Para hacer una llamada de tipo `POST` simplemente se manda un segundo parámetro conteniendo un objeto con lo que se quiere que contenga el `body` de la llamada.

```
axios.post('/user', {
     firstName: 'Diego',
     lastName: 'Caceres'
})
.then(function (response) {
     console.log(response);
})
.catch(function (error) {
     console.log(error);
});
```

```
const baseUrl = "https://myapp.com/api";
axios.post(`${baseUrl}/user`, {
    firstName: 'Diego',
    lastName: 'Caceres'
})
.then(function (response) {
     console.log(response);
})
.catch(function (error) {
     console.log(error);
});
```

Pasando la URL completa.

Presentational – Container (1)

Es un patrón muy utilizado en React ya que facilita la reutilización de componentes. También es llamado patrón Container Component.

La idea es bastante sencilla; un contenedor se encarga de obtener los datos y luego renderiza su correspondiente sub-componente, llamado Presentational Component (Componente de Presentación).

Normalmente, para nombrarlos se mantiene parte del nombre:

- `StockWidgetContainer` **=>** `StockWidget`
- `TagCloudContainer` **=>** `TagCloud`

Xseed

Routes en React

Routes

El Routing es un aspecto importante a la hora de desarrollar una aplicación web.

Esto se refiere principalmente a las distintas "rutas" de nuestra App, por ejemplo, si existe una sección que sólo se accede con credenciales. Otro caso, en una App del estilo Marketplace tendremos una sección central de navegación y luego una sección de "Ver mi carrito".

En React esto ha sido implementado de varias formas por varias librerías. Algunas de las más conocidas son:

- React Router
- Redux First Router
- React Navigation (React Native)
- React Mini Router
- Etc.

React Router v4 (1)

Durante el curso se va a estar usando **React Router v4**, que es la librería más comúnmente utilizada. Además, tiene una muy buena documentación.

En React Router, la mayoría de los elementos de navegación son Componentes de React, tratando de mantener el modo de pensar declarativo que se utiliza al desarrollar aplicaciones en React.

Para poder utilizarlo, se necesita instalarlo en el proyecto por `npm` o `yarn`:

- `$ npm install --save 'react-router-dom'`
- `$ yarn add 'react-router-dom'`

React Router v4 (2)

Se van a utilizar algunos de los componentes que brinda la librería para construir el ruteo de la App. Por ejemplo, **`BrowserRouter`** se utiliza para envolver toda la App y **`Route`** para declarar **qué rutas** renderizan **qué componentes**.

```
const BasicExample = () => {
   return (
       <BrowserRouter>
           <div>
               <Route exact path="/" component={Home} />
               <Route path="/about" component={About} />
           </div>
       </BrowserRouter>
   );
}
```

React Router v4 (3)

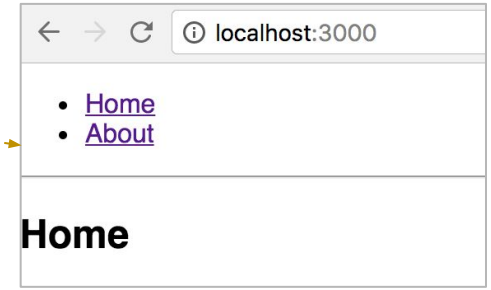
El componente **`Link`** sirve para declarar navegación a rutas determinadas. Estos componentes hay que importarlos de la librería antes de poder utilizarlos.

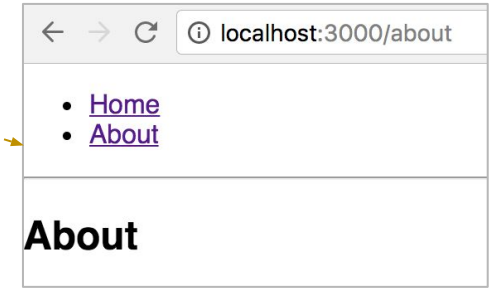
```
import { BrowserRouter, Route, Link } from "react-router-dom";
const BasicExample = () =>
   <BrowserRouter>
       <div>
           <ul>
               <li><Link to="/">Home</Link></li>
               <li><Link to="/about">About</Link></li>
           </ul>
           <hr />
           <Route exact path="/" component={Home} />
           <Route path="/about" component={About} />
       </div>
   </BrowserRouter>;
```

`exact` se utiliza para que no "matchee" `Home` en `About`

React Router v4 (4)

```
import { BrowserRouter as Router, Route, Link } from "react-router-dom";
const Home = () =>
    <div>
        <h2>Home</h2>
    </div>;
const About = () =>
    <div>
        <h2>About</h2>
    </div>;
const BasicExample = () =>
    <Router>
        <div>
            <ul>
                <li><Link to="/">Home</Link></li>
                <li><Link to="/about">About</Link></li>
            </ul>
            <Route exact path="/" component={Home} />
            <Route path="/about" component={About} />
        </div>
    </Router>;
```





React Router v4 (5)

El componente **NavLink** es muy similar a **Link** pero permite darle además una propiedad **activeClassName** para que si la ruta actual coincide con la ruta del link, le aplique la clase de **css** que se le haya pasado es esta propiedad.

```
import { BrowserRouter, Route, NavLink } from "react-router-dom";
const BasicExample = () =>
    <BrowserRouter>
        <div>
            <ul>
                <li><NavLink exact activeClassName="active" to="/">Home</NavLink></li>
                <li><NavLink activeClassName="active" to="/about">About</NavLink></li>
            </ul>
            <hr />
            <Route exact path="/" component={Home} />
            <Route path="/about" component={About} />
        </div>
    </BrowserRouter>;
```

React Router v4 (6)

Todos los componentes que son dibujados por **Route** van a recibir en sus **props** datos específicas del ruteo, con la siguiente forma: **{match, location, history}**.
Dentro de **match**, podemos acceder a la URL por ejemplo:

```
const BasicExample = () =>
 <Router>
   <div>
     <ul>
       <li><Link to="/">Home</Link></li>
       <li><Link to="/topics">Topics</Link></li>
     </ul>
     <hr />
     <Route exact path="/" component={Home} />
     <Route path="/topics" component={Topics} />
   </div>
 </Router>;
```

```
const Topics = (props) =>
   <div>
       <h2>Topics</h2>
       <h3>La url es: {props.match.url}</h3>
   </div>;
```

```
const Topics = ({match, location, history})=>
   <div>
       <h2>Topics</h2>
       <h3>La url es: {match.url}</h3>
   </div>;
```

React Router v4 (7)

Si queremos anidar **`Routes`**, osea, colocar una ruta dentro de otra, lo podemos hacer de la misma forma que típicamente anidamos elementos html, como los divs:

```
const App = () => (
  <BrowserRouter>
    {/* here's a div */}
    <div>
      {/* here's a Route */}
      <Route path="/tacos" component={Tacos}/>
    </div>
  </BrowserRouter>
)
```

```
// when the url matches `/tacos` this component renders
const Tacos  = ({ match }) => (
  // here's a nested div
  <div>
    {/* here's a nested Route,
        match.url helps us make a relative path */}
    <Route
      path={match.url + '/carnitas'}
      component={Carnitas}
    />
  </div>
)
```

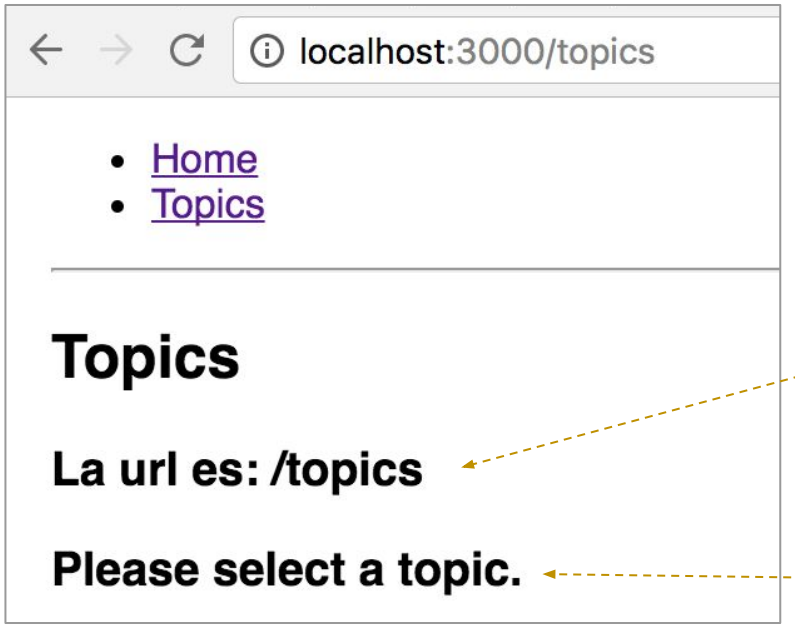
React Router v4 (8)

Cuando **Route** "matchee" la ruta, va a dibujar el componente que reciba en la prop **component**. También se puede usar la propiedad **render** que recibe una función y la evalúa para determinar qué dibujar.

```
const BasicExample = () =>
 <Router>
   <div>
     <ul>
       <li><Link to="/">Home</Link></li>
       <li>
          <Link to="/topics">Topics</Link>
       </li>
     </ul>
     <hr />
     <Route exact path="/" component={Home} />
     <Route path="/topics" component={Topics} />
   </div>
 </Router>;
```

```
const Topics = ({ match }) =>
 <div>
   <h2>Topics</h2>
   <h3>La url es: {match.url}</h3>
   <Route exact path={match.url}
     render={
       () => <h3>Select a topic.</h3>
     }
   />
 </div>;
```

React Router v4 (9)



Este es el valor de: {`match.url`}

Esto aparece solo porque hay un `Route` que "matchea" exactamente con {`match.url`} dentro de `Topics`

React Router v4 (10)

`render` generalmente lo utilizaremos además cuando queramos pasar al componente asociado a la ruta, alguna variable que tengamos en el scope local

```
const App = () => {
  const someVariable = true;
  return (
    <div>
      <Route exact path='/' component={Home} />
      <Route path='/about'
          render={(props) => <About {...props} extra={someVariable}  />}
      />
    </div>
  )
}
```

React Router v4 (11)

Dentro de la prop **match** que reciben los componentes, hay una propiedad **param** donde van a llegar todos los parámetros que se definan en la ruta, con el mismo nombre con el que fueron definidos (en este ejemplo, el nombre es **topicId**).

```
const Topics = ({ match }) =>
    <div>
        <h2>Topics</h2>
        <Route path={`${match.url}/:topicId`} component={Topic} />
        <Route
            exact path={match.url}
            render={() => <h3>Please select a topic.</h3>} />
    </div>;

const Topic = ({ match }) =>
    <div>
        <h3>{match.params.topicId}</h3>
    </div>;
```

React Router v4 (12)

```
import React from "react";
import { BrowserRouter as Router, Route, Link } from "react-router-dom";

const BasicExample = () =>
    <Router>
        <div>
            <ul>
                <li><Link to="/">Home</Link></li>
                <li><Link to="/about">About</Link></li>
                <li><Link to="/topics">Topics</Link></li>
            </ul>
            <hr />
            <Route exact path="/" component={Home} />
            <Route path="/about" component={About} />
            <Route path="/topics" component={Topics} />
        </div>
    </Router>;
export default BasicExample;
```

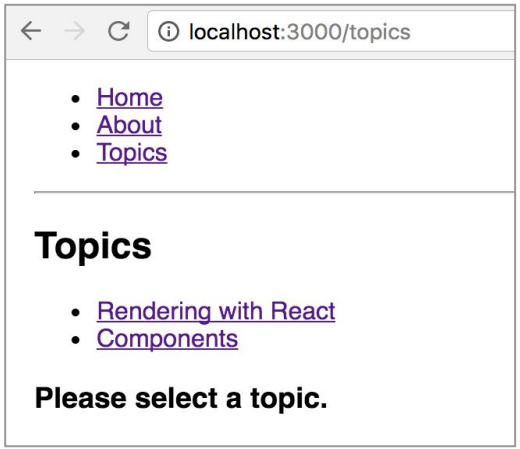
React Router v4 (13)

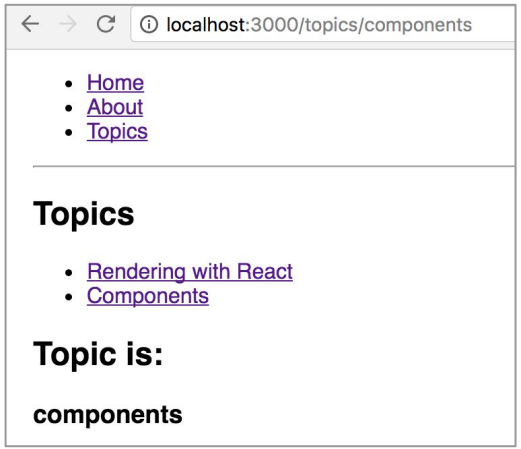
```
const Topics = ({ match }) =>
    <div>
        <h2>Topics</h2>
        <ul>
            <li>
                <Link to={`${match.url}/rendering`}>
                    Rendering with React
                </Link>
            </li>
            <li>
                <Link to={`${match.url}/components`}>
                    Components
                </Link>
            </li>
        </ul>
        <Route path={`${match.url}/:topicId`} component={Topic} />
        <Route exact path={match.url} render={() => <h3>Please select a topic.</h3>} />
 </div>;
```

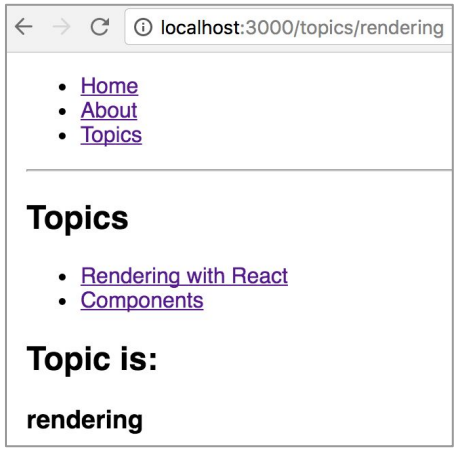
React Router v4 (14)

```
const Home = () =>
    <div>
        <h2>Home</h2>
    </div>;

const About = () =>
    <div>
        <h2>About</h2>
    </div>;

const Topic = ({ match }) =>
    <div>
        <h2>Topic is: </h2>
        <h3>{match.params.topicId}</h3>
    </div>;
```

React Router v4 (15)







React Router v4 (16)

También es posible usar expresiones regulares para limitar cuales son los posibles parámetros que una ruta acepta.

```
{/*
        * "/order/asc"  - matched
        * "/order/desc" - matched
        * "/order/foo"  - not matched
*/}
<Route
      path="/order/:direction(asc|desc)"
      component={ComponentWithRegex}
/>
```

React Router v4 (17)

En el caso de tener rutas **ambiguas**, en la que dos pueden matchear, se puede utilizar el **`Switch`** para envolver las rutas y va a devolver la primera que coincida.

```
const AmbiguousExample = () => <Router>
  <div>
    <ul>
      <li><Link to="/about">About Us (static)</Link></li>
      <li><Link to="/company">Company (static)</Link></li>
      <li><Link to="/kim">Kim (dynamic)</Link></li>
    </ul>
    <Switch>
      <Route path="/about" component={About} />
      <Route path="/company" component={Company} />
      <Route path="/:user" component={User} />
    </Switch>
  </div>
</Router>;
```

En este caso, cuando se navegue a `/about` o `/company` la tercer ruta también "matchea", pero el `Switch` devuelve solo la primera que coincida.

React Router v4 (18)

```
import React from "react";
import { BrowserRouter as Router, Route, Link, Switch } from "react-router-dom";
const AmbiguousExample = () => <Router>
   <div>
     <ul>
       <li><Link to="/about">About Us (static)</Link></li>
       <li><Link to="/company">Company (static)</Link></li>
       <li><Link to="/kim">Kim (dynamic)</Link></li>
       <li><Link to="/chris">Chris (dynamic)</Link></li>
     </ul>
     <Switch>
       <Route path="/about" component={About} />
       <Route path="/company" component={Company} />
       <Route path="/:user" component={User} />
     </Switch>
   </div>
 </Router>;
const About = () => <h2>About</h2>;
const Company = () => <h2>Company</h2>;
const User = ({ match }) => <h2>User: {match.params.user}</h2>;
```

React Router v4 (19)

Se puede utilizar el componente **Redirect** para indicar cuándo se quiere que determinada ruta cambie a otra automáticamente. Otro detalle importante es que un **Route** sin **path** siempre va a matchear, por eso en el siguiente caso se envuelve en un **Switch**, pero sirve para las rutas no definidas.

```
<Switch>
    <Route path="/" exact component={Home} />
    <Redirect from="/old-match" to="/will-match" />
    <Route path="/will-match" component={WillMatch} />
    <Route component={NoMatch} />
</Switch>
```

Xseed

React Router v4 (17)

```
import { BrowserRouter as Router, Route,
           Link, Switch,  Redirect } from "react-router-dom";
const NoMatchExample = () =>
<Router>
 <div>
   <ul>
     <li><Link to="/">Home</Link></li>
     <li><Link to="/old-match">Old, to be redirected</Link></li>
     <li><Link to="/will-match">Will Match</Link></li>
     <li><Link to="/will-not-match">Will Not Match</Link></li>
   </ul>
   <Switch>
     <Route path="/" exact component={Home} />
     <Redirect from="/old-match" to="/will-match" />
     <Route path="/will-match" component={WillMatch} />
     <Route component={NoMatch} />
   </Switch>
 </div>
</Router>;
```

```
const Home = () => <p> Home </p>;

const WillMatch = ({ match }) =>
 <div>
   <h2>Match.url is: {match.url}</h2>
   <h3>Matched!</h3>
 </div>;

const NoMatch= ({ match, location }) =>
 <div>
   <h2>Match.url is: {match.url}</h2>
   <h3>No match for
{location.pathname}</h3>
 </div>;
```

HOC - High Order Components

High Order Components (1)

Un High-Order Component es una técnica avanzada utilizada para reutilizar la lógica de un componente. No es algo específico de la API de React, sino que es un patrón que surge naturalmente de la posibilidad de composición que hay en React.

Un High-Order Component es una **función** que **acepta un componente** y **retorna un nuevo componente**.

```
const higherOrderComponent = (component) => {
    return componentOnSteroids;
}
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

High Order Components (2)

*Un componente básicamente transforma `Props` en UI, mientras que un **High-Order Component** transforma un Componente en otro Componente, agregando nuevas características.*

High Order Components (3)

Normalmente los High-Order Components son utilizados por las librerías de terceros, como se va a ver en Redux. Pero también se pueden crear HOCs propios. Un ejemplo de esto sería, si en determinado momento quisiera envolver todo un componente con un div especial para visualmente identificarlo en pantalla.

```
let DebugComponent = ComponentToDebug => {
   return class extends Component {
       render() {
         return (
           <div className="debug">
             <ComponentToDebug {...this.props}/>
           </div>
         );
       }
   };
}
```

```
<Switch>
   <Route path="/" exact component={Home} />
   <Redirect from="/old-match" to="/will-match" />
   <Route path="/will-match" component={WillMatch} />
   <Route component={DebugComponent(NoMatch)} />
</Switch>
```

High Order Components (4)

Otro uso común, es acceder a las `Props` del componente y modificarlas, borrar determinada `prop`, o agregar más:

```
function AddPropsHOC(WrappedComponent) {
    return class extends Component {
        render() {
            const newProps = {
                user: currentLoggedInUser,
                permisssions: currentUserPermissions
            }
            return <WrappedComponent {...this.props} {...newProps}/>
        }
    }
}
```

High Order Components – React Router

React Router brinda un HOC llamado **`withRouter`** que permite hacer que cualquier componente pueda recibir las mismas **`props`** que un componente renderizado por un **`Route`**.

```
class ShowTheLocation extends React.Component {
  render() {
   const { match, location, history } = this.props
   return (
     <div>You are now at {location.pathname}</div>
   )
 }
}

const ShowTheLocationWithRouter =
withRouter(ShowTheLocation)
```

```
<Router>
   <div>
       <ShowTheLocationWithRouter />
       <ul>
           <li><Link to="/">Home</Link></li>
           <li>
               <Link to="/match"> Match</Link>
            </li>
       </ul>
       <Route path="/" exact component={Home} />
       <Route path="/match" component={WillMatch} />
   </div>
</Router>
```

Ejercicios

Ejercicios

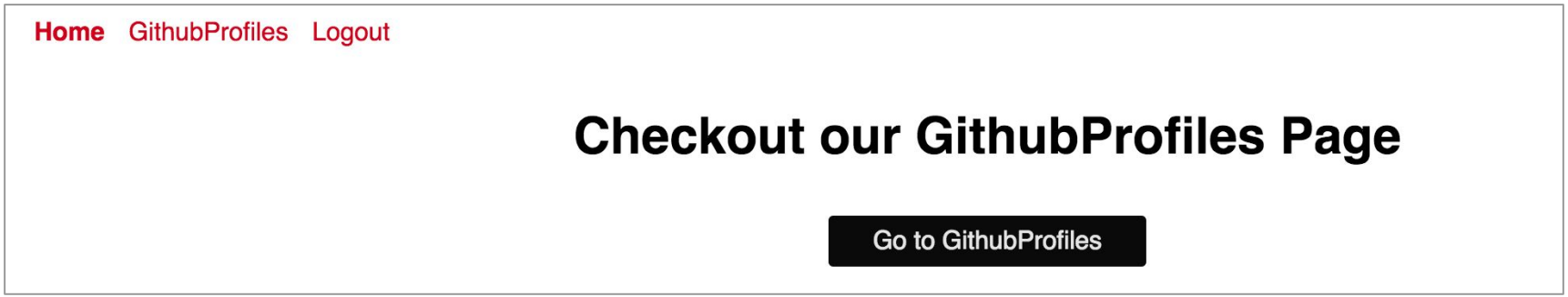
Descargar el zip `EjerciciosClase07`. Cada carpeta dentro corresponde a un ejercicio y es un proyecto creado con Create React App, por lo que para ejecutarlo es necesario:

- 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 1 – Parte 1

Este proyecto ahora siempre muestra el componente `GithubProfiles` que se construyó anteriormente. Ahora se va a construir un sitio con 3 rutas, similar a este:



Ejercicio 1 – Parte 1 (cont.)

- Primero es necesario instalar en el proyecto `react-router-dom`
- Modificar `App.js` y agregar un `Route` para que solo muestre `GithubProfiles` en la ruta `/githubprofiles` (No olvidarse de envolver todo utilizando el `BrowserRouter`).
- Crear un componente nuevo en `src/components` llamado `Nav`, que devuelva un `ul` con tres `li`. Al `ul` ponerle como clase "nav". Dentro de cada `li` agregar un `NavLink` con el nombre de las rutas: **Home**, **GithubProfiles** y **Logout**. Darle como `activeClassName` "active". Cada link tiene que dirigir a la ruta correcta.
- Agregar en `App.js` que muestre este componente `Nav` antes de definir las rutas.

Ejercicio 1 – Parte 1 (cont.)

- Crear otro componente en `src/components` llamado `Home`, que dentro retorne un `div` (darle la clase "home-container"). Dentro del `div`, colocar un `h1` con un texto y un `Link` que redirija a `GithubProfiles` (al `Link` le pueden dar la clase "button"). Agregar el `Route` correspondiente en `App.js` para el `Home`.
- Agregar otro `Route` que dirija a `/logout` y muestre un nuevo componente `Logout` que deben crear. Este componente tiene que tener simplemente un `div` con un `button` que diga "Logout" por ahora.

Ejercicio 1 – Parte 2

Si ahora entran a una ruta (a mano) en el navegador no mostrará nada.

- Cambiar en `App.js` para envolver las rutas en un `Switch`. Crear un nuevo componente llamado `NotFound` que muestre un mensaje cuando se entra a una ruta errónea, y agregar un `Route` sin path que muestre ese componente.
- Hay que agregar una pantalla de `Login` a la App. Crear un nuevo componente en `src` (a la misma altura que `App`) llamado `LoginManager`. Este componente tiene que ser `Class Component`, y en su `state` va a manejar un boolean para indicar si está logueado o no. En su `render`, si no esta logueado mostrará un mensaje de bienvenida, y un botón de Login que en su `onClick` cambie el state a "logueado". Además, en su `render`, si está logueado mostrará `<App />` (tienen que importar el componente `App` en `LoginManager`).
- Luego modificar en index.js para que en vez de mostrar `<App />`, ahora muestre `<LoginManager />` como componente raíz.

Ejercicio 1 – Parte 2 (cont.)

De alguna forma desde el componente `Logout`, que está dentro de `App`, se debe poder cambiar el estado del componente `LoginManager` para poder cerrar sesión. Para resolver esto podemos pasar por props una función desde `LoginManager` hasta `Logout`, y en `Logout` invocarla en el `onClick` del `button`.

Ahora, hay un detalle. Para pasar la prop desde App a Logout, está el problema de que App renderiza Logout a través de un Route, de la siguiente forma:

```
<Route path="/Logout" component={Logout} />
```

¿Cómo se le pasa props? => Se puede utilizar el `render` de `Route`:

```
<Route path="/Logout"
   render={() => <Logout onHandleClick={this.props.onLogout} />}
/>
```

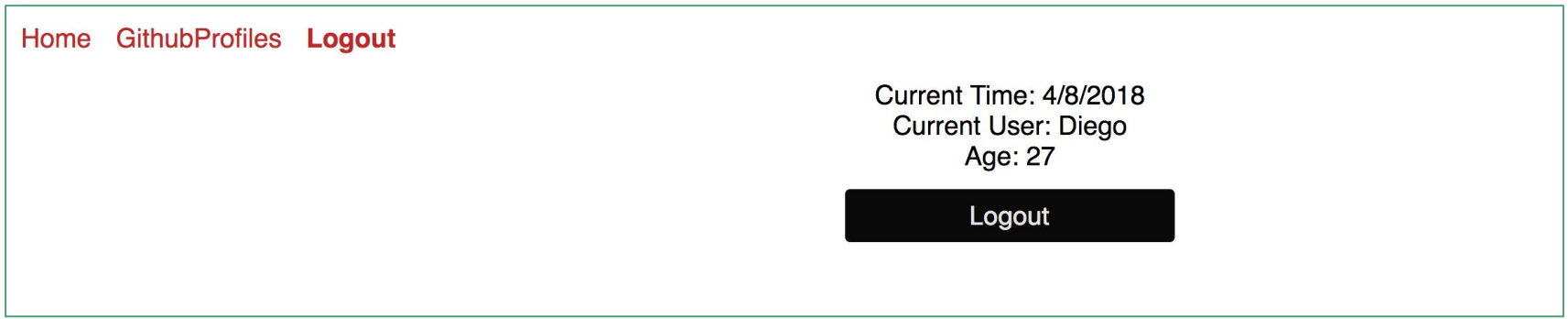
Ejercicio 1 – Parte 2 (cont.)

Por último, se va a implementar un **High-Order Component** llamado `HighLightComponent` que simplemente envuelva a un componente en un `div` con la clase "debug", de forma que lo pinte en pantalla para ubicarlo.

Probarlo, por ejemplo, con `Home`:

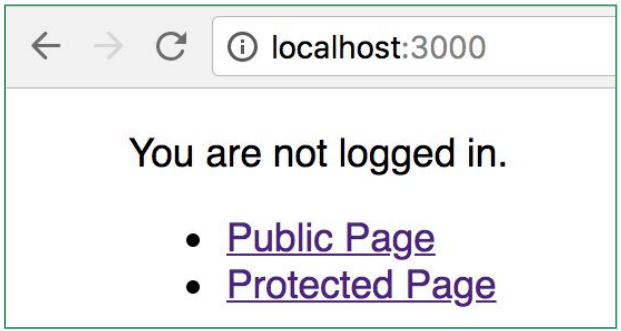
Ejercicio 1 – Parte 3

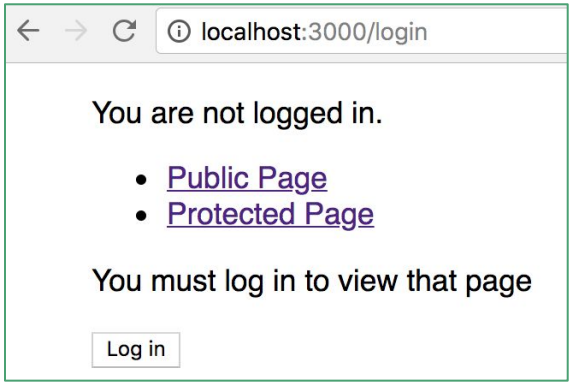
- Agregar en la pantalla de login dos inputs para solicitar al usuario el nombre y su edad antes de hacer Login.
- En el componente de Logout, agregar que muestre un nuevo componente llamado `SystemInformation` (deben crearlo) que mostrará la hora actual, y también mostrará otro componente llamado `CurrentUserProfile`.
- `CurrentUserProfile` debe recibir y mostrar el nombre y la edad del usuario que hizo Login.

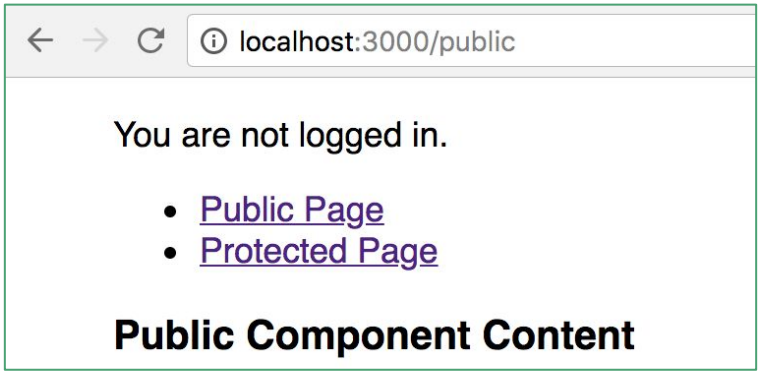


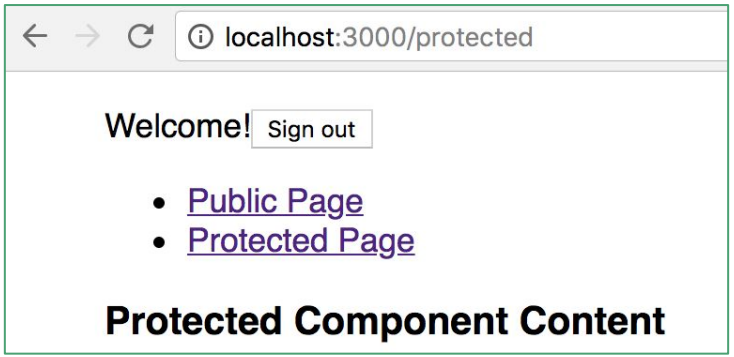
Ejercicio 2

En este ejercicio hay que completar los componentes de forma de obtener el concepto de ruta privada, que se comporte de la siguiente forma:









Ejercicio 2

- Primero completar el componente **`Login`** para que cambie la bandera a true.

- Luego completar el **`AuthButton`** para el signOut. (lo de resetear la ruta en signout lo pueden dejar para el final)

- En el componente **`PrivateRoute`** hay que chequear el valor de **`fakeAuth.isAuthenticated`** para determinar si mostrar el componente que llega por props, o la redirección. Tener en cuenta, que si se muestra el componente que llega, deben también pasar el resto de las props.

- Por último, para resetear la ruta desde el signout en **`AuthButton`**, necesitamos acceder a la propiedad de ruteo `history`, para hacer un `history.push('/')`. Quizás podemos utilizar un HoC...