React intermedio

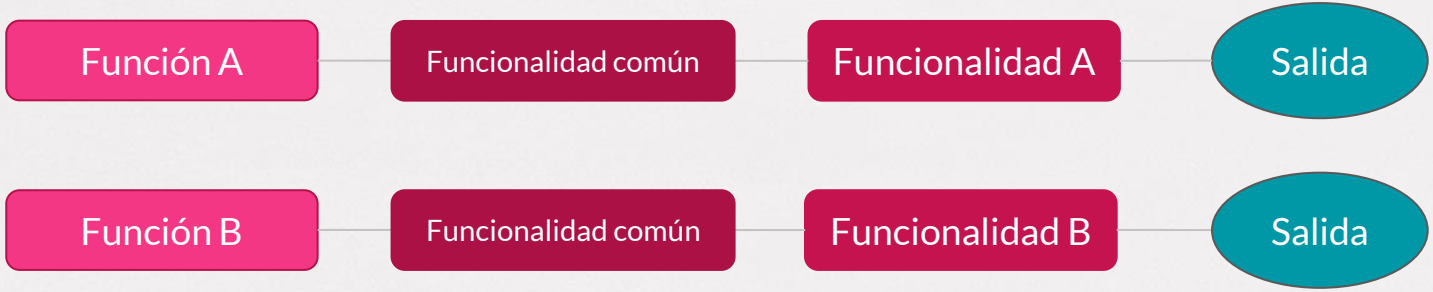
Abstracción de funcionalidades

Componentes de alto nivel o HOC

OpenWebinars

ÍNDICE

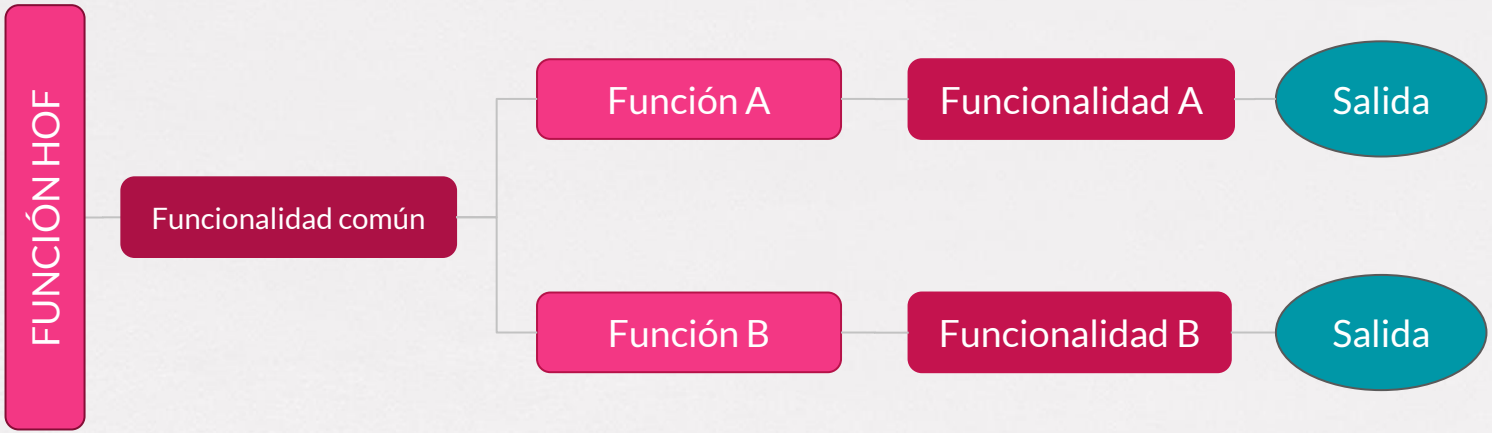
Origen de los HOC



- Los componentes de alto orden (HOC) **están basados en las funciones de alto orden**, concepto de la programación funcional

- Las **HOF** son **funciones que devuelven funciones**, generalmente, para lidiar con funcionalidades comunes

Origen de los HOC

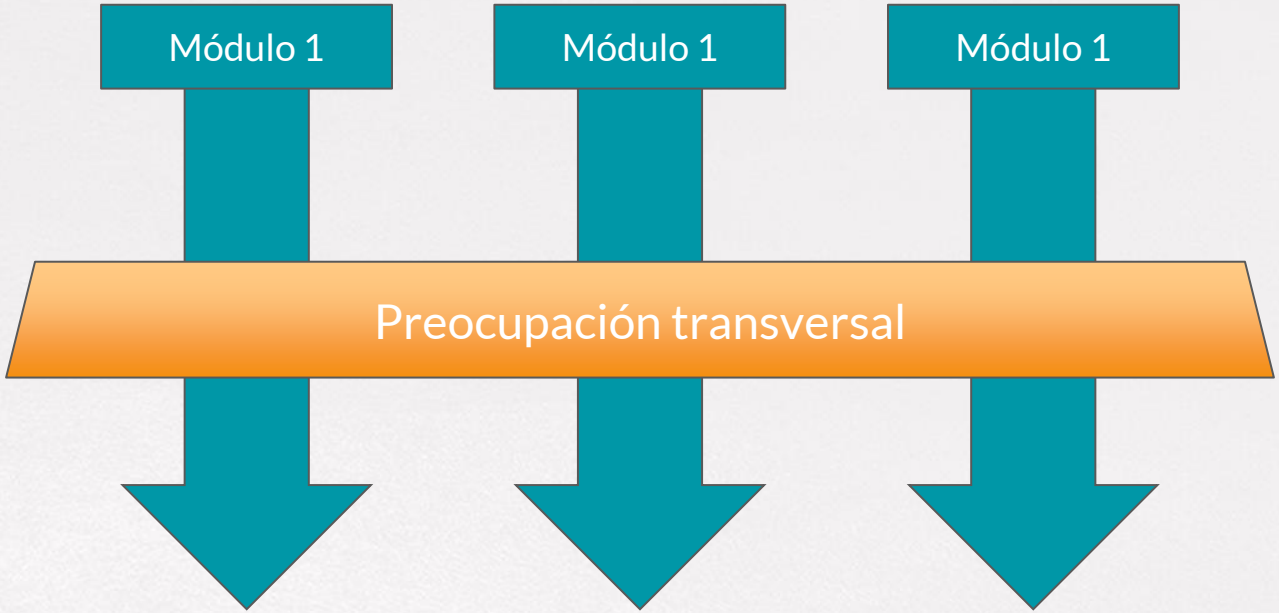


- Ejemplos habituales de funciones HOF son aquellas que realizan **debounce** o **throttling** sobre otra función
- Por tanto un **HOC** es, igualmente, un componente que encapsula componentes, devolviendo un nuevo componente como una función pura

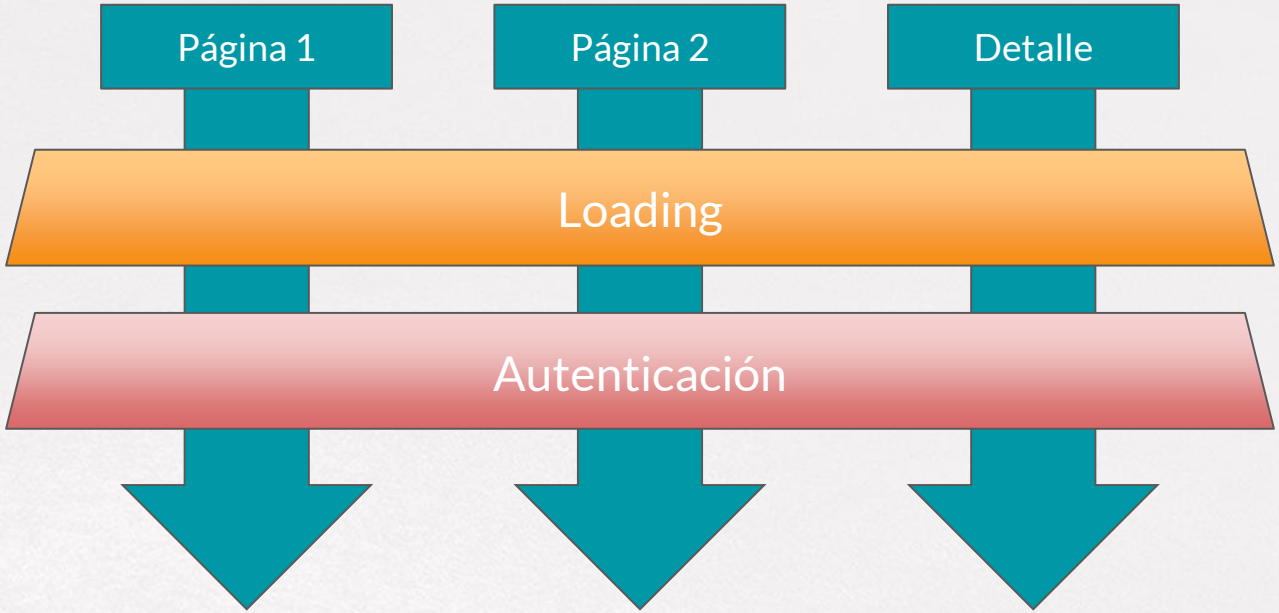
El HOC para preocupaciones transversales

- En componentes, estas funcionalidades comunes que pueden afectar a otras partes de la aplicación las denominamos **preocupaciones transversales** (o **crosscutting concerns**)
- Las preocupaciones transversales son la base de **los aspectos** en la **programación orientada a aspectos**
- Inicialmente se solucionaban popularmente en la comunidad con el uso de **mixins**
- Los propios desarrolladores de React **aconsejaron la composición de componentes** usando **HOCs**

El HOC para preocupaciones transversales



El HOC para preocupaciones transversales



Uso de los HOC

```
const EjemploHOC = InputComponent => {
    return class extends React.Component{
        render() {
            return <InputComponent {...this.props}/>;
        }
    }
}
```

```
const NuevoComponente = EjemploHOC(Componente);

const App = () =>
  <div>
    <NuevoComponente name="un nombre" />
  </div>;

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

- El ejemplo más sencillo es **un componente** que devuelve **otro componente anónimo**
- Las propiedades son pasadas al nuevo componente por **destructuring**
- Siempre que **devolvamos un componente** podemos implementarlo como queramos

Uso de los HOC

```
const withName = InputComponent => {
    return class extends React.Component{
        render() {
            return <InputComponent
                        name="Un nombre"
                        {...this.props}/>;
        }
    }
}
```

```
const NuevoComponente = withName(Componente);
const App = () =>
  <div>
    <NuevoComponente/>
  </div>;

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

- Un HOC puede servir para **introducir propiedades comunes**
- También es posible usar HOCs para **uso de estados compartidos**
- Casos frecuentes de uso de HOCs son funcionalidades como **loaders**, **autenticaciones**, **patrón adaptador...**

Convenciones y consideraciones

```
const logProps = InputComponent => {
    InputComponent.prototype.componentDidUpdate = function(prevProps) {
        console.log('Props actuales: ', this.props);
        console.log('Props anteriores: ', prevProps);
    };
    return InputComponent;
}
const EnhancedComponent = logProps(InputComponent);
```

No mutes el componente

Aparte de restar funcionalidad a los componentes de entrada, si aplicas dos HOC que muten **la misma característica**, se perderá **una de las funcionalidades**

Convenciones y consideraciones

```
const logProps = InputComponent => {
    return class extends React.Component {
        componentDidUpdate(prevProps) {
            console.log('Props actuales: ', this.props);
            console.log('Props anteriores: ', prevProps);
        }
        render() {
            return <InputComponent {...this.props} />;
        }
    }
}
```

No mutes el componente

Usando la estrategia de **composición** podemos encadenar HOCs de manera trazable y **acumular todas las lógicas agregadas**

Convenciones y consideraciones

```
const withName = InputComponent => {
    return class extends React.Component{
        render() {
            return <InputComponent
                        name="Un nombre"
                        {...this.props}/>;
        }
    }
}
```

Pasa de nuevo los props

Como los HOC solucionan **preocupaciones transversales** no deberían de afectar al funcionamiento del componente, **pasa siempre por tanto las props de nuevo**

Convenciones y consideraciones

```
const withName = InputComponent => {
    class WithName extends React.Component{
       render() {
           return <InputComponent name="Un nombre" {...this.props}/>;
       }
   }
   WithName.displayName = `WithName(${getDisplayName(InputComponent.displayName ||
InputComponent.name || 'Component')})`;
   return WithName;
}
```

Envuelve el nombre

Envolviendo el nombre del componente a mostrar facilitamos la depuración, ya que estos **se muestran en las herramientas de React**

PARA RESUMIR

- Un **HOC** está basado en el concepto de **función de alto nivel de Javascript** (HOF) y consiste en un **componente que devuelve otro componente**

- Los HOC se dedican principalmente a s**olucionar preocupaciones transversales**, esto es, características comunes que afectan a **varias partes de la aplicación**

- Para facilitar el desarrollo y la trazabilidad es recomendable seguir **unas convenciones respecto al uso de HOCs**