React

Propiedades (props) y estado (state) de los componentes

Enrique Barra Arias
Álvaro Alonso González

Componentes de React

- Conceptualmente, los componentes son elementos que aceptan argumentos de entrada arbitrarios (llamados "props") y devuelven elementos de React que describen lo que debería aparecer en la pantalla

```
<Welcome name="Pepe"/>
```

- La forma más simple de definir un componente es escribir una función JavaScript que devuelva lo que debe ser renderizado

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

```
const Welcome = (props) => {
  return <h1>Hello, {props.name}</h1>;
}
```

- También puedes usar una clase ES6 para definir un componente
  - Recuerda que la clase React Component debe implementar el método render

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

Props

- Propiedades de cada instancia de un componente
- Los componentes pueden ser personalizados con diferentes parámetros cuando se crean. Estos parámetros de creación se llaman **props**.
- Las props son datos en Javascript que se proveen como atributos al componente a la hora de renderizarse
  - Puede ser de cualquier tipo, string, number, boolean, object, …
- Las props son de solo lectura: **INMUTABLES**
- Un componente **nunca debe modificar sus propias props**

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

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

PropTypes

- https://reactjs.org/docs/typechecking-with-proptypes.html
- Las PropTypes nos ayudan a saber el nombre y el tipo de props que un componente espera
- Para definir los PropTypes de un componente necesitamos importar el objeto PropTypes (hay que instalar el paquete npm https://www.npmjs.com/package/prop-types)

```
import PropTypes from 'prop-types';
```

- Después de la definición del componente añadimos el nombre y los tipos de las props para facilitar el desarrollo y evitar mirar a los componentes padres todo el tiempo.

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

Greeting.propTypes = {
  name: PropTypes.string
};
```

DefaultProps

- Se pueden definir valores por defecto para las props, que entonces tomarán dicho valor si no nos la pasa el padre

- PropTypes se aplica después que defaultProps, así que deben tener el tipo y forma correctos

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

Greeting.defaultProps = {
  name: “Enrique”
};
```

State

- Propiedades de cada instancia de un componente
- El **state** o estado contiene información adicional sobre el componente que no lo provee quien lo renderiza, sino que lo crea y actualiza el propio componente
- El **state** son datos en Javascript
- Es útil cuando el componente tiene que mantener cierta información entre renderizados (pintados)
- El **state** es *mutable*, pero no puede modificarse directamente sino que debe modificarse de un modo especial, usando un Hook o gancho useState que nos dará una función modificadora de dicho estado (lo veremos detenidamente)

State

- La característica principal de state es que el componente **se repinta cada vez que su estado cambia**
- Esa es la principal diferencia con una variable creada con let o const
- Es lo que tiene que “recordar” el componente entre repintados, lo que yo necesitaría para decirle a alguien cómo repintar el componente, lo que guardaría en una base de datos para ser capaz de volver a repintar el componente, …
- Importante empezar a ver que el estado reside en el componente superior en el que se pueda. Es decir cuanto más arriba mejor en el árbol de components. El padre posteriormente pintará a sus hijos pasándoles props

Props vs State

- Entonces elegiremos *state* cuando algo es mutable y *props* cuando es inmutable

- Componentes gordos (listos) y flacos (tontos).
  - Los gordos tienen estado, que es el estado de la aplicación o de esa parte de la aplicación, tendrán también funciones para mutar el estado, para recibirlo con una llamada Ajax, para guardarlo, para ir un paso atrás (deshacer) o adelante (rehacer), etc.
  - Los flacos solo tendrán props y no tendrán casi funcionalidad, solo saben pintar lo que les pasan, si acaso con algún condicional o de determinado estilo dependiendo de las props, pero sin funcionalidad, si les pasa algo (un evento, un click por ejemplo) llamarán al componente padre que recalcule el nuevo estado con lo que ha ocurrido y les vuelva a llamar con otras props.

- Un par de enlaces con información adicional:
  - https://lucybain.com/blog/2016/react-state-vs-pros/
  - https://github.com/uberVU/react-guide/blob/master/props-vs-state.md

Composición de componentes

- Los componentes pueden hacer uso de otros componentes
- Esto nos permite usar la misma abstracción de componentes para cualquier nivel de detalle
- La herencia no se recomienda en React. La composición funciona mejor
- Más información
  https://reactjs.org/docs/composition-vs-inheritance.html

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

function App(props) {
  return (
    <div>
      <h1>Hello, this is the title!</h1>
      <h2>{props.subtitle}</h2>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

ReactDOM.render(
  <App subtitle="second line"/>,
  document.getElementById('root')
);
```

Composición de componentes

- En general es muy útil dividir un componente en otros más pequeños
- Es equivalente a refactorizar el código
- Permite que los componentes sean más reutilizables y aislar la complejidad

- Ver ejemplos en:
- https://github.com/REACT-UPM/ejemplos_simples/tree/main/src/composicion

```
export default function Comment(props) {

  function formatDate(date) {
    return date.toLocaleDateString();
  }

  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar"
             src={props.author.avatarUrl}
             alt={props.author.name} />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

```
export default function Comment(props) {

  function formatDate(date) {
    return date.toLocaleDateString();
  }

    return (
    <div className="Comment">
      <UserInfo user={props.author} />
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

PropTypes y defaultProps ejemplo completo

- Ver https://github.com/REACT-UPM/ejemplos_simples/blob/main/src/composicion/CommentWithPropTypes.js

```
Comment.propTypes = {
  author: PropTypes.object,
  text: PropTypes.string,
  date: PropTypes.instanceOf(Date)
};


Comment.defaultProps = {
  author: {
    name: "Anónimo",
    avatarUrl: "https://cdn-food.tribune.com.pk/users/user.png"}
};
```