

# React

Componentes, Hooks...



# Librería vs Framework

React es una librería, Angular es un Framework.

## Librería

Tu código usa una librería

## Framework

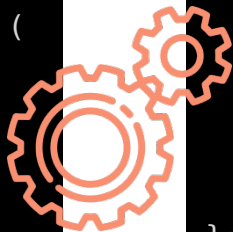
Tu código es framework

# JSX

No es HTML, No es un string... es azúcar en XML que se traduce a JavaScript

```
import React from "react";

export const HelloComponent = () => (
  <div>
    <h2>Hello from React</h2>
    <h3>Lemoncode :)</h3>
  </div>
);
```



```
var HelloComponent = function HelloComponent() {
  return
    _react.default.createElement("div", null,
      _react.default.createElement("h2", null,
        "Hello from React"),
      _react.default.createElement("h3", null,
        "Lemoncode :)")
    );
};
```

# Componentes

Los componentes en React son funciones (\* => recordamos class component legacy).

```
export const HelloComponent = (props) => {  
  return <h2>Hello {props.name}</h2>  
}
```

Son funciones que se ejecuten y mueren una y otra vez en cada render

Devuelven siempre React Elements

Me permiten dividir el UI en piezas independientes y reusables

Los componentes son piezas aisladas

Puedo hacer composición de componentes

Exponen un contrato (props) para interactuar

# Propiedades

Las propiedades definen un contrato que permite que otros componentes puedan consumir un componente que hayamos creado e interactuar con él.

```
interface Props {  
  name: string;  
}  
  
export const HelloComponent = (props: Props) => {  
  return <h2>Hello {props.name}</h2>;  
};
```

Se alimentan de padre a hijo

Son de sólo lectura

La definición de propiedades se lleva muy bien con TypeScript

¿ Cómo informo de hijo a padre? Pasando propiedades de tipo función

# Propiedades - callback

Informando de hijo a padre

```
interface Props {  
  name: string;  
  onChangeName: (newname: string) => void;  
}  
  
export const NameEdit = (props: Props) => {  
  return (  
    <input  
      value={props.name}  
      onChange={e => props.onChangeName(e.target.value)}  
    />  
  );  
};
```

Tengo como propiedad un callback (función)

En cada cambio del input invoco dicha función

La función se ejecuta en el componente padre

El componente padre tiene control de cuando se ejecuta

# Estado

Nos permite almacenar información de lectura y escritura.

```
export const MyComponent = () => {  
  const [name, setName] = React.useState('John');  
  
  return <input value={name} onChange={e => setName(e.target.value)} />;  
};
```

useState me permite persistir datos más allá de la muerte de la función

La asignación de valor es asíncrona

useState me permite almacenar tanto tipos simples como objetos

Si manejo estructuras inmutable en el estado puede hacer optimizaciones

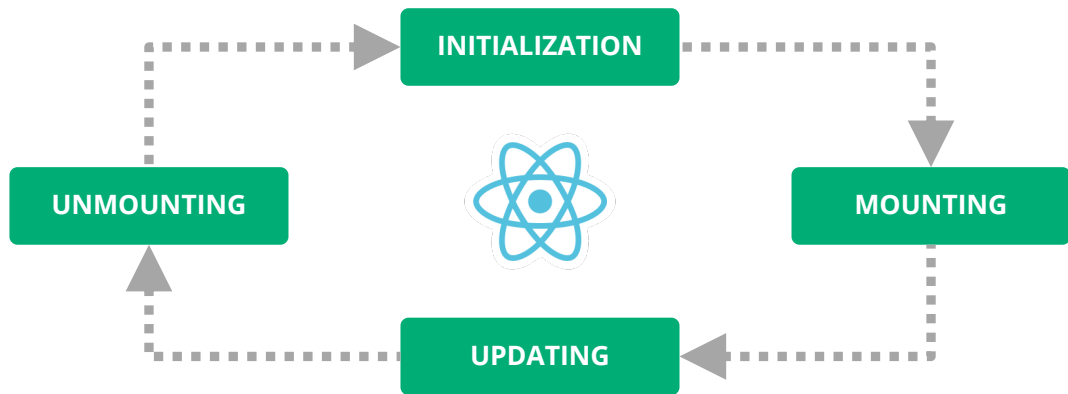
useState se basa en la tecnología de Hooks

Antes de usar Hooks hay que saber bien como funcionan

# Ciclo de vida de un componente

Todo en este mundo sigue un ciclo, por ejemplo humanos o plantas. Nacemos, crecemos y morimos. React no iba a ser menos...

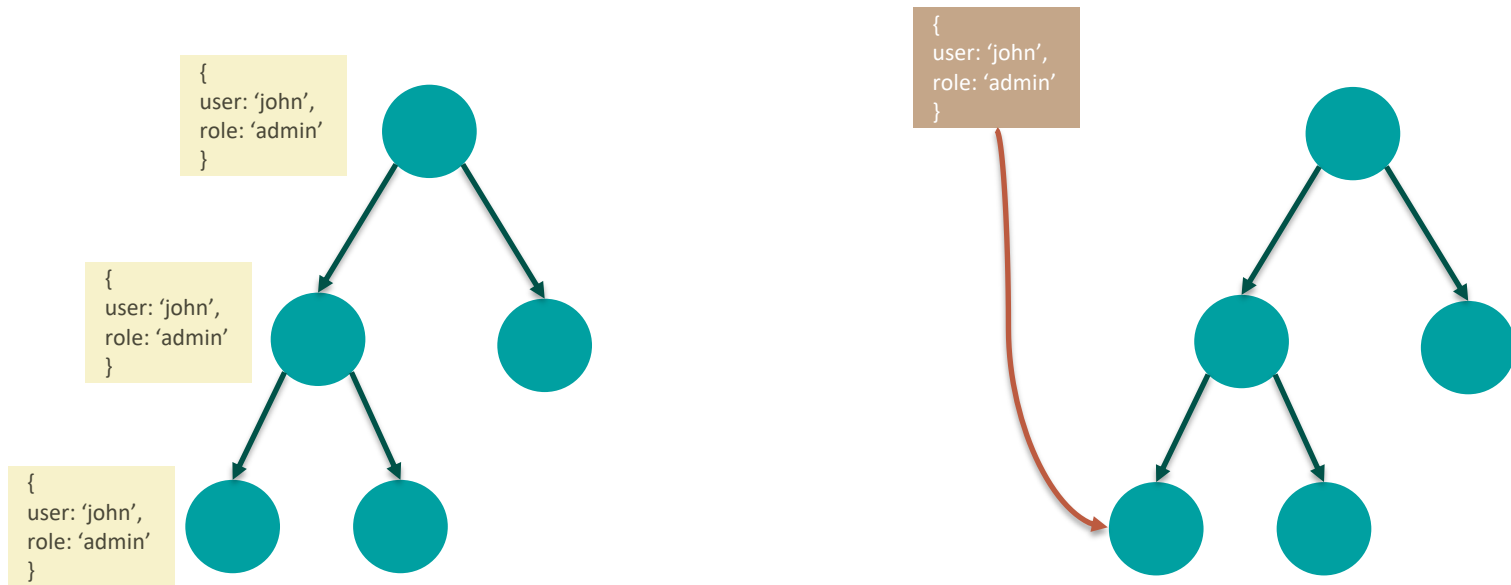
Los **componentes** se crean (se montan en el DOM), se actualizan (“crecen”), y mueren (se liberan del DOM). A esto lo llamamos **ciclo de vida de un componente**.





# Manejo de datos comunes

Cuando desarrollamos una aplicación podemos encontrarnos con datos que son transversales a la misma, e ir pasándolos de padre a hijo sucesivamente puede impactar en la calidad y mantenibilidad de nuestro código.



# ¿ Por qué no un singleton?

¿ Nos podríamos simplificar y usar una variable estática global o un singleton para almacenar estos datos comunes?

Si hacemos un cambio en el contexto automáticamente se propaga, con un singleton tendríamos que hacer nuestro propio event emitter

El contexto lo podemos colocar a al altura del árbol de componetes que veamos oportuna

Con un singleton / variable estática tendríamos problemas si hacemos Server Side Rendering.

# Hooks

React Hooks son funciones que nos permiten enganchar estado y ciclo de vida a componentes funcionales

*"Dotan de super poderes a un componente funcional"*

STATE

LIFECYCLE LIKE

*"Los components Funcionales pasan a ser ciudadanos de primera categoría"*

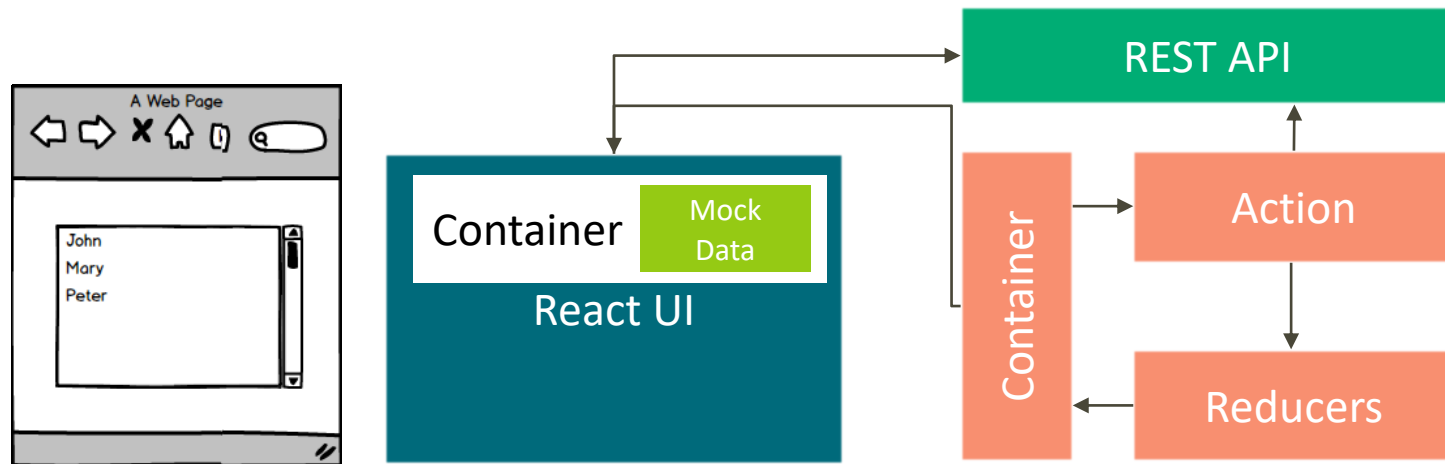
# Componentización

Si rompemos un componente grande en subcomponentes, tenemos código más legible (principio de mismo nivel de abstracción) y potencialmente más reusable.

```
<Card className={classes.card} key={hotel.id}>
  <CardHeader
    avatar={
      <Avatar aria-label="Recipe" className={classes.avatar}>
        {hotel.hotelRating}
      </Avatar>
    }
    action={
      <IconButton>
        <MoreVertIcon />
      </IconButton>
    }
    title={hotel.name}
    subheader={hotel.address1}
  />
  <CardContent>
    <div style=
      {{
        display: 'flex',
        flexDirection: 'column',
        justifyContent: 'center'
      }}>
      <CardMedia
        className={classes.media}
        image={hotel.thumbNailUrl}
        title={hotel.name}
      />
      <Typography variant="subtitle1" gutterBottom>
        {hotel.shortDescription}
      </Typography>
    </div>
    <CardActions className={classes.actions} disableActionSpacing>
      <IconButton aria-label="Add to favorites">
        <FavoriteIcon />
      </IconButton>
    </CardActions>
  </CardContent>
</Card>
```

```
<Card
  className={classes.card}
  key={props.hotel.id}>
  <HotelCardHeader
    name={props.hotel.name}
    address={props.hotel.address1}
    rating={props.hotel.hotelRating}
  />
  <HotelCardContent
    pictureURL={props.hotel.thumbNailUrl}
    name={props.hotel.name}
    description={props.hotel.shortDescription}
  />
  <HotelCardActions/>
</Card>
```

# Desarrollo progresivo



- ✓ Plantea un layout y componentiza.
- ✓ Añade al contenedor datos mock.
- ✓ Sustituye por API real
- ✓ ¿ Está justificado usar Redux? Adelante...

# Creando el proyecto

¿Cómo arranco mi proyecto desde cero?

## **cli**


El equipo de Facebook ofrece  
create-react-app

## **Desde cero**

Con webpack o parcel puedes  
configurarlo adaptado a tus  
necesidades

# ¡ Muchas gracias !



 @lemoncoders



 @basefactorteam



<https://github.com/lemoncode>