Consumir servicios usando React

Jefferson A. Peña Torres
jefferson.amado.pena@correounivalle.edu.co

Universidad Pontificia Bolivariana

- Carga asíncronas de datos
- Prevención de errores
- Carga *Suspense* y perezosa (*lazy*).
- División de código
- Consumiendo APIs (Inicio)

Consumir servicios usando React

Repasemos, formularios y componentes

1. Carga asíncronas de datos
2. Prevención de errores
3. Carga *Suspense* y perezosa
4. División de código
5. Consumiendo APIs

En React o en cualquier aplicación web es importante tener en cuenta los eventos, el DOM posee unos y de manera similar este *framework* provee métodos que capturan eventos sintéticos de acuerdo a las especificaciones W3C.

En cuanto a la construcción de componentes hay dos opciones controlados y no controlados. Los primeros permiten la sincronización entre la entrada del usuario y el estado del componente. Mientras que con la segunda opción, componentes no controlados, es responsabilidad de la aplicación mantener la información.

Consumir servicios usando React

Carga asíncrona de datos

Javascript en corto tiempo involucro llamadas asíncronas y callbacks a promesas en (ES2015) y después en (ES2017). La característica de realizar tareas u operaciones mientras se carga o se obtiene información hace parte de las aplicaciones web de hoy.

Consumir servicios usando React

Carga asíncrona de datos

Javascript en corto tiempo involucro llamadas asíncronas y callbacks a promesas en (ES2015) y después en (ES2017). La característica de realizar tareas u operaciones mientras se carga o se obtiene información hace parte de las aplicaciones web de hoy.

```
const hacerAlgoAsync = () => {
  return new Promise(resolve => {
    setTimeout(() =>
      resolve("Aquí estoy!"), 3000)
  })
}
const hacerAlgo = async () => {
  console.log(await hacerAlgoAsync())
}

console.resolve("Antes");
hacerAlgo();
console.resolve("Después");
```

Consumir servicios usando React

Carga asíncrona de datos

Javascript en corto tiempo involucro llamadas asíncronas y callbacks a promesas en (ES2015) y después en (ES2017). La característica de realizar tareas u operaciones mientras se carga o se obtiene información hace parte de las aplicaciones web de hoy.

Las promesas son clave, una función que retorna una promesa opera de forma asíncrona.

```
const hacerAlgoAsync = () => {
  return new Promise(resolve => {
    setTimeout(() =>
      resolve("Aquí estoy!"), 3000)
  })
}
const hacerAlgo = async () => {
  console.log(await hacerAlgoAsync())
}

console.resolve("Antes");
hacerAlgo();
console.resolve("Despues");
```

Consumir servicios usando React

Carga asíncrona de datos

Javascript en corto tiempo involucro llamadas asíncronas y callbacks a promesas en (ES2015) y después en (ES2017). La característica de realizar tareas u operaciones mientras se carga o se obtiene información hace parte de las aplicaciones web de hoy.

Las promesas son clave, una función que retorna una promesa opera de forma asíncrona.

- async
- await

```
const hacerAlgoAsync = () => {
  return new Promise(resolve => {
    setTimeout(() =>
      resolve("Aquí estoy!"), 3000)
  })
}
const hacerAlgo = async () => {
  console.log(await hacerAlgoAsync())
}

console.resolve("Antes");
hacerAlgo();
console.resolve("Despues");
```

Consumir servicios usando React

Carga asíncrona de datos

Javascript en corto tiempo involucro llamadas asíncronas y callbacks a promesas en (ES2015) y después en (ES2017). La característica de realizar tareas u operaciones mientras se carga o se obtiene información hace parte de las aplicaciones web de hoy.

Las promesas son clave, una función que retorna una promesa opera de forma asíncrona.

- async
- await
- then
- done

```
const hacerAlgoAsync = () => {
  return new Promise(resolve => {
    setTimeout(() =>
      resolve("Aquí estoy!"), 3000)
  })
}
const hacerAlgo = async () => {
  console.log(await hacerAlgoAsync())
}

console.resolve("Antes");
hacerAlgo();
console.resolve("Despues");
```

Consumir servicios usando React

Carga asíncrona de datos

Javascript en corto tiempo involucro llamadas asíncronas y callbacks a promesas en (ES2015) y después en (ES2017). La característica de realizar tareas u operaciones mientras se carga o se obtiene información hace parte de las aplicaciones web de hoy.

Las promesas son clave, una función que retorna una promesa opera de forma asíncrona.

```
const getStudentData = () => {
  return fetch('/users.json') // get users list
    .then(response => response.json()) // parse
    .then(users => users[0]) // pick first user
    .then(user => fetch(`/users/${user.name}`))
    .then(userResponse => userResponse.json())
}

getStudentData()
```

Evitando errores

Situados en acciones, operaciones y carga asíncronas, la responsabilidad de hacer las cosas en el tiempo que se debe y no después es algo que se debe considerar. **Por ejemplo actualizar los datos de un componente que ha sido desmontado.**

Si consideramos hook como el *useEffect* para realizar algo después que se cumpla la promesa de obtener información desde una URI. Se pueden evitar errores utilizando banderas (flags).

Evitando errores

Situados en acciones, operaciones y carga asíncronas, la responsabilidad de hacer las cosas en el tiempo que se debe y no después es algo que se debe considerar. Por ejemplo actualizar los datos de un componente que ha sido desmontado.

Si consideramos hook como el *useEffect* para realizar algo después que se cumpla la promesa de obtener información desde una URI. Se pueden evitar errores utilizando banderas (flags).

```
. . .
useEffect(() => {
  let mounted = true;
  getStudentInformation(name)
    .then(data => {
       if(mounted) {
        showInformation(data)
    }
  });
}, [name])


...
```

Consumir servicios usando React

Carga perezosa Suspense y lazy

En la medida que las aplicaciones crecen existen funcionalidades que no se utilizan si el usuario no las requiere y no son necesarias en el cliente. El código de la aplicación puede ser dividida en *chunks*. React utiliza *Suspense* y *Lazy* con sistemas de construcción de aplicaciones que pueden dividir en partes más pequeñas para ser enviadas bajo demanda.

Carga perezosa Suspense y lazy

En la medida que las aplicaciones crecen existen funcionalidades que no se utilizan si el usuario no las requiere y no son necesarias en el cliente. El código de la aplicación puede ser dividida en *chunks*. React utiliza *Suspense* y *Lazy* con sistemas de construcción de aplicaciones que pueden dividir en partes más pequeñas para ser enviadas bajo demanda.

Los *imports* de un componente pueden ser cargados en la medida que se necesiten.

```
. . .

const ProcInformation = lazy(() =>
import('./src/ProcInformation'))

. . .
```

Carga perezosa Suspense y lazy

En la medida que las aplicaciones crecen existen funcionalidades que no se utilizan si el usuario no las requiere y no son necesarias en el cliente. El código de la aplicación puede ser dividida en *chunks*. React utiliza *Suspense* y *Lazy* con sistemas de construcción de aplicaciones que pueden dividir en partes más pequeñas para ser enviadas bajo demanda.

Los *imports* de un componente pueden ser cargados en la medida que se necesiten. Se necesitan cuándo los componentes de la vista o la aplicación son requeridos.

```
. . .

const ProcInformation = lazy(() =>
import('./src/ProcInformation'))


. . .

function App() {

. . .
<Suspense fallback={<div>Loading Component</div>}>
    {show && <ProcInformation name={. . .}>}
<Suspense>
. . .

}
```

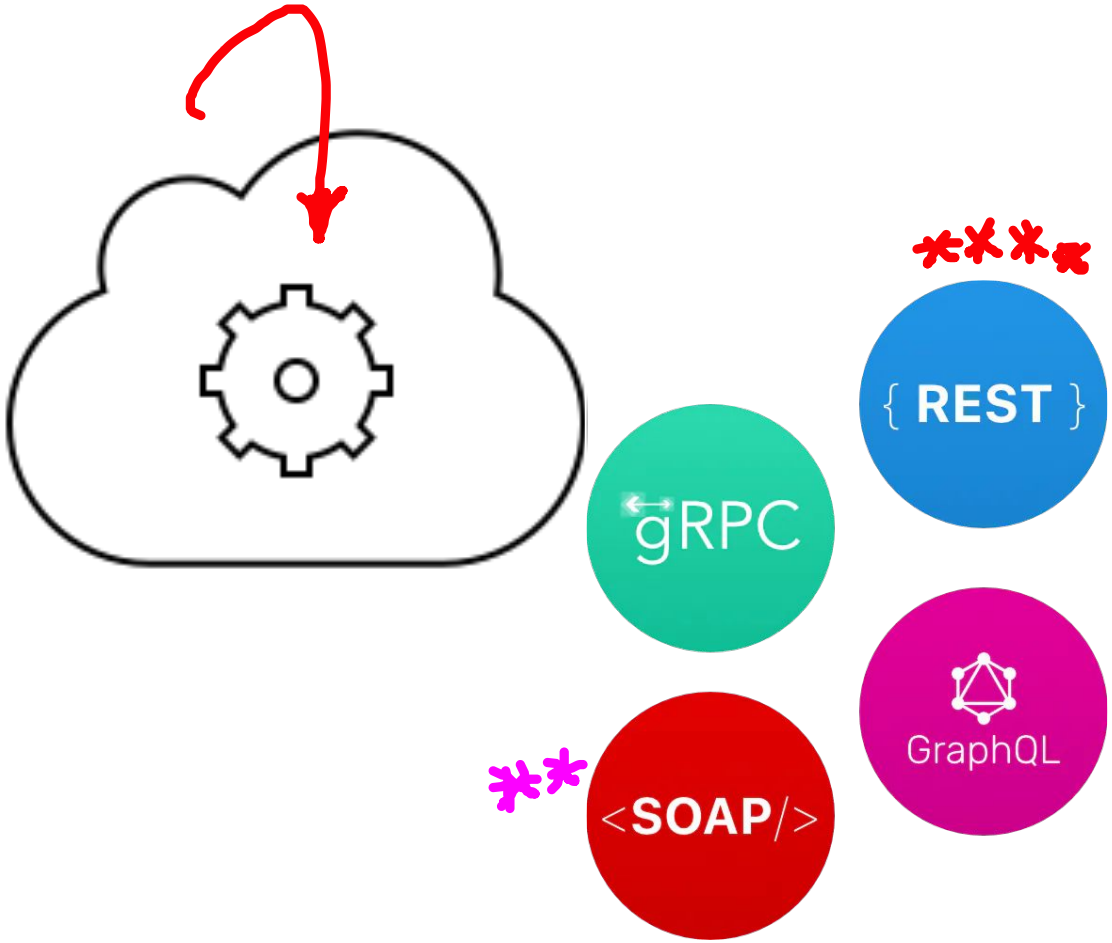
División de código

Es una de las características más convenientes de algunos de los constructores de aplicaciones web, como webpack. Esta permite que dividir el código en varios *bundles* que son cargados bajo demanda o en paralelo. Esto sirve para controlar recursos del lado del cliente o hacer una carga con prioridades.

Consumir servicios usando React

Consumiendo APIs

Los WEB APIs son servicios que ofrecen recursos a través de internet. De estas existen varias arquitecturas que se soportan en el intercambio de información en formato HTML, XML o JSON. Son de dominio público, privado o hechas a la medida para responder a aplicaciones Frontend.



Consumir servicios usando React

Resumen

1.Carga asíncronas de datos

2.Prevención de errores

3.Carga *Suspense* y perezosa

4.División de código

5.Consumiendo APIs

Realizar tareas u operaciones mientras se carga o se obtiene información desde otra fuente de manera paralela o asíncrona es una de las ventajas que provee Javascript en sus versiones ES6 y ES7. React utiliza métodos y operaciones heredadas que permiten realizar de manera asíncronas.

Aunque lo asíncrono suena bien, existen errores que están en manos del programador como la actualización o carga de datos en un componente que ha sido desmontado.

Con Suspense y Lazy el código de una aplicación puede ser dividido en partes que se ejecuten bajo demanda o cuándo el usuario los solicite. Esta división de código se logra con algunas herramientas de empaquetado como Webpack.

Los WEB APIs son servicios que ofrecen recursos a través de internet. De estas existen varias arquitecturas que se soportan en el intercambio de información en formato HTML, XML o JSON.

www.upb.edu.co/es/mision-tic

#MisiónTICSomosTodos