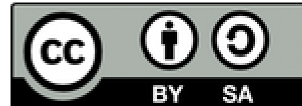
Introduction aux applications clientes avec React

Konstantinos Lambrou-Latreille



React

- Bibliothèque Javascript qui s'exécute côté client
  - … mais peut-être utilisé sur le serveur
- Approche déclarative
  - on indique QUOI afficher et non pas COMMENT l'afficher
- Fonctionnel
  - une composante est une fonction entre un état et un rendu.

Composante simple

macomposante.js

```
export default props => {
  // Autres traitements à faire ici ...

  // Obligatoire à avoir une balise englobante (div)
  return (
    <section>
      <h1>Mon article</h1>
      <p>Un paragraphe de mon article</p>
    </section>
  )
}
```

Plus simple :

```
export default _ => (
  <section>
    <h1>Mon titre</h1>
    <p>Un paragraphe de mon article</p>
  </section>
)
```

Composante simple

Peut être réutilisé dans une autre composante

```
import MaComposante from './macomposante.js'

const ComposanteParente = _ => (
  <div>
    <h1>Titre de ma page</h1>
    <MaComposante />
  </div>
)
```

Le HTML résultant est:

```
<div>
  <h1>Titre de ma page</h1>
  <section>
    <h1>Mon titre</h1>
    <p>Un paragraphe de mon article</p>
  </section>
</div>
```

JSX (Javascript Syntax Extension)

- Mélange de Javascript et HTML
- L'outil Babel est utilisé pour convertir JSX en Javascript.
- Les composantes peuvent être implémentées en Javascript ou en JSX

JSX (Javascript Syntax Extension)

Version JSX

```
export default props => {
  return (
    <section className="post">
      <h1>Mon article</h1>
      <p>Un paragraphe de mon article</p>
    </section>
  )
}
```

JSX (Javascript Syntax Extension)

Version Javascript

```
export default props => {
  const h1Element = React.createElement('h1', null, 'Mon article');

  const pElement = React.createElement('p', null, 'Un paragraphe de mon article')

  const sectionElement = React.createElement("section", class: 'post', [
    h1Element,
    pElement
  ]);

  return sectionElement;
}
```

Types de composantes

- React est basé sur la notion de composante.
- Deux types de composantes :
    1. Composantes basées sur des classes
    2. Composantes fonctionnels
- Dans ce cours, nous allons nous concentrer sur les composantes fonctionnelles
- Les exemples précédents utilisaient les composantes fonctionnelles

Types de composantes

Voici un exemple de composante de classe :

```
class MaComposante extends React.Component {
  render() {
    return (
      <section className="post">
        <h1>Mon article</h1>
        <p>Un paragraphe de mon article</p>
      </section>
    )
  }
}
```

Gabarit *create-react-app*

- Dans ce cours, nous utiliserons le gabarit *create-react-app* de Facebook pour nos applications React.
- Viens avec plusieurs configurations préétablies pour nous permettre de commencer à coder.

Pour créer un nouveau gabarit:

```
npx create-react-app nomDuGabarit
cd nomDuGabarit
npm start
```

Interpolation

- Toute variable qui est dans la portée d'une composante peut être utilisé dans la vue
- L'interpolation se fait avec la syntaxe *{expression}*
- L'expression est une instruction Javascript

```
const composante = _ => {
  const blog = {
    title: 'Mon titre',
    paragraphe: 'Mon paragraphe'
  }

  // Il y a une erreur dans le code suivant.
  // Pouvez-vous la trouver ?
  return (
    <article>
      <h1>{blog.title}</h1>
      <p>{blog.paragraphe}</p>
      <p>{'***' + 'Un autre paragraphe' + '***'}</p>
      <p>{Un dernier paragraphe}</p>
    </article>
  )
}
```

Itérateurs

- Comme dans les gabarits utilisé sur le serveur, il est possible d'itérer sur un tableau pour afficher des éléments
- Contrairement à Pug, on utilise simplement du Javascript
- Pas de boucle *for*

```
const composante = _ => {
  const mesparagraphes = [
    'Mon premier paragraphe', 'Mon deuxième paragraphe', 'Mon dernier paragraphe'
  ]

  return (
    <div>
      {mesparagraphes.map(p => (
        <p>{p}</p>
      ))}
    </div>
  )
}
```

Condtionnels

- Comme dans les gabarits utilisé sur le serveur, il est possible d'utiliser les conditionels
- Contrairement à Pug, les conditions se font encore avec du Javascript
- Pas de *if* dans le HTML

```
const composante = _ => {
  const mesparagraphes = [
    'Mon premier paragraphe', 'Mon deuxième paragraphe', 'Mon dernier paragraphe'
  ]

  return (
    <div>
      {mesparagraphes.filter((_, i) => i > 0).map(p => <p>{p}</p>)}

      {mesparagraphes.map((p, i) => {
        if (i > 0) {
          return <p>{p}</p>
        } else {
          return <p></p>
        }
      })}
    </div>
  )
}
```

Trouver les erreurs

Cinq erreurs se sont glissées dans le code.

```
const composante = _ => {
  const title = 'Mon titre'
  const t = [ 1, 2, 3 ]

  return (
    {t.map(i => <p>titre</p>)}

    {t.map(i => {
      if (i > 0) {
        <p>{i}</p>
      }
    })}
  )
}
```

Propriétés d'une composante

Supposons qu'on voudrait passer des paramètres dans une composante.

```
<Article title="Mon titre"
         paragraphes={['Premier paragraphe', 'Deuxième paragraphe']} />
```

Le premier paramètre de la composante *props* est un objet nous donne cette information.

```
const Article = props => {
  return (
    <article>
      <h1>{props.title}</h1>
      {props.paragraphes.map(p => (
        <p>{p}</p>
      ))}
    </article>
  )
}
```

Une erreur peut se produire dans le code précédent. Avez-vous une idée ?

Propriétés d'une composante

Voici une correction possible :

```
const Article = props => {
  const paragraphes = props.paragraphes === undefined ? [] : props.paragraphes
  return (
    <article>
      <h1>{props.title}</h1>
      { paragraphes.map(p => <p>{p}</p>) }
    </article>
  )
}
```

Ou encore mieux avec ES6 :

```
const Article = ({ title, paragraphes = [] }) => {
  return (
    <article>
      <h1>{title}</h1>
      { paragraphes.map(p => <p>{p}</p>) }
    </article>
  )
}
```

État d'une composante

- Les propriétés permettent une communication entre les composantes (parent-enfant ou enfant-parent)
- Si le parent change la valeur injectée, la composante enfant se recharge.
- Par contre, si on veut que la composante se recharge si jamais un utilisateur effectue une action (un clique), comment peut-on faire ?
- Pourquoi ne peut-on pas utiliser *props* ?

État d'une composante

Par exemple, supposons qu'une composante affiche le nombre de fois qu'on a cliqué sur son bouton.

Voici une première version.

```
const Button = _ => {
  let n = 0

  const handleClick = _ => {
    n += 1
  }

  return (
    <div>
      <p>Vous avez cliqué sur le bouton {n} fois</p>
      <button onClick={handleClick}></button>
    </div>
  )
}
```

État d'une composante

À noter que pour des gestionnaires d'événements simples, on peut l'insérer directement dans le HTML

```
const Button = _ => {
  let n = 0

  return (
    <div>
      <p>Vous avez cliqué sur le bouton {n} fois</p>
      <button onClick={_ => n += 1}></button>
    </div>
  )
}
```

Si on clique sur le bouton, *n* serait toujours égal à 0. Pourquoi ?

État d'une composante

- Il faut créer un état avec le *hook* React *useState*.
- Les Hooks sont une nouveauté de React 16.8. Ils permettent de bénéficier d’un état local et d’autres fonctionnalités de React (*useEffect*) sans avoir à écrire de classes.

```
import React, { useState } from 'react'

export default _ => {
  const [ n, setN ] = useState(0)

  <p>Vous avez cliqué sur le bouton {n} fois</p>
  <button onClick={_ => setN(n + 1)}></button>
}
```

Liaison bidirectionnelle

- Si une valeur change dans la vue (dans un *input* par exemple), elle est immédiatement réflétée dans la composante
- Si la valeur est changé dans la composante, elle est immédiatement réflétée dans la vue
- Pour faire cette liaison, on utilise:
  - la fonction *useState*
  - les événements Javascript classiques

Liaison bidirectionnelle

```
export default _ => {

  const [ value, setValue ] = useState('')

  return (
    <div>
      <input onChange={e => setValue(e.target.value)} value={value} />
      <p>Vous avez entré: {value}</p>
      <button onClick={_ => setValue('')}>Effacer</button>
    </div>
  )
}
```

Liaison bidirectionnelle

Exemple: Jeu de traduction

Hook d'effet

- Le hook d'effet *useEffect* permet l'exécution d'effets de bord dans les composantes
- Des exemples d'effets de bord sont :
  - charger des données d'un serveur distant
  - s'abonner à quelque chose
  - modifier manuellement le DOM avec Javascript ou jQuery
- Le hook d'effet s'exécute :
  - après le rendu de la composante
  - à chaque fois que la composante est modifié

Hook d'effet

```
import React, { useState, useEffect } from 'react';

const Example = () => {
  const [n, setN] = useState(0)

  useEffect(() => {
    document.title = `Vous avez cliqué ${n} fois`
  })

  return (
    <div>
      <p>Vous avez cliqué {n} fois</p>
      <button onClick={() => setN(n + 1)}>Cliquez ici</button>
    </div>
  )
}
```

Hook d'effet

Dans le cas où on veut récupérer des données d'un serveur distant, on utilisera *fetch*.

```
const url = 'http://localhost:3000/api/users'

const fetchUser = async userId => {

  const options = { 'headers': { 'accept-language': 'fr' } }

  const response = await fetch(`${url}/${userId}`, options)

  const user = await response.json()

  return user
}
```

Hook d'effet

Par la suite, on voudrait afficher les feeds qui ont été récupérés du serveur

```
import React, { useState, useEffect } from 'react';

const url = 'http://localhost:3000/api/users'

const DisplayUser = { userId = 0 } => {
  const [name, setName] = useState('')

  useEffect(async () => {
    const user = await fetchUser(userId)
    setName(user.name)
  })

  return <div>{name}</div>
}
```

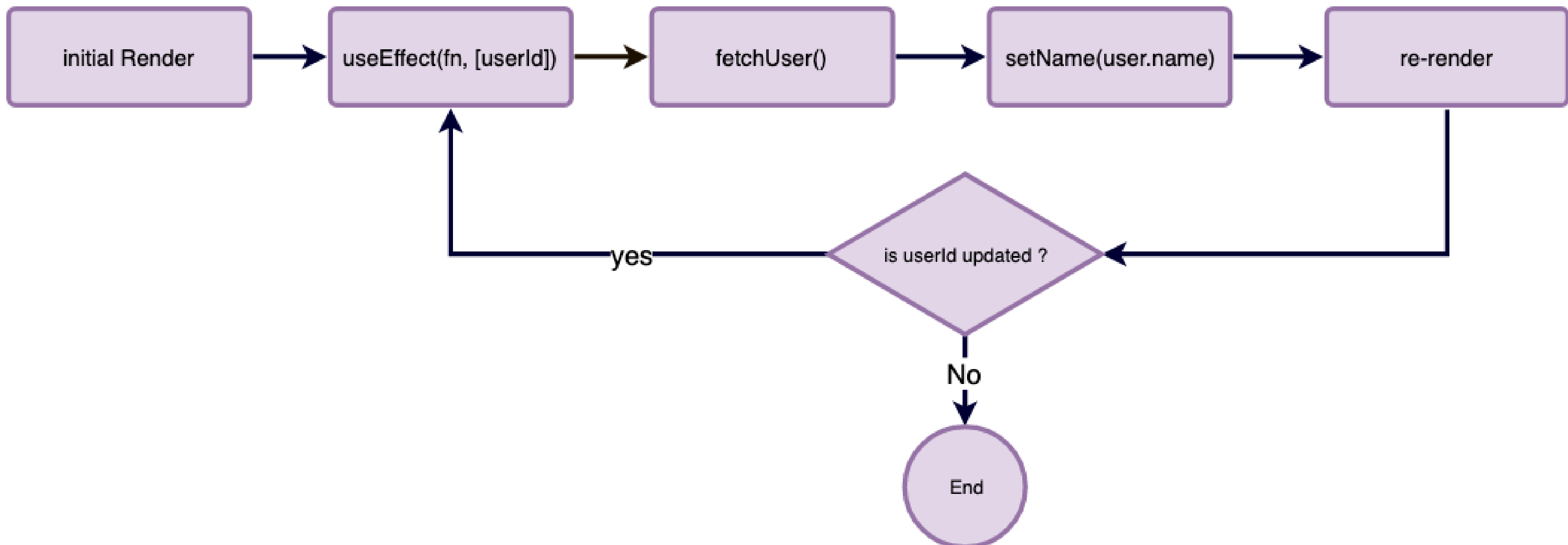
Si vous exécutez le code précédent, vous vous retrouvez avec une boucle infinie de rendu. Pourquoi ?

Hook d'effet

- Le hook d'effet va s'exécuter après le rendu initial et va appeller la fonction *fetchUser*
- La fonction du hook d'effet va ensuite modifier l'état de la composante avec *setName*.
- Un changement d'état recharge la composante, donc *useEffect* va s'exécuter à nouveau
- À l'infini...

Hook d'effet

- Pour prévenir cette boule infinie, nous allons utiliser le deuxième paramètre de *useEffect*
- Ce paramètre est un tableau de variables
- *useEffect* va se reexécuter uniquement si une de ces variables a été modifiée.



Hook d'effet

La solution :

```
import React, { useState, useEffect } from 'react';

const url = 'http://localhost:3000/api/users'

const DisplayUser = { userId = 0 } => {
  const [name, setName] = useState('')

  useEffect(async () => {
    const user = await fetchUser(userId)
    setName(user.name)
  }, [userId])

  return <div>{name}</div>
}
```

Hook d'effet

- Si on met un `console.log(...)` à l'intérieur de `useEffect`, combien de fois pensez-vous que la fonction de rappel sera appellée ?

Hook d'effet

- Si on met un `console.log(...)` à l'intérieur de `useEffect`, combien de fois pensez-vous que la fonction de rappel sera appellée ?
- Elle sera appellé 2 fois
- Pour que `useEffect` soit appellé une seule fois au chargement initiale de la composante, il faut mettre `[]` comme 2ieme paramètre de `useEffect`.

Routes

- Permet d'ajuster l'URL qui apparaît dans la barre de navigation
- Il faut installer le module avec: `npm install react-router-dom`
- Pour utiliser le module dans une composante :

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

// ...
```

Routes

- Ensuite, il faut définir nos routes dans la composante racine

```
export default function App() {
  return (
    <Router>
      <Header />
      <Switch>
        <Route exact path="/"> <Home /> </Route>
        <Route path="/about"> <About /> </Route>
        <Route path="/users/:id"> <User /> </Route>
        <Route path="/users" component={Users} />
      </Switch>
    </Router>
  )
}
```

La composante `User` a accès à `:id` au travers `props.match.params.id`.

Routes

Voici maintenant la composante *Header* qui contient le menu de navigation.

```
const Header = _ => (
  <div>
    <nav>
      <ul>
        <li>
          <Link activeClassName="active" to="/">Home</Link>
        </li>
        <li>
          <Link activeClassName="active" to="/about">About</Link>
        </li>
        <li>
          <Link activeClassName="active" to="/users">Users</Link>
        </li>
      </ul>
    </nav>
  </div>
)
```

Suite

Architecutre d'applications clientes réactives

- Patron SAM
- Libraire Redux inspiré de Flux