



# Créez une application React complète

12 hours



Medium

Last updated on 10/14/21



## Affichez les données d'une API dans un composant classe



Entre le moment où ils sont générés dans le DOM, et le moment où ils en sont retirés, les composants passent par différentes étapes. Dans les composants classe, vous avez une liste d'étapes qui correspondent à un moment précis du cycle de vie, dans lesquelles vous pouvez effectuer des actions.

On parle de *méthodes de cycle de vie*.

### Découvrez les méthodes de cycle de vie

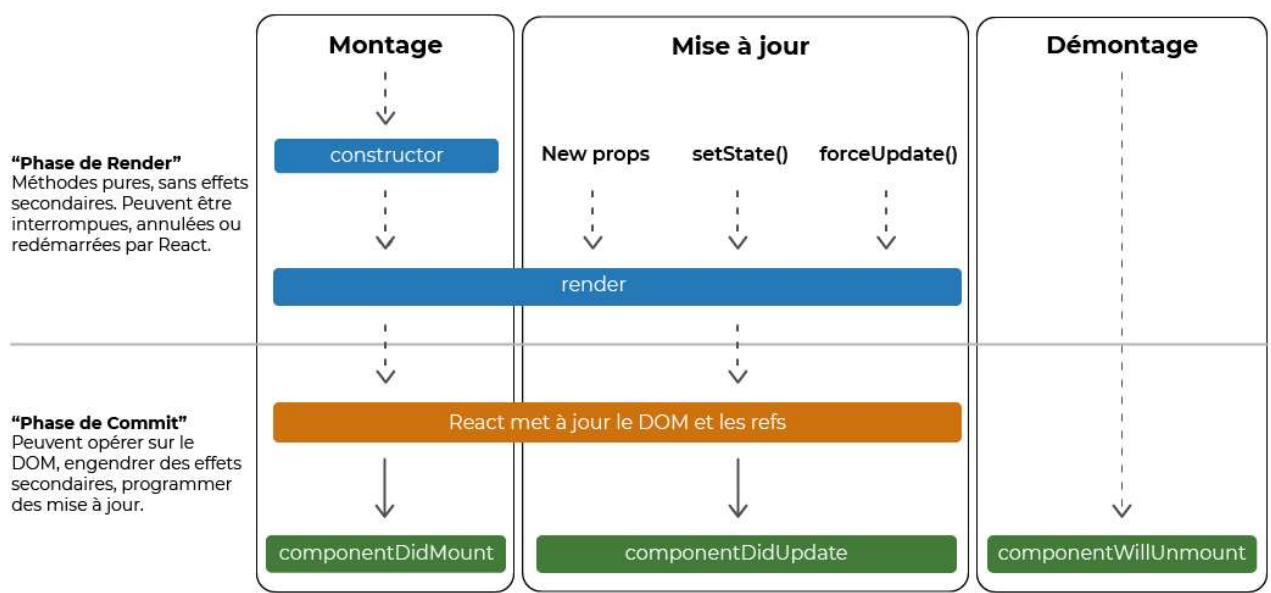


Dans le screencast juste en dessous, vous aurez une petite démonstration des différentes étapes que connaît un composant, et auxquelles vous pouvez accéder. 

Alors, est-ce que ça vous paraît plus clair ?

Dans les composants fonction que vous connaissez davantage, vous avez appris à déclencher une action juste après que le composant a été généré dans le DOM avec `useEffect` et en passant un tableau de dépendances vide.

Pour vous faire un petit résumé des différentes étapes auxquelles vous avez accès dans les composants classe :



Voilà un diagramme des méthodes de cycle de vie.

Comme vous l'avez vu dans le screencast :

- Le `constructeur` est invoqué en premier, comme pour n'importe quel objet, lorsque le composant apparaît pour la première fois dans un DOM virtuel... Il reçoit les props initiales en argument.

- Puis vient le `render`. C'est ensuite le moment où est appelé `componentDidMount()` (une fois que le composant est monté sur le DOM).
- Après, s'il y a une mise à jour et que le composant est re-render, `componentDidUpdate` est appelé.
- Et juste avant que le composant soit retiré du DOM, c'est au tour de `componentWillUnmount` d'être appelé.

Il existe d'autres méthodes de cycle de vie, mais vous aurez moins souvent besoin de celles-ci. Vous pourrez en apprendre davantage [dans la documentation de React](#).

En ce qui concerne les méthodes de cycle de vie, il existe un certain nombre de méthodes qui ont été dépréciées au fil du temps, telles que "componentWillMount". Vous verrez que depuis quelques années, ces méthodes dépréciées ont été renommées pour éviter qu'on les utilise. Ainsi, `componentWillMount` s'appelle désormais `UNSAFE_componentWillMount()`. Ça ne donne pas vraiment envie de l'utiliser, non ? 😊 Dans les dernières versions, elles ne font même plus partie de React.

## Appelez une API dans un composant classe avec `componentDidMount`

C'est le moment de mettre tout ça en application en appelant notre API Shiny dans un nouveau composant classe. Pour l'occasion, nous allons créer un nouveau composant qui permet d'afficher plus d'informations sur un freelance lorsqu'on clique sur la `Card`.

Nous allons commencer par permettre de naviguer sur <http://localhost/profile/:id>. Dans le fichier `index.jsx` à la racine de `/src`, nous avons ajouté une route pour bien rediriger l'utilisateur vers le profil du freelance :

```
1 ...
2 <Route path="/freelances">
3   <Freelances />
4 </Route>
5 <Route path="/profile/:id">
6   <Profile />
7 </Route>
8 <Route path="*">
9   <Error />
10 </Route>
11 ...
```

On enchaîne ensuite en permettant de naviguer sur la page `profile` en ajoutant un lien autour de la `Card` dans `/pages/Freelances/index.jsx`. Pour cela, on fait donc :

```
1 <CardsContainer>
2   {freelancersList?.map((profile) => (
3     <Link key={`freelance-${profile.id}`} to={`/profile/${profile.id}`}>
4       <Card
5         label={profile.job}
6         title={profile.name}
7         picture={profile.picture}
```

```

8           theme={theme}
9           />
10          </Link>
11        )}
12  </CardsContainer>

```

Et on n'oublie pas de supprimer la fonctionnalité de favoris dans le composant classe (pour ne pas avoir des étoiles ajoutées inutilement). Il ne nous reste plus qu'à développer `pages/Profile/index.jsx`... elle-même !

On va commencer par récupérer l'`id` du freelance dont on veut afficher le profil dans les paramètres. Mais oh oh... Comment on va faire ?

Comme on l'avait fait pour la page `Survey` : il nous suffit d'utiliser `useParams`, non ?!

Eh bien non ! Souvenez-vous : les hooks sont uniquement accessibles depuis les composants fonction. 🤔

On va donc devoir utiliser une méthode différente pour la déclaration de route qui nous permet d'accéder à l'historique. Dans `/src/index.jsx`, on fait donc :

```

1 <Route
2   path="/profile/:id"
3   render={(props) => <Profile {...props} />}
4 />

```

javascript

Ce qui va nous permettre d'accéder à l'objet `match` dans nos props.

Dans `pages/Profile/index.jsx`, on déclare donc notre composant, puis on récupère le paramètre passé dans l'URL avec `this.props.match.params.id`, ce qui nous donne :

```

1 import { Component } from 'react'
2
3 class Profile extends Component {
4   render() {
5     const { id } = this.props.match.params
6     return <div><h1>Freelance : {id}</h1></div>
7   }
8 }
9
10 export default Profile

```

javascript

Yaaay ! Notre paramètre s'affiche bien. 🎉

Passons maintenant aux choses sérieuses en lançant notre appel API dans la méthode de cycle de vie `componentDidMount()`. Encore une fois, nous allons devoir nous passer de notre hook `useFetch` puisque nous sommes dans un composant classe.

On commence donc par notre `constructor`... Ici, si vous regardez l'API, vous verrez que nous allons récupérer un objet `profileData`; il nous faudra donc `profileData` dans notre state.

```

1 constructor(props) {
2   super(props)

```

javascript

```

3   this.state = {
4     profileData: {},
5   }
6 }
```

Pour le fetch, vous pouvez réutiliser le code que vous aviez dans `useFetch`, version Promise. On le met tout simplement dans `componentDidMount()`, ce qui nous donne :

javascript

```

1 componentDidMount() {
2   const { id } = this.props.match.params
3
4   fetch(`http://localhost:8000/freelance?id=${id}`)
5     .then((response) => response.json())
6     .then((jsonResponse) => {
7       this.setState({ profileData: jsonResponse?.freelanceData })
8     })
9 }
```

Et..... Ça fonctionne bien ! 🎉

Si vous aviez voulu utiliser la syntaxe `async` / `await`, ça aurait donné ça :

javascript

```

1 componentDidMount() {
2   const { id } = this.props.match.params
3   const fetchData = async () => {
4     const response = await fetch(`http://localhost:8000/freelance?id=${id}`)
5     const jsonResponse = await response.json()
6     if (jsonResponse && jsonResponse.freelanceData) {
7       this.setState({ profileData: jsonResponse?.freelanceData })
8     }
9   }
10  fetchData()
11 }
```

Il ne reste plus qu'à afficher ce qui nous est retourné par l'API. Et voilà notre composant :

javascript

```

1 import { Component } from 'react'
2
3 class Profile extends Component {
4   constructor(props) {
5     super(props)
6     this.state = {
7       profileData: {},
8     }
9   }
10
11 componentDidMount() {
12   const { id } = this.props.match.params
13
14   fetch(`http://localhost:8000/freelance?id=${id}`)
15     .then((response) => response.json())
16     .then((jsonResponse) => {
17       this.setState({ profileData: jsonResponse?.freelanceData })
18     })
19   }
20 }
```

```
21 render() {
22     const { profileData } = this.state
23     const {
24         picture,
25         name,
26         location,
27         tjm,
28         job,
29         skills,
30         available,
31         id,
32     } = profileData
33
34     return (
35         <div>
36             <img src={picture} alt={name} height={150} width={150} />
37             <h1>{name}</h1>
38             <span>{location}</span>
39             <h2>{job}</h2>
40             <div>
41                 {skills &&
42                     skills.map((skill) => (
43                         <div key={`skill-${skill}-$id`}>{skill}</div>
44                     )))
45             </div>
46             <div>{available ? 'Disponible maintenant' : 'Indisponible'}</div>
47             <span>{tjm} € / jour</span>
48         </div>
49     )
50 }
51 }
52
53 export default Profile
```

Et voilà ! Vous avez un tout nouveau composant `Profile` qui permet d'afficher le profil de vos freelances, et le tout écrit avec un composant classe. ☺ Pour l'instant, ce n'est pas stylisé, mais vous trouverez tout sur le style dont vous avez besoin sur le repository GitHub du cours.

## Exercez-vous



C'est le moment de mettre en pratique ce que vous avez appris. Vous pouvez commencer l'exercice sur la branche `P4C2-begin` où vous trouverez :

- le code que nous avons mis en place dans ce chapitre ;
- avec du style ;
- ainsi que l'utilisation du Contexte pour récupérer `theme` ... avec la manière de faire des composants classe.

Cette fois-ci, vous allez convertir le composant classe que nous venons de créer dans `pages/Profile/index.jsx` ... en composant fonction.

Comme toujours, vous trouverez la solution de cet exercice sur la branche [P4C2-solution](#).

## En résumé

- Entre le moment où il est monté dans le DOM et le moment où il en est retiré, un composant passe par différentes étapes.
- Les méthodes de cycle de vie auxquelles on accède depuis les composants classe permettent d'exécuter notre code à des moments précis qui correspondent à ces étapes.
- `componentDidMount()` ... est la méthode privilégiée pour lancer un appel API.

*Et voilà ! Vous avez fait le tour (certes rapide) des composants classe. Vous avez maintenant les bases pour évoluer sur une codebase qui comporte des composants classe. Dans le prochain chapitre, vous trouverez quelques conseils pour voler de vos propres ailes dans l'écosystème React. Alors à tout de suite !*

[MARK THIS CHAPTER AS UNFINISHED](#)

[!\[\]\(3cb60d42b10e53f9522bb0b392c1c4cd\_img.jpg\) APPRIVOISEZ LES ANCIENNES SYNTAXES  
DE REACT](#)

[!\[\]\(d0262bbe9d2356661a2e89321dfcc781\_img.jpg\) EXPLOITEZ VOS ACQUIS POUR ALLER  
PLUS LOIN](#)

## Teacher

### Alexia Toulmet

Développeuse Fullstack React/Node freelance passionnée par le frontend 

OPENCLASSROOMS 

OPPORTUNITIES 

SUPPORT 

FOR BUSINESS



MORE



English

