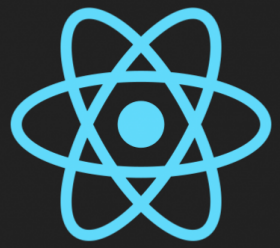

REACT live 7

Formateur :
TAUPIN Jérémy

Props Types React

Lors du 4 ème live, nous avions vue qu'il était possible de passer des données entre un composant parent et un enfant , grâce aux props , seulement nous ne savons pas quel types de props est passée et nous ne verifions donc pas ces derniers.

Les Props Types sont une fonctionnalité de React qui permet de définir le type de ces propriétés (props). Cela peut être utile pour deux raisons principales :

- Les Props Types peuvent vérifier que les props sont passées au composant avec les types corrects. Si un prop n'a pas le type attendu, React initialisera une erreur en développement pour vous informer de l'erreur. Cela aide à découvrir et corriger les erreurs de configuration plus rapidement.

- Les Props Types peuvent également être utilisés pour documenter le composant en décrivant les props attendues et les types de données associées. Cela peut aider la communication entre divers développeurs.

Utiliser les prop-types dans un composant fonctionnel

Dans un composant fonctionnel React, nous recevons les props en argument

Ces derniers ne sont pas vérifié de base.

Nous pouvons cependant les verifiers grâce à prop-types, un package node installé avec React et servant à définir les règles des prop-types

```
import PropTypes from 'prop-types'

const MyFunctionalComponent = (props) => {
  return (
    <div>
      <p>Name: {props.name}</p>
      <p>Age: {props.age}</p>
      <p>
        Is Married: {props.isMarried.toString()}
      </p>
    </div>
  )
}

MyFunctionalComponent.propTypes = {
  name: PropTypes.string,
  age: PropTypes.number,
  isMarried: PropTypes.bool
}
```

Explication

Import de PropTypes depuis le package prop-types

Nous initialisons notre composant fonctionnel avec les props en argument

Nous exploitons les props dans le composant retourné en les mettant entre accolade

Nous exploitons l'outil propTypes sur le composant.
Par exemple ici nous indiquons que name sera du type String donc chaine de caractère.

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

const MyFunctionalComponent = (props) => {
  return (
    <div>
      <p>Name: {props.name}</p>
      <p>Age: {props.age}</p>
      <p>
        Is Married: {props.isMarried.toString()}
      </p>
    </div>
  )
}

MyFunctionalComponent.propTypes = {
  name: PropTypes.string,
  age: PropTypes.number,
  isMarried: PropTypes.bool
}
```

Vérifications des states

prop types n'a pas un fonctionnement optimal sous certaines conditions.

Par exemple avec le state.

Il est donc préférable avec le state de verifier les types grâce à un code personnalisé.

Ce dernier peut être décliner de plusieurs méthode possible :

-> Une fonction = Un type

-> Une fonction = Une verification (valeur et type en argument)

-> Une fonction dans une class verification = Un type

Dans tous les cas, le mieux reste de créer des règles génériques, afin de garder une utilisation plus fréquente de ces dernières

Découvrons en 2 !

Créer une fonction vérifiant le state

Dans ce code, 2 méthodes sont montrées afin de pouvoir vérifier le type des variables en states.

Une est appelée directement dans le composant, quant à l'autre elle exploite une fonction importée.

Dans tous les cas, les 2 doivent être appelée au sein d'un useEffect qui surveillera la variable.

```
if (typeof this.state.value !== 'string') {
  // Code erreur pour une variable
  // devant être une chaine de caractère
}
else {
  return true
}

// Ou une approche plus générique :
/* Fichier ./functions/testType.js */
const testType = (value, type) =>{
    return typeof value === type
}
export default testType

// Avec un import et un appel :
/* Fichier ./components/Compo.js */
import {testType} from './functions'

const Compo = (props) => {
  const [error, setError] = useState(false)
  const [string, setString] = useState()
  useEffect(() => {
    !testType(string, 'string')
        ? setError('Bad format for
'+string) setError(false)
    return () => {}
  }, [string])
}
```

06

Explication

La fonction vérifie le type du state grâce à la fonction typeof de JS est vérifie si cette dernière n'est pas une String (chaine de caractère). Ici nous traitons donc l'exception.
Cette fonction devra se trouver dans un useEffect

Ici le code est utilisé comme utilitaire et est placé dans un fichier spécifique, il prend comme option, la valeure et le type. Ainsi cette fonction peut être utilisée pour tout les types.

Nous importont la fonction précèdemment créée et nous l'exploitons dans un useEffect surveillant la state string.
Grâce à une condition ternaire nous pouvons générer une erreur si cette dernière ne correspond pas au type attendu.

```
if (typeof this.state.value !== 'string') {
  // Code erreur pour une variable
  // devant être une chaine de caractère
}
else {
  return true
}
```

```
// Ou une approche plus générique :
/* Fichier ./functions/testType.js */
const testType = (value, type) =>{
    return typeof value === type
}
export default testType
```

```
// Avec un import et un appel :
/* Fichier ./components/Compo.js */
import {testType} from './functions'

const Compo = (props) => {
  const [error, setError] = useState(false)
  const [string, setString] = useState()
  useEffect(() => {
    !testType(string, 'string')
        ? setError('Bad format for
'+string) setError(false)
    return () => {}
  }, [string])
}
```

Test

Test avec Jest

L'une des étapes clés d'une application est la réalisation des tests. Ces derniers permettent de favoriser un meilleur fonctionnement de l'application et éviter des problèmes de régression lors de l'évolution du code

Testons notre application !

Test par défaut React

Import de la librairie de test

Ici est importé render et screen depuis la librairie de test de react

-> Render = Ce chargera de réaliser un rendu du composant
-> Screen = Constituera l'écran de ce rendu

Import du composant

Ici le composant à tester est importé pour pouvoir l'exploiter

Réalisation du test

Grâce aux éléments importé, nous testons que la fenêtre rendu par le composant **App** contient bien le text **'learn reapmolct'**

Celui ci n'étant pas présent, **le test devrait échouer.**

Lors de l'initialisation d'un projet React, ce dernier est fourni avec un premier test.

```
import {
  render,
  screen
} from '@testing-library/react'
import App from './App'

test('renders learn react link', () => {
  render(<App />)
  const lE = screen.getByText(/learn reapmolct/i)
  expect(lE).toBeInTheDocument()
})
```

Parmis les librairies de test JS, React utilise jest afin de fonctionner.