Développement front

MMI 2 – TP#3 S4

IUT CERGY-PONTOISE

Danielo **JEAN-LOUIS**
Michele **LINARDI**

Patterns – Patrons de conception

- Ensemble de syntaxes et façons de travailler propre à React
  - Conditions
  - Boucles
  - Requêtes asynchrones
- Permettent d'afficher du contenu

Conditions

- Permet d'afficher des éléments en fonction de conditions définies
- Plusieurs syntaxes possibles

Source(s) :

- https://fr.reactjs.org/docs/conditional-rendering.html

Conditions – if classique

```
function Formulaire(props) {
    const isUserRegistered = props.isUserRegistered;
    if (isUserRegistered) {
        return <LogInForm />;
    }

    return <SignInForm />;
}
```

On affiche un composant spécifique en fonction de la valeur de la props "isUserRegistered"

Source(s) :

- https://fr.reactjs.org/docs/conditional-rendering.html

Conditions – if ternaire

```
function Message(props) {
    const isUserRegistered = props.isUserRegistered;
    return (
        <div>
            L’utilisateur <b>{isUserRegistered ? 'a un compte' : 'n’a pas de compte'}</b>.
        </div>
    );
}
```

Le texte s'affiche en fonction de la props "isUserRegistered".
Note : cette syntaxe doit impérativement être dans le « return »

Source(s) :

- https://fr.reactjs.org/docs/conditional-rendering.html

Conditions – if avec l'opérateur &&

```
function Message(props) {
    const [isUserConnected, setIsUserConnected] = React.useState(false);

    const changeStatus = () => {
        // Cette syntaxe inverse un booléen.
        // Si c'est vrai, c'est faux. Si c'est faux, c'est vrai.
        setIsUserConnected(!isUserConnected)
    }

    return (
        <>
            {isUserConnected &&
                <h2>
                    Bonjour {props.name}.
                </h2>
            }
            <button onClick={changeStatus}>Changer</button>
        </>
    );
}
```

Le h2 ne s'affichera que si et seulement si le state "isUserConnected" est égal à "true"

Source(s) :

- https://fr.reactjs.org/docs/conditional-rendering.html

Pratiquons ! - Découvrons ReactJS (Partie 1)

Pré-requis :

- Avoir la ressource ressources/patterns/if

A télécharger ici :
https://download-directory.github.io?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s4/travaux-pratiques/numero-3/ressources

Boucles

- Permet d'afficher une liste d'éléments
- Utilisation de la méthode .map()
  - Renvoie un nouveau tableau modifié

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array/map
- https://fr.reactjs.org/docs/lists-and-keys.html

Boucles

```
function NumberList() {
    const numbers = [1, 2, 3, 4, 5];

    return (
        <ul>
            {numbers.map((number) =>
                <li>
                    {number}
                </li>
            )}
        </ul>
    );
}
```

**Ici on itère dans notre tableau « numbers » et on retourne chaque nombre contenu dans une balise **

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array/map
- https://fr.reactjs.org/docs/lists-and-keys.html

Boucles

```
function EmployeesList() {
    const listEmployees = [{
        name: "Patrick",
        job: "Sr. Graphic Designer"
    }, {
        name: "Joanne",
        job: "CEO"
    }];

    return (
        <ul>
            {listEmployees.map((employee) =>
                <li>
                    {employee.name} - {employee.job}
                </li>
            )}
        </ul>
    );
}
```

Ici, notre tableau contient des objets, on accède donc à chaque propriété de chaque objet pour en afficher le contenu.

Et comme le cap précédent, on itère sur chaque élément du tableau

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array/map
- https://fr.reactjs.org/docs/lists-and-keys.html

Boucles

- Possibilité de faire des boucles dans des boucles

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array/map
- https://fr.reactjs.org/docs/lists-and-keys.html

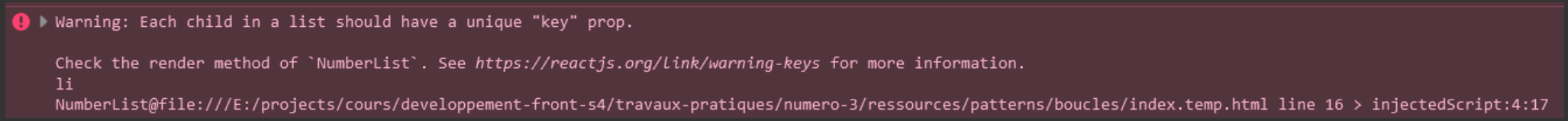
Pratiquons ! - Découvrons ReactJS (Partie 1)

Pré-requis :

- Avoir la ressource ressources/patterns/boucles

A télécharger ici :
https://download-directory.github.io?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s4/travaux-pratiques/numero-3/ressources

Boucles



React lève un avertissement

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array/map
- https://fr.reactjs.org/docs/lists-and-keys.html

Algorithme de résolution

- React ne remplace que les éléments qui ont été modifiés entre chaque mise à jour
- Dans les boucles, React a besoin d'un identifiant unique pour chaque élément
  - Identifiant représenté par la props "key"

Algorithme de résolution – Props key

- Utilisable sur n'importe quelle balise ou composant
- Permet d'identifier un élément dans une boucle
- Dans chaque boucle la valeur de la props « key » doit être unique

Algorithme de résolution – Props key

```jsx
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
```

La props "key" a une valeur unique, ici un numéro. Ceci permet à React d'identifier les éléments.

Algorithme de résolution – Props key

- Préférez la clé "id" de votre objet comme valeur pour la props "key"
  - Si votre jeu de données retourne la clé “id”

Pratiquons ! - Découvrons ReactJS (Partie 2)

Pré-requis :

- Avoir la ressource ressources/patterns/boucles

A télécharger ici :
https://download-directory.github.io?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s4/travaux-pratiques/numero-3/ressources

Requêtes asynchrones

- Permettent de récupérer des ressources externes de façon asynchrones
  - Ne bloquent pas l'ensemble du site lors du chargement
- Utilisation du hook React.useEffect(() => {})
- Appelées également “requêtes AJAX”

React.useEffect(() => {})

- Hook permettant de créer des effets de bord
  - Effectuer une tâche qui n’a pas d’incidence sur le reste
- Idéal pour utiliser la fonction **fetch()**
- Appelé à l'initiation de l'application

Source(s) :
- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch
- https://fr.reactjs.org/docs/hooks-effect.html

fetch

- API native de javascript
  - Ne nécessite pas de dépendance externe
- Version moderne de l’API XMLHttpRequest
  - **Évitez d’utiliser XMLHttpRequest**
- Permet de communiquer avec une ressource externe :
  - API
  - Fichier de texte
  - ...

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

fetch

- **Ne fonctionne pas sans serveur pour un fichier local**
  - Nécessite un serveur php, nodejs...
- Utilise un système de promesses
- Peut être utilisé sans ReactJS

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

fetch - exemple

On indique à « fetch » où se trouve les données qu'on souhaite récupérer

Puis (then) quand les données sont récupérées, on effectue une action. On transforme les données en JSON

Puis (then) on affiche les donnes dans un console.log(). A noter que pour récupérer la valeur du bloc précédent, il faut impérativement la retourner

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

fetch - then

- Méthode de promesse
- Appelée **si et seulement si** l’action (précédente) a réussi
- Une promesse peut avoir une infinité de “then”

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

fetch - then

- Accepte en paramètre une fonction
- Fonction de paramètre doit impérativement renvoyer une valeur pour l’utiliser dans la promesse suivante

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

fetch - then

```
fetch('http://example.com/movies.json')
  .then((response) =>  {
    return response.json();
  })
  .then((data) => {
    console.log(data);
  });
```

Appel de la première promesse. Si tout se passe bien...

Appel de la deuxième promesse. Si tout se passe bien...

Appel de la troisième promesse, etc.

Et en cas de problème, appel de la méthode « catch() »

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch

fetch - catch

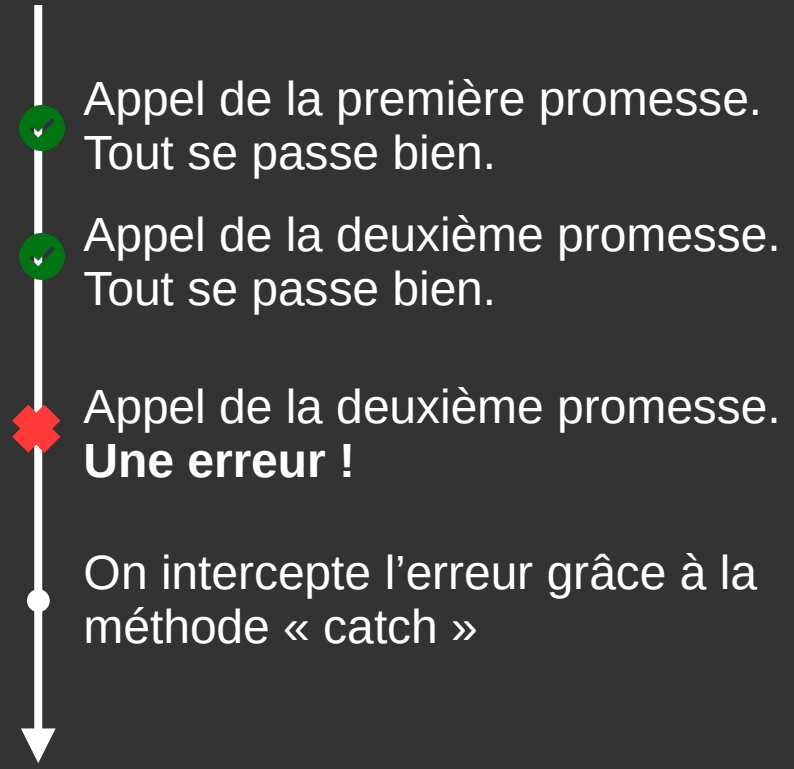
- Intercepte les erreurs (problème de connexion, refus d’API…)
- Unique par fetch
  - Gère toutes les erreurs de “fetch”
- Prend en paramètre une fonction

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch
- https://developer.mozilla.org/fr/docs/Learn/JavaScript/Asynchronous/Promises

fetch - catch

```
fetch('http://example.com/movies.json')
  .then((response) =>  {
    return response.json();
  })
  .then((data) => {
    console.log(data);
  })
  .catch((error) => {
    console.error('Error:', error);
  });
```



Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch
- https://developer.mozilla.org/fr/docs/Learn/JavaScript/Asynchronous/Promises

fetch - catch

- Intercepter les erreurs permet d’afficher le bon message au bon moment
  - Meilleur pour l’expérience utilisateur

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch
- https://developer.mozilla.org/fr/docs/Learn/JavaScript/Asynchronous/Promises

Pratiquons ! - Découvrons ReactJS (Partie 1)

Pré-requis :

- Avoir la ressource ressources/patterns/fetch

A télécharger ici :
https://download-directory.github.io?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s4/travaux-pratiques/numero-3/ressources

React.useEffect(() => {}) et fetch

```
function Example() {
    const [listItems, setListItems] = useState([]);

  1 React.useEffect(() => {
     2 fetch('monurl.com')
            .then(function (response) {
              3 return response.json();
            })
            .then(function (json) {
              4 setListItems(json)
            });
    }, []);

    /* [...] */
}
```

1) On définit notre hook avec notre promesse à l'intérieur
2) On fait une requête asynchrone vers l'url « monurl.com » grâce à l'API fetch
3) On transforme la réponse en JSON
4) On change la valeur de notre state « listItems » avec la réponse de l'API

Le tableau vide indique que notre React.useEffect ne doit être appelé qu'une seule foi. En absence, le contenu du hook serait appelé indéfiniment.

Pratiquons ! - Découvrons ReactJS (Partie 2)

Pré-requis :

- Avoir la ressource ressources/patterns/fetch

A télécharger ici :
https://download-directory.github.io?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s4/travaux-pratiques/numero-3/ressources

Requêtes asynchrones – Bonne pratique

- Avoir un state qui indique le chargement
  - Au début le state indique qu'on charge des données
  - Quand les données sont chargées, le state devient faux
  - Bien évidemment l’afficher visuellement

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/API/Fetch_API/Using_Fetch
- https://fr.reactjs.org/docs/hooks-effect.html

Chargement – Exemple code

```jsx
const MyComponent = (props) => {
  const [isLoading, setIsLoading] = React.useState(true);
  const [data, setData] = React.useState([]);
  /* [...] */
  if(isLoading) {
    return <p>Chargement en cours...</p>
  }
  return (
    <ul>
      {data.map((item) => {/* [...] */})}
    </ul>
  )
}
```

Dans cette partie, nous avons React.useEffect() qui :

- Rempli le state « data »
- Change le state « isLoading » vers false

(voir slide suivante)

Chargement – Exemple code

```
React.useEffect(() => {
  fetch("url.com")
    .then((res) => { return res.json() })
    .then((json) => {
      setData(json);
      setIsLoading(false)
    })
}, [])
```

Pratiquons ! - Découvrons ReactJS (Partie 3)

Pré-requis :

- Avoir la ressource ressources/patterns/fetch

A télécharger ici :
https://download-directory.github.io?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s4/travaux-pratiques/numero-3/ressources

Questions ?