

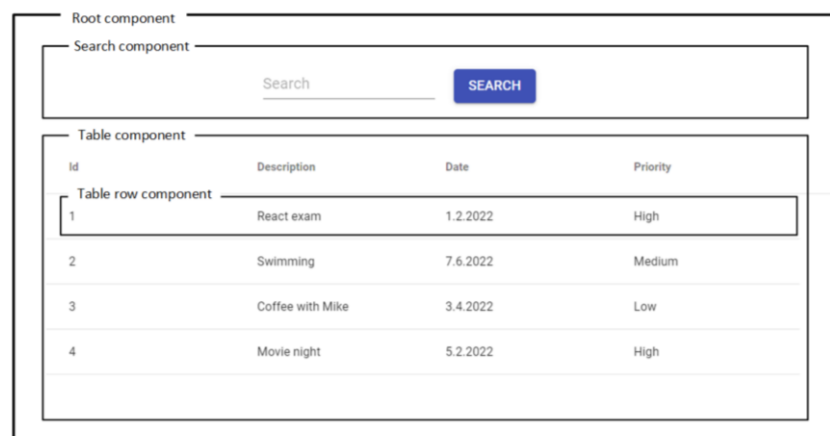
Cours INF3522 - Développement d'Applications N tiers

Lab 7 : Débuter avec React

Ce lab décrit les bases de la programmation React. Nous aborderons les compétences nécessaires pour créer des fonctionnalités de base pour le frontend React. Nous y verrons notamment les notions de composant, props, état, entre autres.

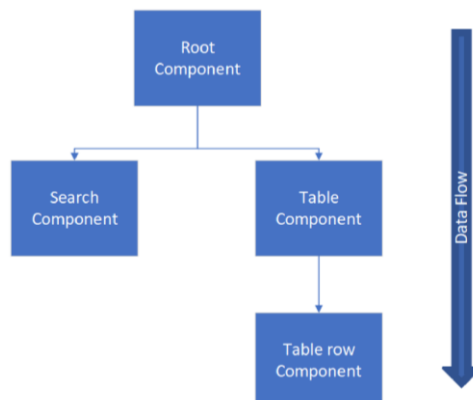
Comment créer des composants React

Selon Meta Platforms, Inc., React est une bibliothèque JavaScript pour les interfaces utilisateur (UI). React est basé sur des composants, et ces composants sont indépendants et réutilisables. Les composants sont les éléments de base de React. Lorsque vous commencez à développer une UI avec React, il est conseillé de commencer par créer des interfaces fictives (interfaces mock). De cette manière, il sera facile d'identifier les types de composants à créer et comment ils interagissent.



Les composants peuvent ensuite être disposés dans une hiérarchie arborescente, comme le montre la capture d'écran suivante. L'aspect important à comprendre avec React est que le flux de données se fait du composant parent vers le composant enfant. Nous verrons plus tard comment les données peuvent être transmises d'un composant parent à un composant enfant en utilisant les props :

À partir de la capture d'écran suivante de l'UI fictive, on peut voir comment l'UI peut être divisée en composants. Dans ce cas, il y aura un composant racine d'application, un composant de barre de recherche, un composant de tableau et un composant de ligne de tableau :

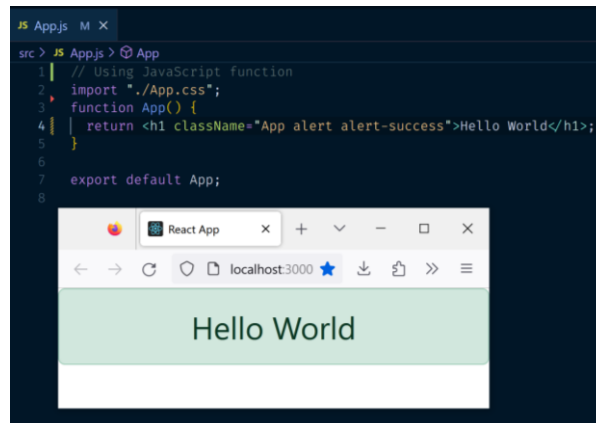


React utilise le modèle d'objet virtuel Document Object Model (VDOM) pour le rendu sélectif de l'interface utilisateur (UI), ce qui le rend plus rentable. Le VDOM est une copie légère du DOM, et la

manipulation du VDOM est beaucoup plus rapide que celle du véritable DOM. Après la mise à jour du VDOM, React le compare à une capture d'écran qui a été prise à partir du VDOM avant les mises à jour. Après la comparaison, React saura quelles parties ont été modifiées, et seules ces parties seront mises à jour dans le véritable DOM.

Un composant React peut être défini en utilisant une fonction JavaScript ou la classe JavaScript ES6. Nous approfondirons davantage l'ES6 dans la section suivante. Voici un exemple de code source de composant simple qui affiche le texte "Hello World".

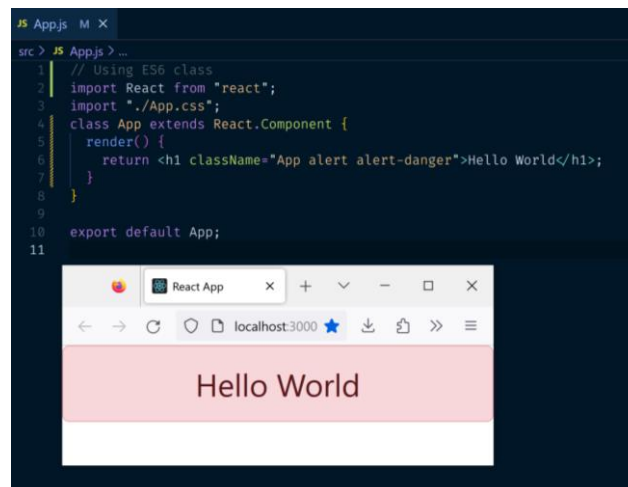
Le premier exemple utilise une fonction pour créer un composant :



```
1 // Using JavaScript function
2 import './App.css';
3 function App() {
4   return <h1 className="App alert alert-success">Hello World</h1>;
5 }
6
7 export default App;
```

The screenshot shows a web browser window titled "React App" at localhost:3000. The page displays "Hello World" in a green box with a success alert.

Le second exemple utilise une classe ES6 pour créer un composant :



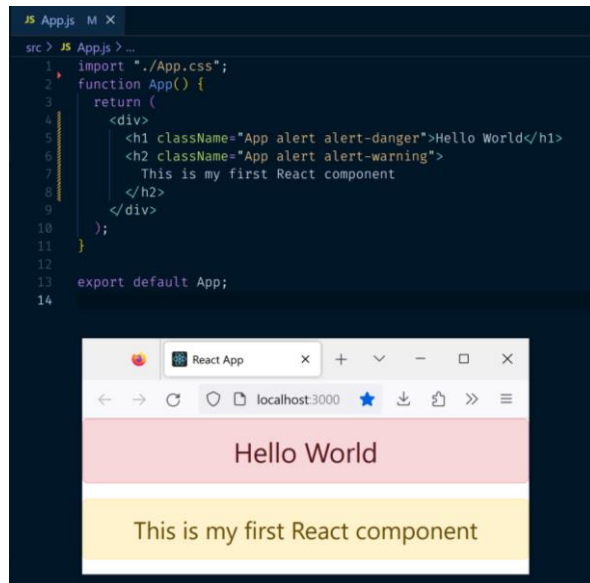
```
1 // Using ES6 class
2 import React from "react";
3 import './App.css';
4 class App extends React.Component {
5   render() {
6     return <h1 className="App alert alert-danger">Hello World</h1>;
7   }
8 }
9
10 export default App;
```

The screenshot shows a web browser window titled "React App" at localhost:3000. The page displays "Hello World" in a pink box with a danger alert.

Le composant qui a été implémenté en utilisant la classe contient la méthode `render()` requise, qui affiche et met à jour le rendu du composant. Si vous comparez les composants App en fonction et en classe, vous pouvez voir que la méthode `render()` n'est pas nécessaire dans le composant fonction. Avant la version 16.8 de React, vous deviez utiliser des composants de classe pour pouvoir utiliser des états. Maintenant, vous pouvez utiliser des hooks pour créer des états également avec des composants fonctionnels. Nous en apprendrons plus sur les états et les hooks plus tard.

Dans ce lab, nous créons des composants en utilisant des fonctions. Vous devez écrire moins de code lorsque vous utilisez des composants fonctionnels, mais vous pouvez toujours utiliser des composants de classe également.

Si votre composant renvoie plusieurs éléments, vous devez les envelopper à l'intérieur d'un seul élément parent. Pour cela, nous pouvons envelopper les éléments d'en-tête dans un élément tel qu'une div.



```
1 import './App.css';
2 function App() {
3   return (
4     <div>
5       <h1 className="App alert alert-danger">Hello World</h1>
6       <h2 className="App alert alert-warning">
7         This is my first React component
8       </h2>
9     </div>
10  );
11 }
12
13 export default App;
```

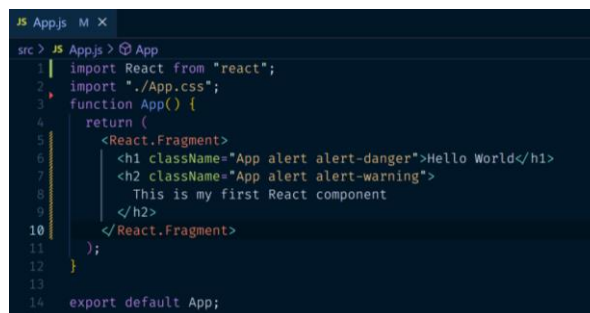
React App

localhost:3000

Hello World

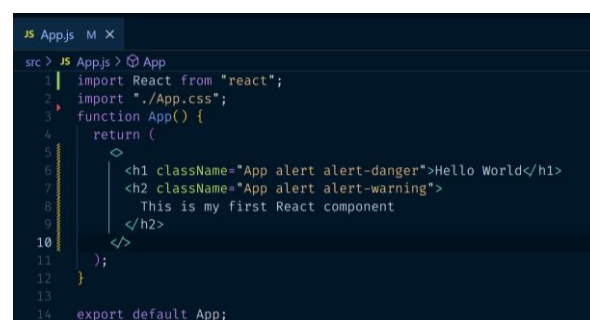
This is my first React component

Depuis la version 16.2 de React, nous pouvons également utiliser des fragments, comme indiqué dans l'extrait de code suivant :



```
1 import React from "react";
2 import './App.css';
3 function App() {
4   return (
5     <React.Fragment>
6       <h1 className="App alert alert-danger">Hello World</h1>
7       <h2 className="App alert alert-warning">
8         This is my first React component
9       </h2>
10    </React.Fragment>
11  );
12 }
13
14 export default App;
```

Il existe également une syntaxe plus courte pour les fragments, qui ressemblent à des balises JSX vides. Cela est illustré dans l'extrait de code suivant :



```
1 import React from "react";
2 import './App.css';
3 function App() {
4   return (
5     <>
6       <h1 className="App alert alert-danger">Hello World</h1>
7       <h2 className="App alert alert-warning">
8         This is my first React component
9       </h2>
10    </>
11  );
12 }
13
14 export default App;
```

JavaScript XML (JSX) et styles

JSX est une extension de la syntaxe de JS. Il n'est pas obligatoire d'utiliser JSX avec React, mais il présente certains avantages qui facilitent le développement. JSX, par exemple, prévient les attaques d'injection car toutes les valeurs sont échappées dans le JSX avant d'être rendues. La fonctionnalité la plus utile est que vous pouvez incorporer des expressions JavaScript dans le JSX en les enveloppant avec des accolades.

Dans l'exemple suivant, nous pouvons accéder aux props du composant lors de l'utilisation de JSX. Les props du composant sont abordées plus tard :

```
function App(props) {  
  return <h1>Bonjour le monde {props.user}</h1>;  
}
```

Vous pouvez également transmettre une expression JavaScript en tant que props, comme indiqué dans l'extrait de code suivant :

```
<Hello count={2+2} />
```

JSX est compilé en JavaScript classique par Babel.

Vous pouvez utiliser à la fois la stylisation interne et externe avec les éléments JSX de React. Voici deux exemples de stylisation en ligne. Le premier définit le style à l'intérieur de l'élément div :

```
<div style={{ height: 20, width: 200 }}>  
  Bonjour  
</div>
```

Le deuxième exemple crée d'abord un objet de style, qui est ensuite utilisé dans l'élément div. Le nom de l'objet doit suivre la convention de dénomination camelCase :

```
const divStyle = { color: 'red', height: 30 };  
const MyComponent = () => (  
  <div style={divStyle}>Bonjour</div>  
);
```

Comme indiqué dans la section précédente, vous pouvez importer une feuille de style vers un composant React. Pour référencer des classes à partir d'un fichier externe de feuilles de style en cascade (CSS), vous devez utiliser un attribut className, comme indiqué dans l'extrait de code suivant :

```
import './App.css';  
...  
<div className="App">Ceci est mon application</div>
```

Dans la prochaine section, nous verrons les props et les états de React.

Propriétés (props) et état (state)

Les props et les états sont les données d'entrée pour le rendu du composant. Le composant est rendu à nouveau lorsque les props ou les états changent.

Props

Les props sont des entrées pour les composants, et ils sont un mécanisme permettant de transmettre des données du composant parent à son composant enfant. Les props sont des objets JavaScript, donc ils peuvent contenir plusieurs paires clé-valeur.

Les props sont immuables, ce qui signifie qu'un composant ne peut pas modifier ses props. Les props sont reçues du composant parent. Un composant peut accéder aux props à travers l'objet props qui est passé au composant fonction en tant que paramètre. Par exemple, regardons le composant suivant :

```
function Hello() {  
  return <h1>Bonjour John</h1>;  
}
```

Le composant affiche simplement un message statique et n'est pas réutilisable. Au lieu d'utiliser un nom codé en dur, nous pouvons transmettre un nom au composant Hello en utilisant les props, comme ceci :

```
function Hello(props) {  
  return <h1>Bonjour {props.user}</h1>;  
}
```

Le composant parent peut envoyer les props au composant Hello de la manière suivante :

```
<Hello user="John" />  
  
<Hello user="Daffy" />
```

Vous pouvez également passer plusieurs props à un composant, comme indiqué ici :

```
<Hello firstName="John" lastName="Doe" />
```

Maintenant, vous pouvez accéder aux deux props dans le composant en utilisant l'objet props, comme suit :

```
function Hello(props) {  
  return <h1>Bonjour {props.firstName} {props.lastName}</h1>;  
}
```

Maintenant, la sortie du composant est Bonjour John Doe.

On peut également déstructurer l'objet props, ce qui peut être assez utile lorsque nous avons plusieurs champs dans l'objet props :

```
function Hello( { firstName, lastName } ) {  
  return <h1>Bonjour {firstName} {lastName}</h1>;  
}
```

État (State)

La valeur de l'état peut être mise à jour à l'intérieur d'un composant. L'état est créé en utilisant la fonction `useState` du hook. Elle prend un argument, qui est la valeur initiale de l'état, et renvoie un tableau de deux éléments. Le premier élément est le nom de l'état, et le deuxième élément est une fonction utilisée pour mettre à jour la valeur de l'état. La syntaxe de la fonction `useState` est indiquée dans l'extrait de code suivant :

```
const [state, setState] = React.useState(valeurInitiale);
```

L'exemple de code suivant crée une variable d'état appelée name, et la valeur initiale est Bugs :

```
const [name, setName] = React.useState('Bugs');
```

Vous pouvez également importer la fonction useState de React, comme ceci :

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

Ensuite, vous n'avez pas besoin de taper le mot-clé React, comme indiqué ici :

```
const [name, setName] = useState('Bugs');
```

La valeur de l'état peut maintenant être mise à jour en utilisant la fonction setName, comme illustré dans l'extrait de code suivant. C'est la seule façon de modifier la valeur de l'état :

```
// Mettre à jour la valeur de l'état name
```

```
setName('John');
```

Vous ne devez jamais mettre à jour la valeur de l'état directement en utilisant l'opérateur =. Si vous mettez à jour directement l'état, React ne réaffichera pas l'interface utilisateur et vous obtiendrez également une erreur car vous ne pouvez pas réassigner la variable const, comme indiqué ici :

```
// Ne faites pas cela, l'interface utilisateur ne se réaffichera pas
```

```
name = 'John';
```

Si vous avez plusieurs états, vous pouvez appeler la fonction useState plusieurs fois, comme indiqué dans l'extrait de code suivant :

```
// Créer deux états : firstName et lastName
```

```
const [firstName, setFirstName] = useState('John');
```

```
const [lastName, setLastName] = useState('Doe');
```

Maintenant, vous pouvez mettre à jour les états en utilisant les fonctions setFirstName et setLastName, comme indiqué dans l'extrait de code suivant :

```
// Mettre à jour les valeurs des états
```

```
setFirstName('Daffy');
```

```
setLastName('Duck');
```

Vous pouvez également définir l'état en utilisant un objet, comme ceci :

```
const [name, setName] = useState({  
  firstName: 'John',  
  lastName: 'Doe'  
});
```

Maintenant, vous pouvez mettre à jour à la fois les paramètres de l'objet d'état `firstName` et `lastName` en utilisant la fonction `setName`, comme ceci :

```
setName({ firstName: 'Daffy', lastName: 'Duck' });
```

Si vous souhaitez effectuer une mise à jour partielle de l'objet, vous pouvez utiliser l'opérateur `spread`. Dans l'exemple suivant, nous utilisons la syntaxe de l'opérateur `spread (...)` d'objet introduite dans ES2018. Elle clone l'objet d'état `name` et met à jour la valeur `firstName` pour qu'elle soit `Bugs` :

```
setName({ ...name, firstName: 'Bugs' });
```

Un état peut être accédé en utilisant le nom de l'état, comme indiqué dans l'exemple suivant. La portée de l'état est le composant, donc il ne peut pas être utilisé en dehors du composant dans lequel il est défini :

```
// Affiche Bonjour John
import React, { useState } from 'react';
function MyComponent() {
  const [firstName, setFirstName] = useState('John');
  return <div>Bonjour {firstName}</div>;
}
```

Nous avons maintenant appris les bases des états et des props, mais nous en apprendrons plus sur les états plus tard.

Composants sans état (stateless components)

Le composant sans état est simplement une fonction JavaScript pure qui prend des props en argument et renvoie un élément React. Voici un exemple de composant `stateless` :

```
function HeaderText(props) {
  return (
    <h1>
      {props.text}
    </h1>
  )
}
export default HeaderText;
```

Notre exemple de composant `HeaderText` est également appelé composant pur. Un composant est considéré comme pur si sa valeur de retour est toujours la même pour les mêmes valeurs d'entrée.

React fournit `React.memo()`, qui optimise les performances des composants fonctionnels purs. Dans l'extrait de code suivant, nous enveloppons notre composant avec `memo()` :

```
import React, { memo } from 'react';
function HeaderText(props) {
  return (
    <h1>
      {props.text}
    </h1>
  )
}
export default memo(HeaderText);
```

Maintenant, le composant est rendu et mémorisé. Lors du rendu suivant, React renvoie un résultat mémorisé si les props n'ont pas changé. L'un des avantages des composants fonctionnels est les tests unitaires, qui sont simples car leur valeur de retour est toujours la même pour les mêmes valeurs d'entrée.

Rendu conditionnel

Vous pouvez utiliser une instruction conditionnelle pour afficher des interfaces utilisateur différentes en fonction d'une condition vraie ou fausse. Cette fonctionnalité peut être utilisée, par exemple, pour afficher ou masquer certains éléments, gérer l'authentification, etc.

Dans l'exemple suivant, nous vérifierons si `props.isLoggedIn` est vrai. Si c'est le cas, nous rendrons le composant `<Logout />` ; sinon, nous rendrons le composant `<Login />`. Cela est maintenant mis en œuvre à l'aide de deux instructions de retour distinctes :

```
import Login from "./Login";
import Logout from "./Logout";

function Condition(props) {
  if (props.isLoggedIn)
    return <div><Logout /></div>;
  return <div><Login /></div>;
}
```

export default Condition;

Vous pouvez également implémenter cela en utilisant les opérateurs logiques **condition ? true : false**, ce qui ne nécessite qu'une seule instruction de retour, comme illustré ici :

```
import Login from "./Login";
import Logout from "./Logout";

function Condition(props) {
  return <div>{props.isLoggedIn ? <Logout /> : <Login />}</div>;
}
export default Condition;
```

Une dernière façon d'utiliser une instruction `if else` est d'utiliser les courts circuits :

```
import Login from "./Login";
import Logout from "./Logout";

function Condition(props) {
  return (
    <div>
      {props.isLoggedIn && <Logout />}
      {!props.isLoggedIn && <Login />}
    </div>
  );
}
export default Condition;
```