

# Introduction à React

React, introduit par Facebook en 2013, est rapidement devenu une bibliothèque JavaScript de premier plan pour le développement d'interfaces utilisateur dynamiques. Son approche innovante en matière de composants réutilisables et de rendu efficace a transformé le développement front-end.

React utilise le principe du SPA (Single Page Application), c'est une application web qui fonctionne à l'intérieur d'une seule page. Cela offre une expérience utilisateur plus fluide et interactive.

React comme les autres framework du moment (Angular, Vue.js, etc..) est adapté pour construire des SPA en raison de sa capacité à gérer efficacement les mises à jour de l'interface utilisateur et à réagir aux interactions de l'utilisateur en temps réel.

## Fonctionnement

### Le DOM Virtuel et React

#### **Qu'est-ce que le DOM Virtuel ?**

Le DOM Virtuel est une abstraction légère du DOM (Document Object Model) réel. React crée une copie en mémoire du DOM pour optimiser les mises à jour de l'interface utilisateur.

Cela permet à React de minimiser les opérations coûteuses sur le DOM réel, ce qui améliore les performances, en particulier dans les applications complexes.

Lorsqu'un composant change d'état ou reçoit de nouvelles données (props), React met à jour d'abord le DOM Virtuel.

React compare ensuite le DOM Virtuel avec le DOM réel (processus appelé "diffing" ou "réconciliation" en français) et applique les modifications nécessaires de manière optimale.

## JSX

C'est une extension syntaxique de JavaScript utilisée dans React pour décrire l'interface utilisateur. Il ressemble à du HTML dans son apparence, mais il a la puissance de JavaScript.



`function Welcome(props) {...}` : C'est la définition d'un composant fonctionnel en React. Les composants sont comme des fonctions JavaScript qui retournent des éléments d'interface utilisateur.

`props`: Ce sont des propriétés passées au composant, semblables aux arguments d'une fonction. Ici, `props.name` est utilisé pour accéder à la propriété `name` qui est passée au composant.

`<h1>Bonjour, {props.name}!</h1>` : C'est la partie JSX. Elle ressemble à du HTML mais avec des capacités supplémentaires.

`{props.name}` est une expression JavaScript. Dans JSX, vous pouvez intégrer des expressions JavaScript en les enveloppant dans des accolades `{}`. Ici, `props.name` sera remplacé par la valeur de la propriété `name` lorsque le composant est utilisé.

# Les outils nécessaires à React

L'utilisation de React dans le développement web moderne implique souvent un écosystème d'outils et de technologies complémentaires. Voici une explication des principaux outils souvent utilisés avec React :

## **NPM (Node Package Manager)**

NPM est un gestionnaire de paquets pour JavaScript. Il permet de partager et d'utiliser des bibliothèques de code, appelées "paquets" ou "modules".

NPM est utilisé pour installer et gérer les dépendances de votre projet React.

## **Babel**

Babel est un outil de transcompilation qui permet d'écrire du code JavaScript moderne (ES6, ES7, etc.) et de le convertir en une version compatible avec les navigateurs actuels.

Babel transforme également le JSX, utilisé dans React, en JavaScript standard.

## **Webpack**

Webpack est un "bundler", c'est-à-dire un outil qui regroupe tous les modules de votre application en un ou plusieurs fichiers, optimisés pour le navigateur.

Il gère les dépendances et peut charger différentes sortes de ressources (comme le CSS, les images, ou le HTML) en les transformant en modules JavaScript.:

Un "bundler" est un outil général qui regroupe, compile et organise les fichiers et modules de votre projet. Il optimise les ressources en réduisant leur taille et en les adaptant pour une distribution efficace sur le web.

## **Create React App**

Create React App est un environnement prêt à l'emploi pour créer des applications React. Il encapsule plusieurs outils (comme Babel et Webpack) et configure automatiquement votre projet. Il permet de démarrer un nouveau projet React rapidement, sans configuration manuelle complexe. Autrement dit, créer son application React grâce à create react app regroupe tous les outils nécessaire au bon fonctionnement de react.

## Node.js

Node.js joue un rôle essentiel dans le développement d'applications React, bien que React lui-même soit une bibliothèque front-end et Node.js soit principalement utilisé pour le développement back-end. Voici quelques-unes des manières dont Node.js est utilisé dans le contexte de React :

Il fournit l'environnement nécessaire pour exécuter divers outils et utilitaires de développement utilisés dans les projets React, tels que Create React App, Webpack, Babel, et NPM (Node Package Manager).

Avec NPM ou Yarn (un autre gestionnaire de paquets), qui fonctionnent sur Node.js, les développeurs peuvent installer et gérer facilement les bibliothèques et les outils nécessaires pour leurs projets React.

Il permet également de mettre en place un serveur de développement local pour React, offrant des fonctionnalités comme le "hot reloading", qui rafraîchit automatiquement votre application React dans le navigateur lorsque vous modifiez le code.

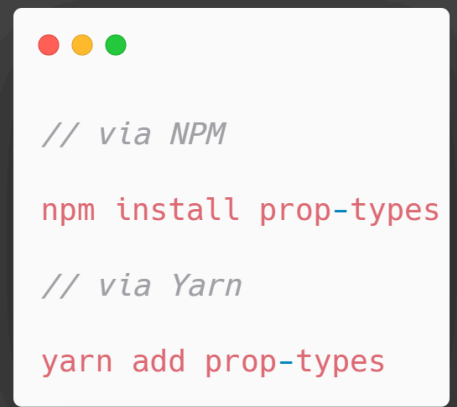
Node.js est utilisé pour exécuter des scripts de build qui préparent votre application React pour le déploiement en production, en optimisant et en regroupant les ressources.

Bien que Node.js ne soit pas nécessaire pour servir une application React en production (car React génère du HTML/CSS/JS statique), il est souvent utilisé dans les processus de build et de déploiement automatisés.

## Prop Types

PropTypes est une bibliothèque pour la validation de types en React. Elle est utilisée pour s'assurer que les composants reçoivent les bons types de props (propriétés). C'est un outil important pour la robustesse et la maintenabilité des applications React, particulièrement dans les projets de grande envergure ou en équipe.

Avant d'utiliser PropTypes, vous devez l'installer dans votre projet React :

A terminal window with a dark background and a title bar with three colored circles (red, yellow, green). It displays two sets of commands for installing PropTypes. The first set is for NPM, showing the command 'npm install prop-types' in red text. The second set is for Yarn, showing the command 'yarn add prop-types' in red text. The comments '// via NPM' and '// via Yarn' are in light gray.

```
// via NPM
npm install prop-types

// via Yarn
yarn add prop-types
```

Dans votre fichier de composant React, importez PropTypes :



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

Après avoir importé PropTypes, vous pouvez les définir pour un composant. Voici comment vous le faites :



```
class MyComponent extends React.Component {  
  // ...  
}  
  
MyComponent.propTypes = {  
  name: PropTypes.string,  
  age: PropTypes.number,  
  isStudent: PropTypes.bool,  
  subjects: PropTypes.arrayOf(PropTypes.string),  
  onGraduate: PropTypes.func  
};
```

## Exemples d'utilisation standard :

### Type string :

```

// Pour un string obligatoire

MyComponent.propTypes = {
  name: PropTypes.string.isRequired
};
```

### Exemple de types composés et obligatoires :

```

// Pour un tableau de chaînes de caractères :

MyComponent.propTypes = {
  subjects: PropTypes.arrayOf(PropTypes.string)
};

//Pour une fonction et la rendre obligatoire :

MyComponent.propTypes = {
  onGraduate: PropTypes.func.isRequired
};
```

Les PropTypes aident à prévenir de nombreux bugs en s'assurant que les composants reçoivent les bons types de données.

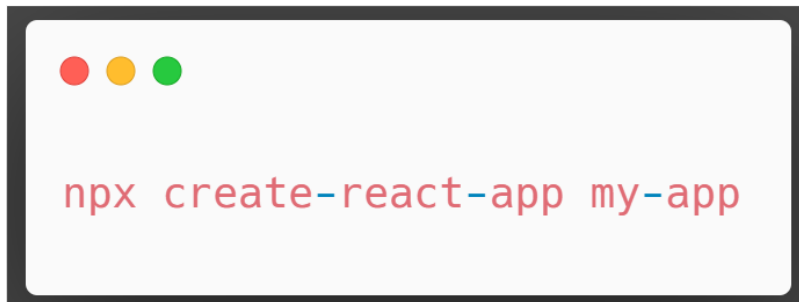
Ils servent également de forme de documentation, indiquant clairement quels types de props un composant attend.

En cas de type incorrect, React avertit le développeur via la console, facilitant ainsi le débogage.

# Installation et Configuration de React

Assurez-vous que Node.js et NPM sont installés.

Créez une nouvelle application React avec `npx create-react-app my-app`.



**npx** est un outil de gestion de paquets qui fait partie de NPM.

**create-react-app** est un outil qui configure automatiquement un nouveau projet React avec les bonnes pratiques.

**my-app** est le nom de votre nouvelle application.

Une fois le processus terminé, vous aurez un nouveau dossier appelé my-app avec la structure de projet suivante :

node\_modules/ :

Contient tous les modules NPM que votre projet utilise.

public/ :

- index.html : Le fichier HTML principal de votre application. React injectera dynamiquement les composants dans ce fichier.
- favicon.ico : L'icône de votre application qui apparaît dans l'onglet du navigateur.
- manifest.json : Un fichier de configuration pour les Progressive Web Apps (PWA), il définit l'apparence et le comportement de votre application lorsqu'elle est ajoutée à l'écran d'accueil d'un appareil mobile.
- 

src/ :

App.js : Le composant principal de votre application React.

App.css : Le fichier CSS pour le style de votre composant App.

index.js : Le point d'entrée JavaScript pour votre application. Il utilise ReactDOM pour rendre votre application dans le DOM.

index.css : Le fichier CSS global pour votre application.

reportWebVitals.js : Un script pour mesurer les performances de votre application

.gitignore : Liste des fichiers et dossiers à ignorer dans les commits Git.

package.json : Contient les métadonnées de votre projet, ainsi que la liste des dépendances et scripts utilisables.


README.md : Un fichier Markdown contenant des informations de base sur l'utilisation de Create React App.

Dans index.js, vous trouverez une ligne d'importation pour react-dom.

ReactDOM est importé pour utiliser sa méthode render, qui est essentielle pour injecter votre application React dans le DOM.

ReactDOM.render prend deux arguments : le composant racine (généralement <App />) et l'élément DOM dans lequel le composant doit être rendu (généralement un élément avec l'ID root dans index.html).

Exemple dans index.js :

A code editor window with a dark border and three colored window control buttons (red, yellow, green) in the top-left corner. The editor contains the following JavaScript code:

```
import ReactDOM from 'react-dom';  
import App from './App';  
  
ReactDOM.render(  
  <React.StrictMode>  
    <App />  
  </React.StrictMode>,  
  document.getElementById( 'root' )  
)
```



## ReactDOM

React-dom est la bibliothèque qui permet à React d'interagir avec le DOM dans un environnement de navigateur. Elle fournit des méthodes pour rendre les composants React dans le DOM et pour mettre à jour l'interface utilisateur en réponse aux changements d'état et de données.

## Les Composants React

Les composants sont les éléments de base de toute application React. Ils encapsulent le rendu et le comportement de parties de l'interface utilisateur.

Il existe différents types de Composants:

Composants Fonctionnels: Ils sont définis par des fonctions JavaScript simples et sont souvent utilisés pour les **composants sans état**, tout du moins initialement.

```
function Welcome(props) {  
  return <h1>Bonjour, {props.name}</h1>;  
}  
  
export default Welcome;  
  
// Utilisation ailleurs dans l'application  
// import Welcome from './Welcome';  
// <Welcome name="Alice" />
```

Composants de Classe: Ce sont des classes ES6 avec des fonctionnalités étendues, telles que le maintien d'un état interne et l'accès **aux méthodes de cycle de vie**.

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

class Welcome extends Component {
  render() {
    return <h1>Bonjour, {this.props.name}!</h1>;
  }
}

export default Welcome;

// Utilisation ailleurs dans l'application
// import Welcome from './Welcome';
// <Welcome name="Alice" />
```

### Composants Fonctionnels Avant les Hooks :

Initialement, les composants fonctionnels en React étaient utilisés pour des composants "sans état" (stateless), c'est-à-dire des composants qui n'avaient pas besoin de gérer un état interne ou d'utiliser les méthodes de cycle de vie. Ces composants étaient simplement des fonctions qui acceptaient des props et retournaient du JSX.

Les composants de classe, d'autre part, étaient utilisés pour des composants "avec état" (stateful). Ils utilisaient les fonctionnalités des classes ES6 pour maintenir un état interne (this.state et this.setState) et avaient accès aux méthodes de cycle de vie (comme componentDidMount, componentDidUpdate, etc.).

Avec l'introduction des Hooks dans React 16.8, les composants fonctionnels ont acquis la capacité de gérer un état interne et d'utiliser d'autres caractéristiques de React qui étaient auparavant exclusives aux composants de classe.

**useState:** Un Hook qui permet aux composants fonctionnels de maintenir un état interne.

**useEffect:** Un Hook qui permet d'effectuer des opérations de cycle de vie dans les composants fonctionnels. Exemple :

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

function Compteur() {
  const [compte, setCompte] = useState(0);

  const incrementer = () => {
    setCompte(compte + 1);
  };

  return (
    <div>
      <p>Compte: {compte}</p>
      <button onClick={incrementer}>Incrementer</button>
    </div>
  );
}
```

## Les Hooks

Les Hooks sont une des fonctionnalités les plus révolutionnaires et puissantes introduites dans React 16.8. Ils permettent d'utiliser l'état, les effets secondaires, et d'autres fonctionnalités de React dans les composants fonctionnels sans avoir recours à des classes. Cela représente un changement fondamental dans la façon dont les développeurs peuvent composer et réutiliser la logique dans leurs applications React.

Avant les Hooks, les composants fonctionnels étaient limités car ils ne pouvaient pas gérer l'état interne ni accéder aux cycles de vie du composant. Les Hooks répondent à ce besoin en offrant une manière élégante et efficace de gérer l'état, les effets, et bien plus dans les composants fonctionnels.

### Les hook "useState" et "useEffect":

Il existe d'autres types de hook mais ces deux là sont les plus courants et il est important de bien les comprendre avant d'appréhender les autres hook !

### useState :

Le Hook useState est utilisé pour ajouter un état local à un composant fonctionnel. Chaque appel à useState retourne une paire de valeurs : l'état actuel et une fonction qui permet de mettre à jour cet état.

```
const [count, setCount] = useState(0);
```

Ici, count est la variable d'état, et setCount est la fonction pour mettre à jour cette variable. useState(0) initialise l'état avec la valeur 0.

### useEffect :

Le Hook useEffect est utilisé pour effectuer des opérations qui seraient normalement effectuées dans les méthodes de cycle de vie des composants de classe, telles que componentDidMount, componentDidUpdate, et componentWillUnmount.

```
useEffect(() => {  
  // Code exécuté après chaque rendering  
  document.title = `Vous avez cliqué ${count} fois`;  
  
  return () => {  
    // Nettoyage, équivalent à componentWillUnmount  
  };  
}, [count]); // Exécuté seulement si 'count' change
```

Ce Hook prend deux paramètres : une fonction à exécuter et un tableau de dépendances. La fonction peut retourner une autre fonction qui sert à nettoyer (par exemple, supprimer un abonnement).

### **Règles d'utilisation des Hooks :**

Uniquement au Niveau Racine : Les Hooks doivent être appelés au niveau le plus haut de votre composant. Ne les placez pas dans des boucles, des conditions, ou des fonctions imbriquées.

Seulement dans des Fonctions React: Utilisez les Hooks uniquement dans des composants fonctionnels ou dans des fonctions personnalisées de Hook.

# Gestion des Événements et Formulaires en React

En React, les événements sont nommés en utilisant la convention camelCase plutôt que la notation en minuscules utilisée en HTML pur.

```
<button onClick={handleClick}>Cliquez ici</button>
```

## Passer une Fonction, Pas un Appel de Fonction :

Lorsque vous attribuez un gestionnaire d'événement, vous devez passer une fonction et non l'appel d'une fonction.

Correct : `onClick={handleClick}`

Incorrect : `onClick={handleClick()}`

React encapsule l'événement natif du navigateur dans un objet `SyntheticEvent` pour assurer la compatibilité entre les navigateurs.

`event.preventDefault()` est couramment utilisé pour empêcher le comportement par défaut (par exemple, pour empêcher la soumission d'un formulaire de recharger la page).

Dans un composant, la valeur d'entrée du formulaire est contrôlée par l'état React du composant.

```
<input type="text" value={this.state.value} onChange={this.handleChange} />
```

Pour gérer la soumission d'un formulaire, utilisez `onSubmit` sur la balise `<form>` et définissez une méthode pour traiter les données du formulaire.

```
<form onSubmit={this.handleSubmit}>
  <button type="submit">Envoyer</button>
</form>
```

Voici un exemple de création de formulaire au complet :

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

function InscriptionForm() {
  const [nom, setNom] = useState('');
  const [email, setEmail] = useState('');
  const [errors, setErrors] = useState([]);

  const handleChange = (event) => {
    const { name, value } = event.target;

    if (name === 'nom') setNom(value);
    if (name === 'email') setEmail(value);
  };

  const validerForm = () => {
    let errors = [];

    if (!email.includes('@')) {
      errors.push("Email invalide");
    }

    // Autres validations...

    setErrors(errors);

    return errors.length === 0;
  };

  const handleSubmit = (event) => {
    event.preventDefault();

    if (validerForm()) {
      console.log('Form soumis:', { nom, email });

      // Traitement des données...
    }
  };
}
```

```

return (
  <form onSubmit={handleSubmit}>
    <input
      name="nom"
      type="text"
      value={nom}
      onChange={handleChange}
      placeholder="Nom"
    />
    <input
      name="email"
      type="email"
      value={email}
      onChange={handleChange}
      placeholder="Email"
    />
    <button type="submit">Inscription</button>
    {errors.length > 0 && (
      <div>{errors.join(", ")}</div>
    )}
  </form>
);
}

export default InscriptionForm;

```

Ici **useState** est utilisé pour créer des variables d'état (nom, email, errors) pour stocker les données du formulaire et les messages d'erreur.



**handleChange** est une fonction unique pour gérer les changements sur les champs de saisie. Elle met à jour l'état correspondant basé sur le name de l'élément `<input>`.

**validerForm** est une fonction pour valider les données du formulaire. Elle met à jour l'état `errors` si des erreurs sont détectées.

**handleSubmit** est la fonction qui est appelée lorsque le formulaire est soumis. Elle empêche le rechargement par défaut de la page, valide le formulaire et, si tout est correct, traite les données.

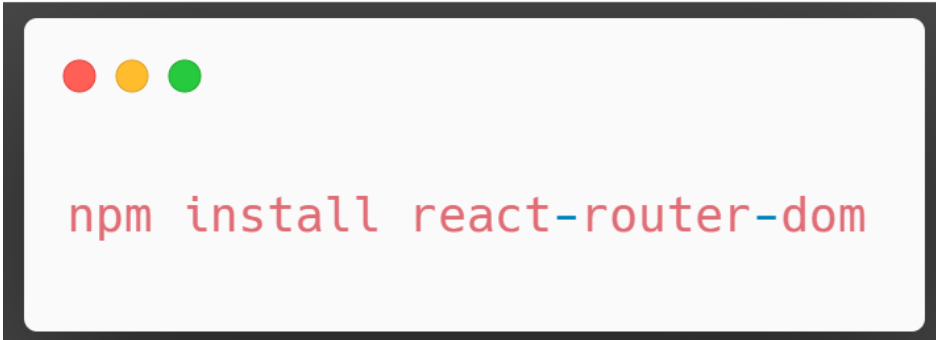
Des modules existe pour faciliter la gestion des formulaires, vous pouvez consulter par exemple Formik (<https://formik.org/docs/overview>)

## Routage avec React Router

React Router est une bibliothèque de routage standard pour React qui permet de gérer la navigation dans une application React. Elle offre une manière dynamique de rendre des composants basés sur l'URL du navigateur, simulant ainsi le comportement d'une application multi-pages.

### Configuration de React Router

Pour utiliser React Router, commencez par l'installer dans votre projet React :

A terminal window with a dark border and a light gray background. In the top-left corner, there are three colored circles: red, yellow, and green. The command `npm install react-router-dom` is displayed in a red monospace font.

```
npm install react-router-dom
```

`react-router-dom` est la version de React Router utilisée pour les applications web.

Importez **BrowserRouter** dans votre fichier principal (souvent `index.js` ou `App.js`) et utilisez-le pour envelopper votre application. Il s'occupe de gérer le routage dans l'application.

Exemple dans App.js :

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from 'react-router-dom';

function App() {
  return (
    <BrowserRouter>
      <App/>
      {/* Les autres composants de votre app viennent ici */}
    </BrowserRouter>
  );
}
```

**Route** est utilisée pour déclarer une route individuelle et le composant qui doit être rendu.

**Routes** est utilisé pour grouper plusieurs routes et s'assure que seulement le premier <Route> qui correspond à l'URL actuelle est rendu.

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/about" element={<About />} />
  <Route path="/contact" element={<Contact />} />
</Routes>
```

## Navigation avec <Link>

Le composant **Link** de la bibliothèque react-router-dom, est essentiel pour créer des liens de navigation dans une application React utilisant React Router



```
import { Link } from 'react-router-dom';  
  
<Link to="/about">À Propos</Link>
```

Lorsqu'un utilisateur clique sur un Link, React Router change l'URL dans la barre d'adresse du navigateur et rend le composant associé à cette URL sans recharger la page entière. Cela permet une expérience utilisateur plus rapide et plus fluide.

La propriété `to` du composant Link spécifie la destination du lien. Elle peut être une simple chaîne de caractères représentant le chemin (comme dans `/about`) ou un objet contenant des informations plus détaillées sur la destination.

Contrairement à une balise `<a>` HTML classique qui provoque un rechargement complet de la page, Link permet de naviguer entre les composants de l'application sans rechargement, ce qui améliore les performances et l'expérience utilisateur.

Link fonctionne en harmonie avec le système de routage de React Router, assurant que les transitions entre les routes sont gérées de manière cohérente et selon les meilleures pratiques de React.

# React et Bootstrap

L'utilisation de Bootstrap avec React est une pratique courante pour développer des interfaces utilisateur réactives et esthétiquement agréables.

Pour intégrer Bootstrap dans une application React, **vous pouvez soit inclure les fichiers CSS et JavaScript de Bootstrap directement dans votre projet**, soit utiliser des packages spéciaux comme **React-Bootstrap** ou **Reactstrap**, qui encapsulent les composants Bootstrap sous forme de composants React.

Pour l'intégrer manuellement vous pouvez inclure les fichiers CSS et JS de Bootstrap dans le fichier index.html de votre projet React. Il existe également d'autres méthodes pour importer bootstrap.

Exemple dans index.html situé dans le dossier public :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta name="description" content="Web site created using create-react-app"/>
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>Demo React</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-C6RzsynM9kWDrmNeT87bh950GnyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL"
crossorigin="anonymous"></script>
  </body>
</html>
```

Voici quelques unes des classes les plus utiles de bootstrap :

### Boutons :



```
<button className="btn btn-primary">Cliquez-moi</button>
```

### Système de Grille :



```
<div className="container">
  <div className="row">
    <div className="col-md-8">Contenu principal</div>
    <div className="col-md-4">Barre latérale</div>
  </div>
</div>
```

### Navbar :



```
<nav className="navbar navbar-expand-lg navbar-light bg-light">
  <a className="navbar-brand" href="#">MonApp</a>
  { /* Autres éléments de la navbar */ }
</nav>
```

## Formulaires :



```
<div className="form-group">  
  <label>Email</label>  
  <input type="email" className="form-control" placeholder="Email" />  
</div>
```

## Cartes (Cards) :



```
<div className="card">  
  <div className="card-body">  
    <h5 className="card-title">Titre de la carte</h5>  
    <p className="card-text">Contenu de la carte.</p>  
  </div>  
</div>
```

# React Context

React Context est un mécanisme qui permet de passer des données à travers l'arbre de composants d'une application React sans avoir à passer explicitement les données via les props à chaque niveau. Cela résout le problème de "prop drilling" où les données doivent être transmises à travers plusieurs niveaux de composants.

React Context est particulièrement utile lorsque l'on a des données qui sont nécessaires par de nombreux composants dans l'application, comme le thème de l'interface utilisateur, les préférences de l'utilisateur, ou les données d'authentification.

Pour commencer à utiliser React Context, on doit d'abord créer un Context. Un Context est créé en utilisant la fonction `React.createContext()`.

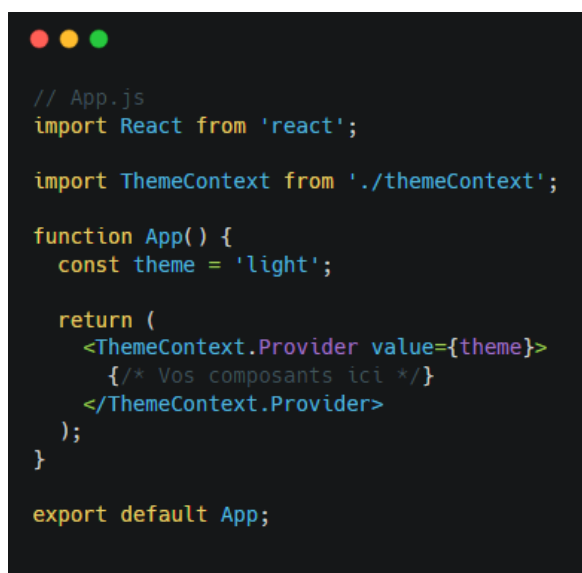


```
// themeContext.js
import React from 'react';

const ThemeContext = React.createContext();

export default ThemeContext;
```

Une fois que l'on a créé le Context, on peut fournir une valeur à ce Context en utilisant un composant `Provider`. Le composant `Provider` est un composant fourni par React qui permet de fournir des données à tous ses composants enfants. Il est souvent utilisé en combinaison avec le contexte pour fournir des données globales à travers l'application, sans avoir besoin de les transmettre explicitement via les props.



```
// App.js
import React from 'react';

import ThemeContext from './themeContext';

function App() {
  const theme = 'light';

  return (
    <ThemeContext.Provider value={theme}>
      {/* Vos composants ici */}
    </ThemeContext.Provider>
  );
}

export default App;
```

Une fois que l'on a fourni une valeur au Context, on peut y accéder à n'importe quel niveau de l'arbre de composants en utilisant un composant Consumer ou le hook useContext.

Avec un composant Consumer :

```
// ExampleComponent.js
import React from 'react';
import ThemeContext from './themeContext';

function ExampleComponent() {
  return (
    <ThemeContext.Consumer>
      {theme => (
        <div style={{ background: theme === 'light' ? '#fff' : '#000' }}>
          {/* Contenu ici */}
        </div>
      )}
    </ThemeContext.Consumer>
  );
}

export default ExampleComponent;
```

Avec le hook useContext :

```
// ExampleComponent.js
import React, { useContext } from 'react';
import ThemeContext from './themeContext';

function ExampleComponent() {
  const theme = useContext(ThemeContext);

  return (
    <div style={{ background: theme === 'light' ? '#fff' : '#000' }}>
      {/* Contenu ici */}
    </div>
  );
}

export default ExampleComponent;
```



Si l'on souhaite permettre à des composants enfants de modifier les valeurs d'un contexte et de faire en sorte que ces modifications soient reflétées dans d'autres composants, nous devons créer des fonctions ou des mécanismes dans le contexte lui-même qui permettent de mettre à jour ses valeurs de manière globale.

Par exemple, dans un contexte de gestion du thème d'une application, nous pourrions avoir une fonction `toggleTheme` qui permet aux composants enfants de basculer entre les thèmes 'clair' et 'sombre'. Lorsque cette fonction est appelée par un composant enfant, elle met à jour le thème dans le contexte, et tous les autres composants qui consomment ce contexte seront automatiquement mis à jour pour refléter le nouveau thème.

```
// ThemeContext.js
import React, { createContext, useState } from 'react';

const ThemeContext = createContext();

const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState('light');

  const toggleTheme = () => {
    setTheme(prevTheme => (prevTheme === 'light' ? 'dark' : 'light'));
  };

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};

export { ThemeProvider, ThemeContext };
```

Dans cet exemple, le composant `ThemeProvider` fournit un contexte de thème à ses composants enfants. Il expose une fonction `toggleTheme` qui permet de basculer entre les thèmes 'clair' et 'sombre'. Lorsqu'un composant enfant appelle `toggleTheme`, le thème est mis à jour dans le contexte, et tous les autres composants qui consomment ce contexte sont automatiquement mis à jour pour refléter le nouveau thème.

En conclusion, React Context est un outil puissant pour gérer l'état global dans une application React et pour éviter la propagation excessive des props.

# Atelier pratique

Ces projets permettent de consolider les compétences en React, couvrant des aspects essentiels tels que la gestion de l'état, le routage et l'interaction utilisateur.

## Premier sujet : Site E-commerce

Description : Création d'un site e-commerce basique en React. Le site permet aux utilisateurs de parcourir des produits, les ajouter à un panier, et passer une commande.

Vous pourrez alimenter votre site en utilisant l'url : "<https://fakestoreapi.com/products>"

### Réalisation Attendue :

Catalogue de produits : Afficher une liste de produits avec des descriptions et des images.

Fonctionnalité de Panier : Ajouter et retirer des produits du panier, et voir le total de la commande.

Page de Détail des Produits : Vue détaillée de chaque produit lorsque l'utilisateur clique sur un produit dans le catalogue.

Formulaire de commande : Formulaire pour saisir les informations de livraison et passer la commande.

Navigation : Utilisation de React Router pour naviguer entre différentes pages (accueil, détails du produit, panier).

Gestion de l'État : Utiliser les Hooks de React pour gérer l'état du panier et des informations du formulaire.

**Les informations ci-dessus sont des informations "brute", à vous d'implémenter correctement toutes ces features en bonne intelligence.**

## Deuxième sujet : Memory Game

Description : Développement d'un jeu de mémoire où les utilisateurs doivent trouver des paires de cartes identiques. Le jeu inclura plusieurs niveaux de difficulté.

### Réalisation Attendue :

Grille de Jeu : Interface où les cartes sont disposées et peuvent être retournées par les utilisateurs.

Logique du jeu : Mécanisme pour gérer le retournement des cartes, vérifier les correspondances et mémoriser les paires trouvées.

Score et Chronomètre : Afficher le score basé sur le nombre de tentatives et un chronomètre pour suivre le temps écoulé.

Niveaux de difficulté : Options pour jouer à différents niveaux (par exemple, facile, moyen, difficile), avec un nombre de cartes variable.

État du Jeu : Utilisation des Hooks de React pour gérer l'état du jeu, y compris le suivi des cartes retournées et des paires trouvées.

Animations et Feedback Visuel : Effets visuels pour rendre le jeu plus interactif et engageant.

**Les informations ci-dessus sont des informations “brute”, à vous d’implémenter correctement toutes ces features en bonne intelligence.**