



DÉVELOPPEMENT REACT DANS LARAVEL

Présentation de la formation

Client

Server



User

Interacts with the system in browser

Frontend

Client app runs on any device in browser

Backend

Business logic, web services and database

3rd party service

3rd party service

Objectif de la formation

- Comprendre React en tant que framework.
- Acquérir la capacité d'utiliser React avec Laravel.

Contenu de la formation

Initiation au Développement avec React

Intégration du React avec Laravel

1. "Concepts Fondamentaux de React"
2. "React Router : Navigation dans les Applications React"
3. "Gestion de l'État avec React (state management)"

PARTIE 1

C'EST QUOI REACT ?

- React est une puissante bibliothèque JavaScript qui améliore l'interactivité et le dynamisme des sites Web. Il agit comme un superpouvoir, augmentant les capacités de HTML, CSS et JavaScript traditionnels.
- un framework est un ensemble d'outils ultra complets permettant de créer une application de A à Z et fournissant tous les outils nécessaires au développement d'une application. Alors qu'une bibliothèque s'ajoute à une partie de votre application.

Avantages de React

- **Réactivité** : Permet la création de sites Web hautement réactifs qui réagissent instantanément aux interactions avec les utilisateurs.
- **Composabilité** : Permet la construction de sites Web comme des ensembles LEGO, en utilisant de petits blocs de construction (composants) pour créer facilement des structures Web complexes.
- **Réutilisabilité** : Permet la réutilisation des éléments de base (composants) pour développer efficacement diverses parties d'un site Web.
- **Performances** : Améliore la vitesse du site Web en mettant à jour uniquement les pièces nécessaires, garantissant une expérience utilisateur rapide et fluide.

Avantages de l'utilisation de React

- **Largement utilisé** : Adopté par une vaste communauté de développeurs dans le monde entier, offrant d'abondantes opportunités de carrière.
- Favorisé pour accélérer et améliorer les processus de développement Web.
- **Documentation** : Appuyée par des instructions et des exemples clairs, ce qui simplifie la courbe d'apprentissage.
- **Fonctions de développement Web modernes** : Offre un accès à des fonctions de développement Web de pointe, ce qui rehausse l'apparence des sites Web à un niveau moderne et professionnel.

REACT SYNTAX ET JSX

JAVASCRIPT XML

```
mirror_mod = modifier_obj.mirror_mod
# set mirror object to mirror
mirror_mod.mirror_object = active_object
if operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
if operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
if operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

selection at the end -add
mirror_mod.select= 1
modifier.select=1
bpy.context.scene.objects.active = bpy.context.selected_objects[0]
bpy.context.selected_objects[0].name
bpy.data.objects[one.name].select = 1
int("please select exactly one object")
-- OPERATOR CLASSES --
types.Operator:
    # X mirror to the selected object.mirror_mirror_x"
    "mirror X"
context):
    # context.active_object is not None
```

Comprendre JSX

JSX, abréviation de JavaScript XML, sert d'extension de syntaxe pour JavaScript, permettant l'intégration transparente de code de type HTML dans les fichiers JavaScript. Cela simplifie la définition de la structure des composants React en fusionnant la syntaxe HTML et JavaScript.

Code JSX minimum pour "Hello, World"

Exploration du code JSX minimal requis pour afficher un message « Hello, world! » dans un composant React :

```
import React from 'react';
import ReactDOM from 'react-dom';
function HelloWorld() {
  return <h1>Hello, world!</h1>;
}
```

- **Instructions d'importation** : Importez les bibliothèques React et ReactDOM nécessaires.
- **Déclaration de fonction** : Définissez un composant de fonction nommé HelloWorld.
- **Déclaration de retour** : Utilisez JSX pour retourner un élément contenant le texte "Hello, world!".
- **Rendu ReactDOM** : Rend le composant HelloWorld dans l'élément HTML avec la racine ID.

Prise en main de React

Configuration de votre environnement de développement

L'un des moyens les plus simples d'initier le développement de React est d'utiliser un réseau de diffusion de contenu (CDN) pour inclure directement React dans votre fichier HTML.

Modèle HTML pour React Development

Le modèle HTML fourni comprend les scripts React et ReactDOM nécessaires, ainsi que Babel pour compiler JSX dans le navigateur. Il définit également un composant React de base rendant un message "Hello, world!".

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Include React and ReactDOM scripts from CDN -->
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
      // Define and render a simple React component
    </script>
  </body>
</html>
```

- La page est un excellent moyen d'essayer React mais elle n'est pas adaptée à la production. Il compile lentement JSX avec Babel dans le navigateur et utilise une grande version de développement de React.
- Lisez cette page pour commencer un nouveau projet React avec JSX :

<https://react.dev/learn/start-a-new-react-project>

FRAGMENT DANS REACT

- Fragment fournit un moyen de regrouper plusieurs éléments dans un composant sans ajouter de nœuds supplémentaires au DOM. Ils ont été introduits pour répondre à la limitation de renvoyer plusieurs éléments de la méthode de rendu d'un composant.

```
import React from 'react';
function MyApp() {
  return (
    <React.Fragment>
      <h1>Hello!</h1>
      <p>This is a second DOM element</p>
    </React.Fragment>
  );
}
```

or using the shorthand syntax:

```
import React from 'react';
function MyApp() {
  return (
    <>
      <h1>Hello!</h1>
      <p>This is a second DOM element</p>
    </>
  );
}
```

NODE.JS, NPM ET OUTILS DE CONSTRUCTION



- **Node.js** : Environnement d'exécution JavaScript permettant l'exécution de code JavaScript en dehors d'un navigateur Web, essentiel pour les flux de travail de développement Web modernes.
- **npm (Node Package Manager)** : Gestionnaire de paquets par défaut pour Node.js, facilitant l'installation, la gestion et le partage des paquets JavaScript.
- **Outils de construction comme Vite** : Automatisez des tâches telles que transpiling, bundling et optimisation du code, Vite étant un outil de construction moderne idéal pour les projets React.

Utiliser npm pour gérer les dépendances

Avant de commencer un projet React, les développeurs utilisent généralement npm pour initialiser un nouveau projet et installer des dépendances essentielles, y compris React et d'autres bibliothèques ou outils requis.

Mise en place de Vite pour React Développement

Les développeurs peuvent utiliser Vite comme un outil de construction pour le développement React, en le configurant pour construire et servir leur application React efficacement.

Construire pour la production

Lorsque l'application React est prête pour la production, Vite peut être utilisé pour créer des paquets de code optimisés et réduits, générant des actifs statiques déployables sur des serveurs Web ou des réseaux de diffusion de contenu pour l'hébergement.

NOTIONS DE BASE DE JAVASCRIPT



Notions de base de JavaScript pour React

Opérations de synchronisation et d'asynchrone

Dans React, comprendre les opérations synchrones et asynchrones est crucial pour gérer des tâches telles que la récupération de données et la gestion des événements.

Exemple :

```
// Asynchronously fetch data from an API
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error.message));
```

Notions de base de JavaScript pour React

Arrow Fonctions

Les fonctions Arrow fournissent une syntaxe concise pour définir les fonctions et sont couramment utilisées dans React pour écrire des fonctions de rappel et des composants fonctionnels.

Exemple :

```
// Regular function
function greet(name) {
  return 'Hello, ' + name + '!';
}

// Arrow function
const greetArrow = (name) => {
  return 'Hello, ' + name + '!';
};

// Shortened arrow function
const greetShort = (name) => 'Hello, ' + name + '!';
```

Notions de base de JavaScript pour React

Exportation et importation

Les mots-clés d'exportation et d'importation sont utilisés pour organiser le code en modules réutilisables, une pratique essentielle pour maintenir une application React structurée.

Exemple :

```
// math.js
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;
// main.js
import { add, subtract } from './math.js';
console.log(add(5, 3));
console.log(subtract(7, 2));
```

LA DÉSTRUCTURATION EN

ES6

ES6 a introduit l'affectation de déstructuration, qui est un moyen pratique d'extraire plusieurs valeurs de tableaux ou d'objets et de les affecter à des variables.

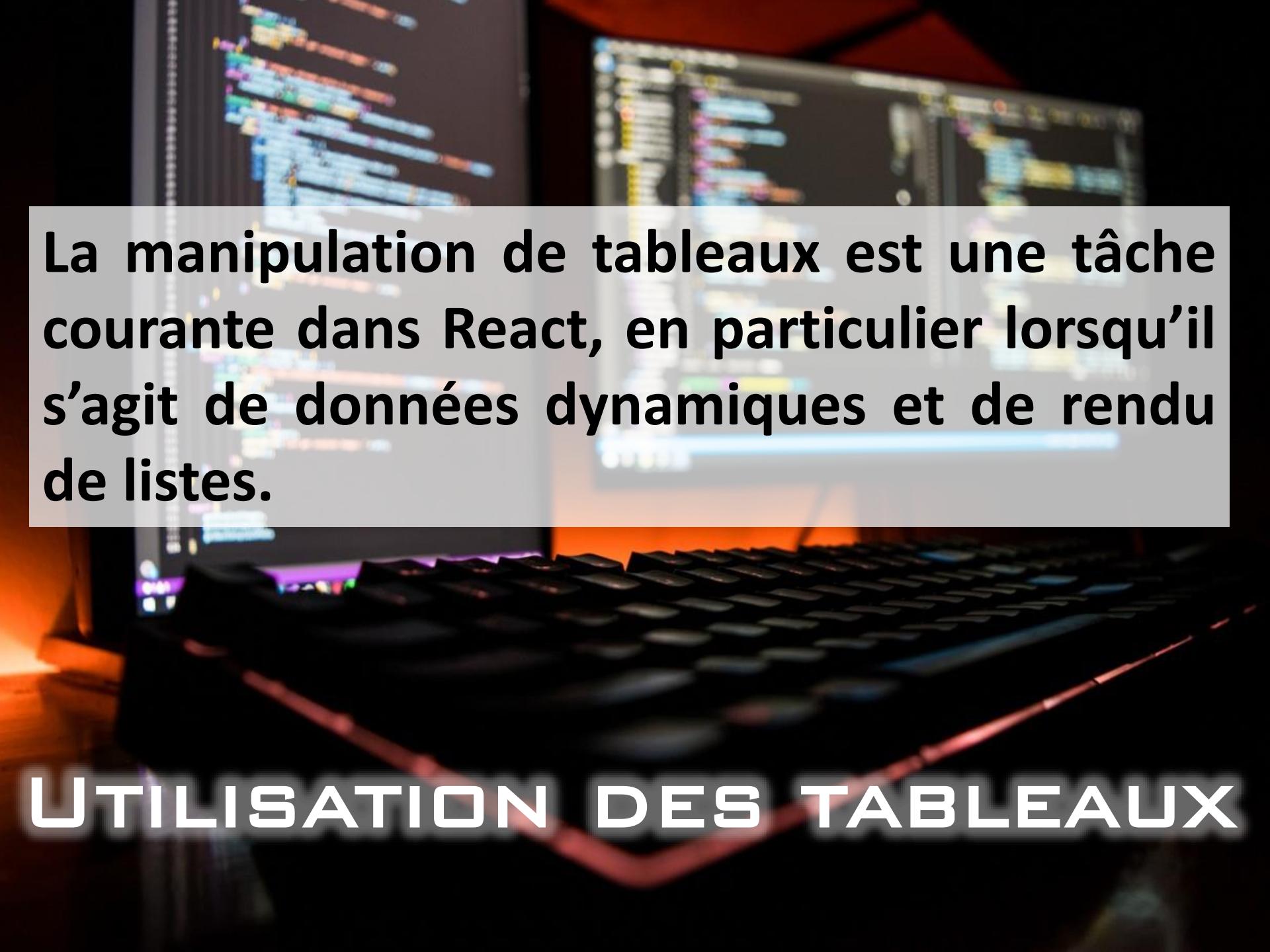
La Déstructuration en ES6

Destruction du tableau

```
const numbers = [1, 2, 3, 4, 5];
const [first, second, ...rest] = numbers;
console.log(first); // Output: 1
console.log(second); // Output: 2
console.log(rest); // Output: [3, 4, 5]
```

Destruction d'objets

```
const person = { name: 'John', age: 30, city: 'New York' };
const { name, age } = person;
console.log(name); // Output: John
console.log(age); // Output: 30
```



La manipulation de tableaux est une tâche courante dans React, en particulier lorsqu'il s'agit de données dynamiques et de rendu de listes.

UTILISATION DES TABLEAUX

Utilisation des tableaux

forEach : Exécute une fonction fournie une fois pour chaque élément de tableau.

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach((number) => {
  console.log(number);
});
```

map : Crée une nouvelle table en appliquant une fonction à chaque élément de la table d'origine.

```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map((number) => {
  return number * 2;
});
```

Utilisation des tableaux

filter : Crée un nouveau tableau avec des éléments qui passent un test spécifié par une fonction.

```
const numbers = [1, 2, 3, 4, 5];
const evenNumbers = numbers.filter((number) => {
  return number % 2 === 0;
});
```

reduce : Exécute une fonction réducteur sur chaque élément du tableau, ce qui donne une valeur de sortie unique.

```
const numbers = [1, 2, 3, 4, 5];
const sum = numbers.reduce((accumulator, currentValue) => {
  return accumulator + currentValue;
}, 0);
```

Utilisation des tableaux

find : Retourne le premier élément du tableau qui satisfait la fonction de test fournie.

```
const numbers = [1, 2, 3, 4, 5];
const found = numbers.find((number) => {
  return number > 3;
});
```

indexOf : Retourne le premier index auquel un élément donné peut être trouvé dans le tableau, ou -1 s'il n'est pas présent.

```
const numbers = [1, 2, 3, 4, 5];
const index = numbers.indexOf(3); // index is 2
```

Utilisation des tableaux

includes : Détermine si un tableau inclut un certain élément, retournant vrai ou faux selon le cas.

```
const numbers = [1, 2, 3, 4, 5];
const isIncluded = numbers.includes(3); // isIncluded is true
```

slice : Retourne une copie superficielle d'une partie d'un tableau dans un nouvel objet de tableau.

```
const numbers = [1, 2, 3, 4, 5];
const sliced = numbers.slice(2, 4); // sliced is [3, 4]
```

Références

Documentation de React

La documentation officielle de React fournit des conseils complets sur les fonctionnalités de React, y compris le cycle de vie des composants, la gestion des états, les hooks et le routage.

- <https://react.dev/>
- <https://www.w3schools.com/react/>
- <https://vitejs.dev/>

FIN DE PARTIE 1

```
        _mod = modifier_ob.modifiers.new("MIRROR")
        _mod.object_to_mirror_ob = mirror_ob
        _mod.mirror_object = mirror_ob
    if operation == "MIRROR_X":
        mod.use_x = True
        mod.use_y = False
        mod.use_z = False
    if operation == "MIRROR_Y":
        mod.use_x = False
        mod.use_y = True
        mod.use_z = False
    if operation == "MIRROR_Z":
        mod.use_x = False
        mod.use_y = False
        mod.use_z = True
    #selection at the end - add back the else
    ob.select= 1
    mirror.ob.select=1
    context.scene.objects.active = modifier
    print("selected" + str(modifier_ob)) # modifier
    mirror_ob.select = 0
    for key,context.selected_objects[0]
    context.objects[one].select = 1
    print("please select exactly two objects")
    #OPERATOR CLASSES -->
#operator classes
def execute(self, context):
    """mirror to the selected object"""
    if mirror.mirror_mirror_x:
        mirror X
    if context:
        if context.active_object is not None
```



Chapitre 1 : Concepts Fondamentaux de React

1. Introduction

2. Modélisation avec React (React Templating)

3. Concepts de Base de React



React offre une solution robuste pour le stylisme et la modélisation. Sa structure basée sur les composants permet la création d'éléments d'interface utilisateur réutilisables, favorisant la cohérence et l'efficacité dans la conception et le développement.

STYLING ET TEMPLATING DANS REACT

UTILISATION DES VARIABLES DANS LES MODÈLES REACT

Dans React, les variables jouent un rôle crucial dans la définition dynamique des valeurs des attributs et le rendu du contenu dynamique dans les modèles. Cela permet aux développeurs de créer des interfaces utilisateur plus flexibles et interactives.

Exemple :

```
import React from 'react';
const App = () => {
  // Define a variable to hold dynamic content
  const message = 'Hello, World!';
  return (
    <div>
      /* Utilize the variable to render dynamic content */
      <h1>{message}</h1>
    </div>
  );
}
export default App;
```

Dans cet exemple, la variable message est utilisée pour maintenir le contenu dynamique "Hello, World!" et est ensuite rendue dans l'élément h1. Cela montre comment les variables peuvent être utilisées pour définir dynamiquement le contenu dans les modèles React..

Valeurs d'attribut

```
const App = () => {
  // Define variables to hold attribute values
  const imageUrl = 'path_to_image.jpg';
  const linkUrl = 'https://www.example.com';
  return (
    <div>
      /* Utilize variables to dynamically set attribute values */
      <img src={imageUrl} alt="Dynamic Image" />
      <a href={linkUrl}>Dynamic Link</a>
    </div>
  );
}
export default App;
```

Les variables **imageUrl** et **linkUrl** sont utilisées pour définir dynamiquement les attributs `src` et `href`, respectivement, des éléments **img** et **a**

STYLES EN REACT

Styliser un composant React peut être réalisé en utilisant diverses méthodes telles que des styles en ligne ou des feuilles de style CSS, chaque méthode a ses propres avantages et cas d'utilisation.



Styles intégrés :

Les styles en ligne impliquent l'application directe de styles à des éléments individuels à l'aide de l'attribut style dans JSX.

Pour appliquer des styles intégrés à un composant React, procédez comme suit :

1. Définissez les styles comme un objet JavaScript.
2. Utilisez l'attribut style pour appliquer les styles au composant.

Exemple :

```
const InlineStyledComponent = () => {
  const styles = {
    backgroundColor: 'lightblue',
    padding: '20px',
    fontSize: '18px',
  };
  return <div style={styles}>This is a component with inline
  styles</div>;
};
export default InlineStyledComponent;
```

Feuilles de style CSS : L'utilisation de feuilles de style CSS implique la création de fichiers css séparés et leur importation dans vos composants React.
Pour styliser un composant React à l'aide de feuilles de style CSS, procédez comme suit :

- 1. Créez un fichier .css avec vos styles.**
- 2. Importez le fichier css dans votre composant React.**

Exemple :

```
// styles.css
.container {
  background-color: lightblue;
  padding: 20px;
  font-size: 18px;
}

// CSSStyledComponent.jsx
import React from 'react';
import './styles.css';
const CSSStyledComponent = () => {
  return <div className="container">This is a component styled using
CSS stylesheet</div>;
};
export default CSSStyledComponent;
```

INCLURE LE FRAMEWORK CSS BOOTSTRAP DANS UN PROJET REACT :

Pour intégrer le framework CSS Bootstrap dans un projet React, le paquet du framework doit être installé et son fichier CSS importé. Cela facilite l'utilisation des composants pré-conçus de Bootstrap et du système de mise en page réactif dans l'application React.

L'intégration de Bootstrap dans un projet React implique quelques étapes clés pour assurer une intégration transparente du framework CSS Bootstrap. Voici un guide détaillé sur la façon d'y parvenir :

Installation

1. Install Bootstrap Package : Commencez par installer le package Bootstrap via npm en exécutant la commande suivante dans le terminal :

```
npm install bootstrap
```

Importation

2. Importer le CSS bootstrap : Dans le composant React où des styles bootstrap sont nécessaires, importer le fichier CSS bootstrap. Pour ce faire, ajoutez l'instruction d'importation suivante en haut du fichier de composants :

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

Usage

3. Utiliser les classes Bootstrap : Une fois Bootstrap importé, vous pouvez directement utiliser les classes prédéfinies de Bootstrap dans vos composants React pour exploiter son système de grille, ses composants et ses classes utilitaires. Intégrer le framework CSS Bootstrap dans le projet React, permettant de tirer parti de ses composants pré-conçus et de son système de mise en page réactif.

RENDU DE LISTE DANS REACT

- Le rendu de liste dans React implique un rendu dynamique des listes d'éléments, offrant un moyen transparent d'afficher les données à partir de tableaux ou d'autres sources d'itération. Cela permet la création d'interfaces utilisateur dynamiques et évolutives.
- La création de modèles dans React implique l'utilisation d'accolades {} pour incorporer des expressions JavaScript dans JSX, permettant un rendu de contenu dynamique. comme itération sur une liste et créer un nouveau tableau d'éléments React en utilisant la fonction map.

```
const ListComponent = () => {
  const items = ["Apple", "Banana", "Orange", "Mango"];

  return (
    <div>
      <h2>Items</h2>
      <ul>
        {items.map((item, index) => (
          <li key={index}>{item}</li>
        )));
      </ul>
    </div>
  );
}

export default ListComponent;
```

Importer et exporter des composants dans React:

Le processus d'importation et d'exportation des composants dans React est essentiel pour améliorer l'organisation et la réutilisation du code. Cette approche permet la création de composants modulaires, maintenables et facilement partageables dans une application React.

Syntaxe d'exportation :

```
const OtherComponent = () => {
  return <div>Other Component</div>;
}

export default OtherComponent;
```

Syntaxe d'importation :

```
import React from 'react';

import OtherComponent from './OtherComponent';
```

USESTATE HOOK DANS REACT

Pour Dans React, le hook useState est une fonctionnalité fondamentale qui permet aux composants fonctionnels de gérer state.

State représente les données qui peuvent changer au fil du temps, telles que les entrées utilisateur, la visibilité des composants ou les réponses de l'API. Le Hook useState fournit un moyen de déclarer les variables state et de les mettre à jour dans les composants fonctionnels. Voici comment utiliser le Hook useState :

Importation du Hook :

Avant d'utiliser le hook useState, vous devez l'importer à partir de la bibliothèque React.

```
import { useState } from "react";
```

Déclaration des variables State:

Pour créer une variable state, appelez le Hook useState et fournissez une valeur initiale. Le Hook renvoie un tableau avec deux éléments : la valeur de state actuelle et une fonction pour mettre à jour cette valeur.

```
function ExampleComponent() {  
  const [count, setCount] = useState(0);  
  // Initial value of count is 0  
}
```

Dans cet exemple, count est la variable state et setCount est la fonction utilisée pour mettre à jour sa valeur. La valeur initiale de count est définie sur 0.

Accès à State :

Vous pouvez accéder à la valeur actuelle d'une variable state directement dans votre code JSX.

```
function ExampleComponent() {  
  const [count, setCount] = useState(0);  
  
  return (  
    <div>  
      <p>Count: {count}</p>  
    </div>  
  );  
}
```

Ici, {count} est remplacé par la valeur actuelle de la variable state de count

Mise à jour de state :

Pour mettre à jour la valeur d'une variable state, appelez la fonction setter rentrée par le Hook useState et passez la nouvelle valeur en argument.

Dans cet exemple, cliquer sur le bouton "Incrémenter" appelle la fonction d'incrément, qui met à jour la variable state de count en incrémentant sa valeur.

```
function ExampleComponent() {  
  const [count, setCount] = useState(0);  
  
  const increment = () => {  
    setCount(count + 1);  
  };  
  
  return (  
    <div>  
      <p>Count: {count}</p>  
      <button onClick={increment}>Increment</button>  
    </div>  
  );  
}
```

Variables de state multiples :

Vous pouvez utiliser le Hook useState plusieurs fois dans un seul composant pour gérer plusieurs variables state

```
function ExampleComponent() {  
  const [count, setCount] = useState(0);  
  const [name, setName] = useState('John');  
  
  // More state variables and update functions can be declared here  
}
```

Chaque variable state a sa propre fonction de réglage, vous permettant de les mettre à jour indépendamment.

```
import { useState } from "react";
import ReactDOM from "react-dom/client";
function Car() {
  const [car, setCar] = useState({
    brand: "Ford",
    model: "Mustang",
    year: "1964",
    color: "red"
  });

  return (
    <>
      <h1>My {car.brand}</h1>
      <p>
        It is a {car.color} {car.model} from {car.year}.
      </p>
    </>
  )
}
ReactDOM.createRoot(document.getElementById('root')).render(<Car />);
```

Lorsque variable state est mis à jour, leur valeur entier est écrasé.

Que faire si nous voulons seulement mettre à jour la couleur de notre voiture?

Si nous appelions seulement setCar({color:"blue"}), cela supprimerait la marque, le modèle et l'année de notre state.

```
import { useState } from "react";
import ReactDOM from "react-dom";

function Car() {
  const [car, setCar] = useState({
    brand: "Ford",
    model: "Mustang",
    year: "1964",
    color: "red"
  });

  const updateColor = () => {
    setCar(previousState => {
      return { ...previousState, color: "blue" }
    });
  }

  return (
    <>
      <h1>My {car.brand}</h1>
      <p>
        It is a {car.color} {car.model} from {car.year}.
      </p>
      <button
        type="button"
        onClick={updateColor}
      >Blue</button>
    </>
  )
}

ReactDOM.createRoot(document.getElementById('root')).render(<Car />);
```

Nous pouvons utiliser l'opérateur de propagation JavaScript spread pour nous aider.

Comme nous avons besoin de la valeur de state actuelle, nous passons une fonction dans notre fonction setCar.

Cette fonction reçoit la valeur précédente.

Nous retournons ensuite un objet, en étalant l'état précédent de la variable state et en écrasant uniquement la couleur.

Double Binding : liaison bidirectionnelle

Dans React, la liaison bidirectionnelle fait référence à la synchronisation des données entre un élément d'interface utilisateur (comme un champ de saisie input) et une variable state dans le composant. Cela signifie que les modifications apportées à l'élément UI par l'utilisateur sont reflétées dans le state du composant, et les modifications apportées au state sont reflétées dans l'élément UI. Voici comment fonctionne la liaison bidirectionnelle dans React :

```
function ExampleComponent() {  
  const [value, setValue] = useState('');  
  
  const handleChange = (event) => {  
    setValue(event.target.value);  
};  
  
return (  
  <div>  
    <input type="text" value={value} onChange={handleChange} />  
    <p>Value: {value}</p>  
  </div>  
);  
}
```

Liez la valeur de l'élément UI (tel qu'un champ de saisie) à la variable state à l'aide de valeur de l'attribut. En outre, attachez un gestionnaire d'événements onChange pour mettre à jour le state chaque fois que l'utilisateur interagit avec l'élément UI.

REACT PROPS

```
from . import
from res.core import items

class Unit(object):
    def __init__(self, **kwargs):
        self.name = kwargs.get("name")
        self.damage = kwargs.get("damage")
        self.armor = kwargs.get("armor")
        self.hit_points = kwargs.get("hp")
        self.current_hit_points = kwargs.get("hp")
        self.level = kwargs.get("level")

    def attack(self, enemy: 'Unit') -> int:
        """
        Attack enemy unit. Return number of damage
        """
        damage_top_limit = self.damage + round(self.level / 10)
        damage_bot_limit = self.damage - round(self.level / 10)
        calculated_damage = random.randint(damage_bot_limit, damage_top_limit)
        if calculated_damage < 0:
            return 0
        enemy.current_hit_points -= calculated_damage
        return calculated_damage
```

props dans React :

Dans React, les props (abréviation de properties) sont un moyen de passer des données des composants parents aux composants children. Ils vous permettent de personnaliser et de configurer les composants chilren de manière dynamique, rendant votre application plus flexible et réutilisable. Voici comment utiliser les props pour passer des données et des méthodes entre les composants :

Transmettre des données avec des accessoires :

Pour transmettre des données d'un composant parent à un composant enfant, vous pouvez simplement ajouter des attributs au composant enfant lors du rendu.

```
// ParentComponent.js
import React from 'react';
import ChildComponent from './ChildComponent';

function ParentComponent() {
  const greeting = 'Hello, world!';

  return (
    <div>
      <ChildComponent message={greeting} />
    </div>
  );
}

// ChildComponent.js
import React from 'react';

function ChildComponent(props) {
  return <p>{props.message}</p>;
}

export default ChildComponent;
```

Dans cet exemple, le ParentComponent passe la variable greeting comme une prop appelée message au ChildComponent. Le ChildComponent reçoit la prop et affiche sa valeur dans un élément de paragraphe.

Accès aux Props dans les ChildComponent :

Dans un ChildComponent, vous pouvez accéder aux props via l'objet props passé en argument à la fonction component.

```
function ChildComponent(props) {  
  return <p>{props.message}</p>;  
}
```

Ici, props.message accède à la valeur de la prop de message passée à partir du composant parent.

Passer des méthodes avec des Props :

En plus des données, vous pouvez également passer des méthodes sous forme props aux composants chilren. Cela permet aux composants chilren de communiquer avec leurs composants parents en invoquant ces méthodes.

```
// ParentComponent.js
function ParentComponent() {
  const handleClick = () => {
    console.log('Button clicked!');
  };

  return (
    <div>
      <ChildComponent onClick={handleClick} />
    </div>
  );
}

// ChildComponent.js
function ChildComponent(props) {
  return <button onClick={props.onClick}>Click me</button>;
}
```

Dans cet exemple, le ParentComponent définit une méthode handleClick et la passe comme une prop appelée onClick au ChildComponent. Lorsque le bouton dans le composant ChildComponent est cliqué, il invoque la méthode handleClick définie dans le composant parent.