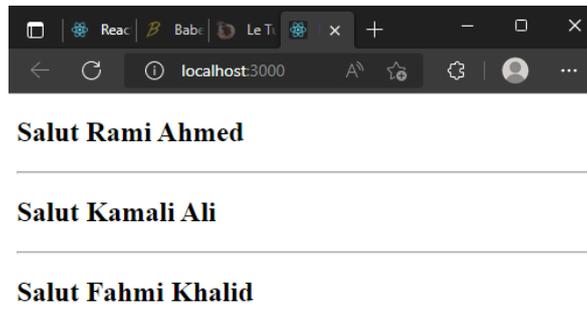




6. Manipuler les propriétés et gérer les états

6.1. Introduction :

Dans la séance précédente, on avait dit que les composants sont réutilisables, si on est dans la situation ou on souhaite afficher le même composant plusieurs fois mais avec des informations différentes.



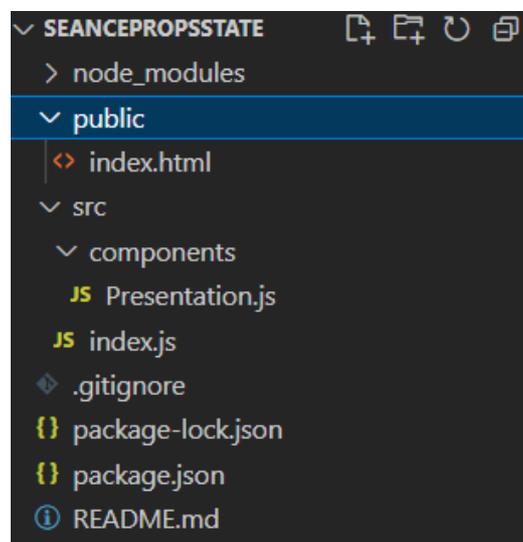
React utilise un Object props qui va nous permettre de passer les informations nécessaires au composant, ce qui rend le composant dynamique.

6.2. Manipulation des props dans un composant créé via une fonction

Pour illustrer ce besoin on va créer un composant Presentation, dans un premier temps on va créer le composant via une fonction puis dans un deuxième temps on va créer le composant via une classe.

Pour ce faire, il est souhaitable de créer le composant dans un fichier js Presentation.js qui se trouvera dans le dossier components.

- Créer le projet **seancePropsState**
- Créer le dossier components
- Créer le fichier Presentation.js dans le dossier src/components
- Ajouter le code suivant dans Presentation.js





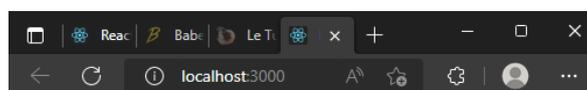
Presentation.js

```
import React from 'react'  
export default function Presentation(props){  
  console.log(props)  
  return (  
    <div >  
      <h2>Salut {props.nom} {props.prenom}</h2>  
      <hr/>  
    </div>  
  )  
}
```

Le fichier index.js

```
import React from 'react';  
import ReactDOM from 'react-dom/client';  
import Presentation from './components/Presentation';  
  
const element=document.getElementById("root")  
const root=ReactDOM.createRoot(element)  
function App(){  
  return(  
    <div>  
      <Presentation nom="Rami" prenom="Ahmed"/>  
      <Presentation nom="Kamali" prenom="Ali" />  
      <Presentation nom="Fahmi" prenom="Khalid"/>  
    </div>  
  )  
}  
root.render(<App/>)
```

Le rendu



Salut Rami Ahmed

Salut Kamali Ali

Salut Fahmi Khalid

Le rendu de la console

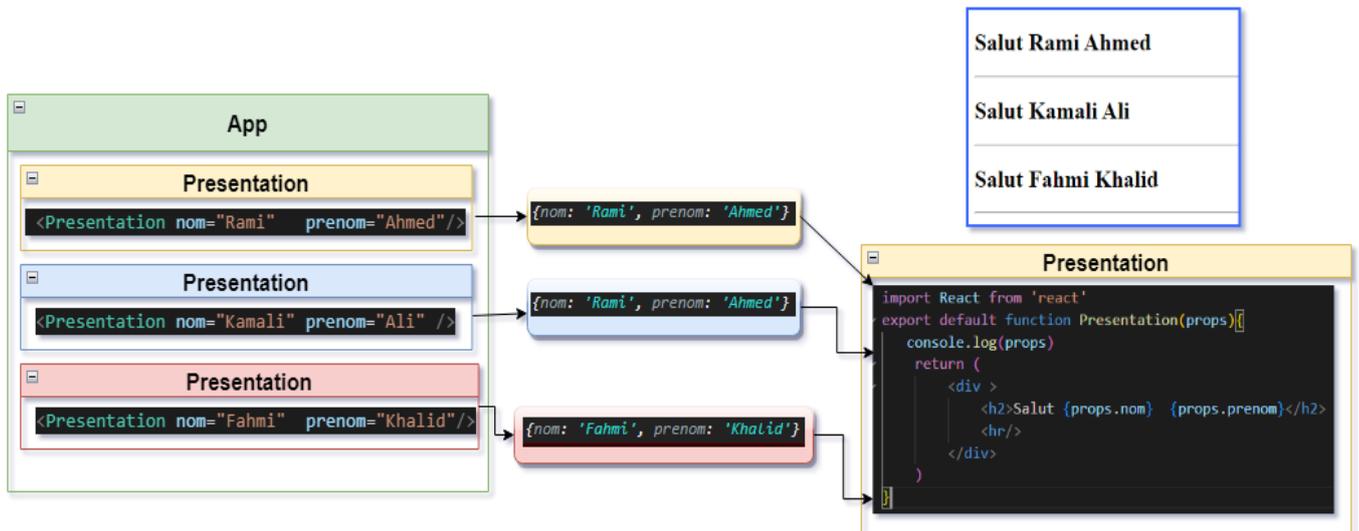
```
▶ {nom: 'Rami', prenom: 'Ahmed'}  
▶ {nom: 'Kamali', prenom: 'Ali'}  
▶ {nom: 'Fahmi', prenom: 'Khalid'}
```



On remarque que **props** c'est un objet JavaScript

L'objet props est immuable c.-à-d. on ne peut pas changer les propriétés

Cette écriture n'est pas permise : ~~props.nom='Jamili'~~



L'écriture :

```
<Presentation nom="Rami" prenom="Ahmed" />
```

Fait appel à la fonction Presentation en passant en argument l'objet props qui fait référence à l'objet **{nom: 'Rami', prenom: 'Ahmed'}**, cet objet est créé à partir des attributs nom, prenom de l'élément Presentation.

Il est recommandé de choisir le nom props

```
Presentation
<Presentation nom="Rami" prenom="Ahmed" />
```

```
l'objet props= {nom: 'Rami', prenom: 'Ahmed'}
```

La fonction Presentation retourne le JSX qui affiche

Salut Rami Ahmed



```
import React from 'react'
export default function Presentation(props){
  console.log(props)
  return (
    <div >
      <h2>Salut {props.nom} {props.prenom}</h2>
      <hr/>
    </div>
  )
}
```

Cette écriture permet de récupérer les propriétés nom et prenom de l'objet props passé en argument de la fonction Presentation

```
<h2>Salut {props.nom} {props.prenom}</h2>
```

Remarque : On peut passer dans l'objet props :

- une valeur
- un objet
- une liste

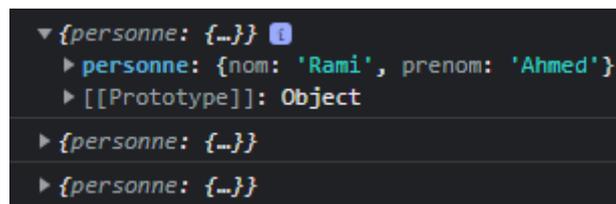
Passage d'un objet dans props

```
function App(){
  let personne1={nom:"Rami",prenom:"Ahmed"}
  let personne2={nom:"Kamali",prenom:"Ali"}
  let personne3={nom:"Fahmi",prenom:"Khalid"}
  return(
    <div>
      <Presentation personne={personne1}/>
      <Presentation personne={personne2} />
      <Presentation personne={personne3} />
    </div>
  )
}
```

Remarque : on peut passer directement l'objet

```
<Presentation personne={{nom:"Fahmi",prenom:"Khalid"}} />
```

Le rendu de console



```
▼ {personne: {...}} ⓘ
  ► personne: {nom: 'Rami', prenom: 'Ahmed'}
  ► [[Prototype]]: Object
  ► {personne: {...}}
  ► {personne: {...}}
```

Cette fois ci props c'est l'objet {personne :{nom :'Rami',prenom :'Ahmed'}}



Récupération des données passées dans l'objet props

```
import React from 'react'
export default function Presentation(props){
  console.log(props)
  return (
    <div >
      <h2>Salut {props.personne.nom} {props.personne.prenom}</h2>
      <hr/>
    </div>
  )
}
```

On peut utiliser le destructeur objet javascript ES6 voir cours séance1

```
import React from 'react'
export default function Presentation(props){
  console.log(props)
  const {nom,prenom}=props.personne
  return (
    <div >
      <h2>Salut {nom} {prenom}</h2>
      <hr/>
    </div>
  )
}
```

Passer un Array dans props

```
import React from 'react';
import ReactDOM from 'react-dom/client'
import Presentation from './components/Presentation';

const element=document.getElementById("root")
const root=ReactDOM.createRoot(element)
function App(){
  let personne1={nom:"Rami",prenom:"Ahmed"}
  let diplomes=["Bac","Licence","Master"]
  return(
    <div>
      <Presentation personne={personne1} diplomes={diplomes} />
    </div>
  )
}
root.render(<App/>)
```



Composent Presentation

```
import React from 'react'
export default function Presentation(props){
  console.log(props)
  const {nom,prenom}=props.personne
  return (
    <div >
      <h2>Salut {nom} {prenom}</h2>
      <hr/>
      <h3>Diplomes</h3>
      <p>{props.diplomes}</p>
    </div>
  )
}
```

Le rendu console

```
▼ {personne: {...}, diplomes: Array(3)} ⓘ
  ► diplomes: (3) ['Bac', 'Licence', 'Master']
  ► personne: {nom: 'Rami', prenom: 'Ahmed'}
  ► [[Prototype]]: Object
```

Maintenant props c'est l'objet javascript

```
{personne :{ nom:"Rami", prenom:"Ahmed"},diplomes:["Bac","Licence","Master"]}
```

Salut Rami Ahmed

Diplomes

BacLicenceMaster

Comme vous remarquez React fait la concaténation des éléments de la liste diplomes dans une chaîne de caractère {props.diplomes}, il y aura une séance ultérieure qui traite la manipulation des éléments d'un Array en utilisant la méthode map.



6.3. Manipulation des props dans un composant créé via une classe

On utilise le même projet

On utilisera le composant Salutation créé via classe
pour ce faire créer un fichier Salutation.js dans le dossier components

Salutation.js

```
import React from 'react'
export default class Salutation extends React.Component{
  render(){
    console.log(this.props)
    return (
      <div >
        <h2>Salut {this.props.nom} {this.props.prenom}</h2>
        <hr/>
      </div>)
  }
}
```

Vous allez remarquer que l'objet props est accessible directement dans l'objet Salutation, mais il faut ajouter le mot clé **this** : **{this.props.nom}**

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import Salutation from './components/Salutation';
const element=document.getElementById("root")
const root=ReactDOM.createRoot(element)
function App(){
  return(
    <div>
      <Salutation nom="Rami" prenom="Ahmed" />
      <Salutation nom="Kamali" prenom="Ali" />
      <Salutation nom="fahmi" prenom="Khalid" />
    </div>
  )
}
root.render(<App/>)
```

L'écriture :

```
<Salutation nom="Rami" prenom="Ahmed"/>
```

Crée une instance de la classe Salutation après la création, la méthode render est exécutée, elle retourne du JSX qui affiche

Salut Rami Ahmed

On peut passer dans l'objet props une valeur, un objet et une liste



Passage d'un objet dans props

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client'
import Salutation from './components/Salutation';
const element=document.getElementById("root")
const root=ReactDOM.createRoot(element)
function App(){
  let personne1={nom:"Rami",prenom:"Ahmed"}
  let personne2={nom:"Kamali",prenom:"Ali"}
  let personne3={nom:"Fahmi",prenom:"Khalid"}

  return(
    <div>
      <Salutation personne={personne1} />
      <Salutation personne={personne2} />
      <Salutation personne={personne3} />
    </div>
  )
}
root.render(<App/>)
```

Salutation.js

```
import React from "react";

export default class Salutation extends React.Component {
  render() {
    console.log(this.props);
    return (
      <div>
        <h2>
          Salut {this.props.personne.nom} {this.props.personne.prenom}
        </h2>
        <hr />
      </div>
    );
  }
}
```



Passage d'un Array dans props

Index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import Salutation from "../components/Salutation";
const element = document.getElementById("root");
const root = ReactDOM.createRoot(element);
function App() {
  let personne1 = { nom: "Rami", prenom: "Ahmed" };
  let diplomes = ["Bac", "Licence", "Master"];
  return (
    <div>
      <Salutation personne={personne1} diplomes={diplomes} />
    </div>
  );
}
root.render(<App />);
```

Salutation.js

```
import React from "react";

export default class Salutation extends React.Component {
  render() {
    console.log(this.props);
    return (
      <div>
        <h2>
          Salut {this.props.personne.nom} {this.props.personne.prenom}
        </h2>
        <h3>Diplomes</h3>
        <p>{this.props.diplomes}</p>
        <hr />
      </div>
    );
  }
}
```



6.4. Passer dynamique contenu a un composant

Retournant a notre exemple Presentation

```
import React from "react";
import ReactDOM from "react-dom/client";
import Presentation from "../components/Presentation";
const element = document.getElementById("root");
const root = ReactDOM.createRoot(element);
function App() {
  return (
    <div>
      <Presentation nom="Rami" prenom="Ahmed">
        <p>ce ci est un children props</p>
      </Presentation>
      <Presentation nom="Kamali" prenom="Ali">
        <button>quitter</button>
      </Presentation>
    </div>
  );
}
root.render(<App />);
```

Dans ce cas l'élément Presentation contient un élément enfant entre la balise ouvrante et fermante.

Les éléments enfants sont différents, un paragraphe et un bouton

```
import React from 'react'
export default function Presentation(props){
  console.log(props)
  return (
    <div >
      <h2>Salut {props.nom} {props.prenom}</h2>
      {props.children}
      <hr/>
    </div>
  )
}
```

Pour récupérer l'élément enfant en utilise la propriété children : **{props.children}**

Le rendu

Salut Rami Ahmed

ce ci est un children props

Salut Kamali Ali

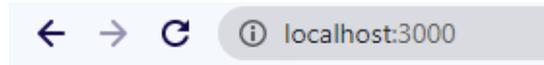
quitter



6.5. Utiliser l'objet state pour mettre à jour un composant créer par classe

State est un objet pouvant être mis à jour qui peut être utilisé pour contenir les données et contrôler le comportement du composant. Seuls les composants de classe peuvent avoir un état, pas les composants fonctionnels. Lorsque l'état est modifié, React restitue automatiquement le composant qui provoque une mise à jour d'affichage.

Pour expliquer l'objet state, considérons que nous souhaitons écrire un composant Message simple qui permet d'afficher le rendu suivant

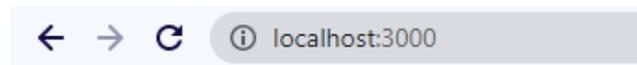


Bien venu visiteur

inscription

Si l'utilisateur clique sur le bouton inscription

Le rendu devient



votre inscription est effectuée

merci

Vous allez remarquer que après le click sur le bouton le message Bien Venu visiteur change pour devenir votre inscription est effectuée, le texte de bouton inscription change pour devenir merci.

D'où il faut avoir deux variables message et btnMessage dont les valeurs vont être modifiées après la clique sur le bouton, cette modification doit provoquer un rafraichissement d'affichage.

Pour illustrer ces notions de gestion d'état, on va expliquer ensemble le code de cet exemple.

Etapes à suivre pour créer l'application Message.

Ajouter un fichier Message.js dans le dossier src/components.

Message.js

```
import React from 'react'
export default class Message extends React.Component{
  constructor(){
    super()
    this.state={message:"Bien venu visiteur",btnMessage:"inscription"}
  }
}
```



```
inscription(){
this.setState({message:"votre inscription est effectuée",btnMessage:"merci"})
}
render(){
return(<div>
<h2>{this.state.message}</h2>
<button onClick={()=>this.inscription()} >{this.state.btnMessage}</button>
</div>)
}
}
```

index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import Message from "./components/Message";
const element = document.getElementById("root");
const root = ReactDOM.createRoot(element);
function App() {
  return (
    <div>
      <Message/>
    </div>
  );
}
root.render(<App />);
```

Explication du code de la classe Message

```
constructor(){
  super()
  this.state={message:"Bien venu visiteur",btnMessage:"inscription"}
}
```

Dans le constructeur on fait appel au constructeur de la classe mère React.Component par

```
super()
```

création de l'objet state qui contient les deux propriétés message et btnMessage ,les deux propriétés sont initialisées,

```
this.state={message:"Bien venu visiteur",btnMessage:"inscription"}
```

on remarque que state est un objet de la classe Message
mise à jour de state

```
inscription(){
  this.setState({message:"votre inscription est effectuée",btnMessage:"merci"})
}
```

Comme vous remarquez, la modification des valeurs des propriétés se fait via la fonction **this.setState()**,



La fonction `setState` modifie les propriétés puis provoque un appel de la méthode `render()`, c'est ainsi que l'interface utilisateur est mise à jour avec les nouvelles valeurs de `{this.state.message}` et `{this.state.btnMessage}`

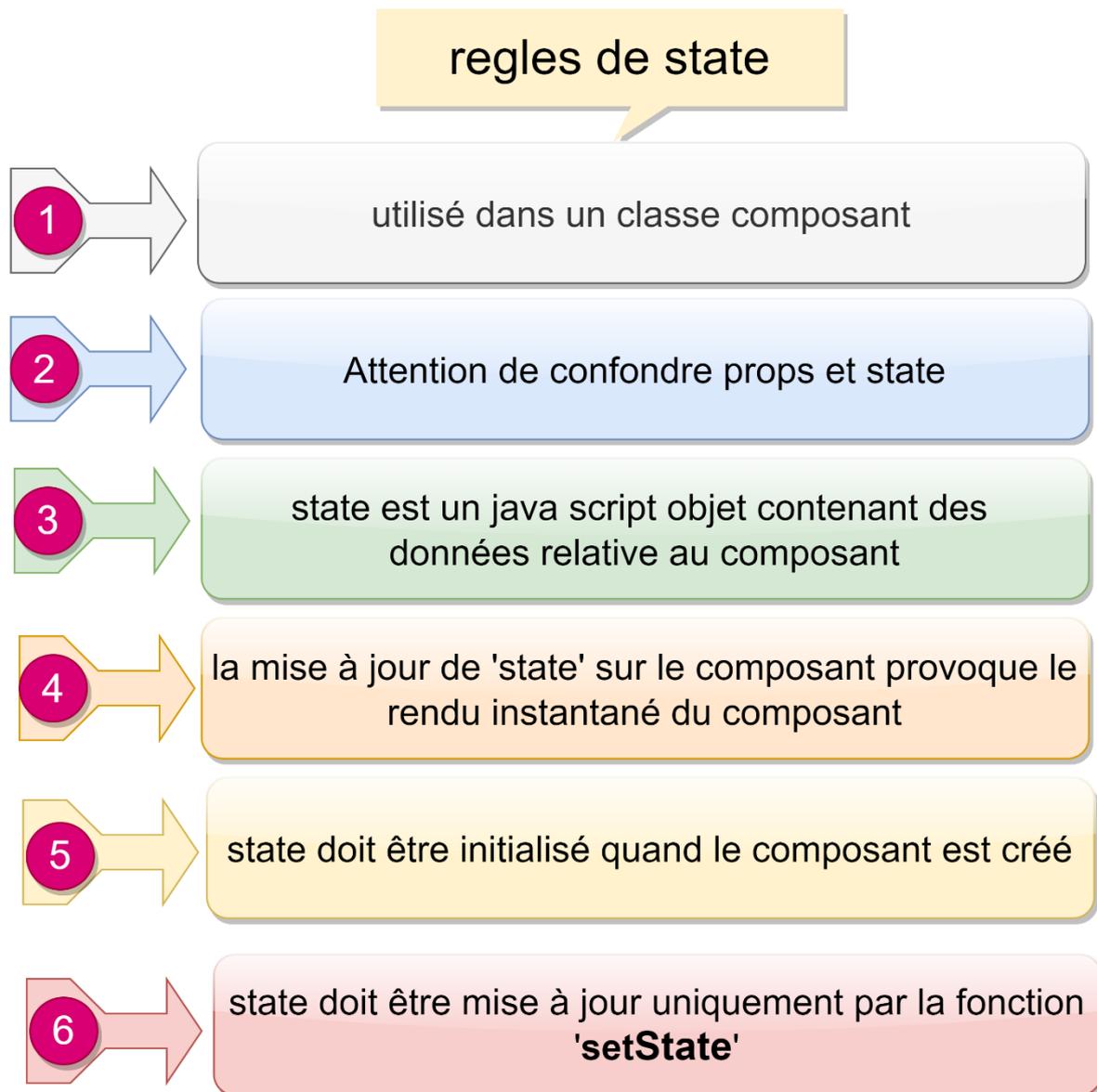
Remarque Très importante

Il ne faut pas modifier l'état `state` directement

```
this.state.message="message 1"
```

```
this.state.btnMessage="message 2"
```

en effet cette modification directe change l'état mais elle ne provoque pas l'appel de la méthode `render()`, d'où l'interface utilisateur n'affiche pas les nouvelles valeurs des propriétés de l'état .





Il faut impérativement faire attention à la dernière règle, la mise à jour de l'état (state) doit être fait uniquement via la fonction setState

```
import React from 'react'
export default class Message extends React.Component{ 1
  constructor(props){ 2
    super(props)
    5 this.state={message:"Bien venu visiteur",btnMessage:"inscription"} 3
  }
  inscription(){
    6 this.setState({message:"votre inscription est effectuée",btnMessage:"merci"})
  }
  render(){ 4
    return(<div>
    <h2>{this.state.message}</h2>
    <button onClick={()=>this.inscription()} >{this.state.btnMessage}</button>
    </div>)
  }
}
```