

# Développement front-end avec React

Pr ERRATTAHI Rahhal  
errattahi.r@ucd.ac.ma

# single-page apps (SPA)

# SPA vs MPA

---

## ■ **MPA (Multiple Page Application) – concept de base**

Dans une architecture MPA client/serveur traditionnelle, chaque clic de l'utilisateur déclenche une requête HTTP vers le serveur. Le résultat de cette nouvelle requête est un rafraîchissement complet de la page, même si une partie du contenu reste inchangée.

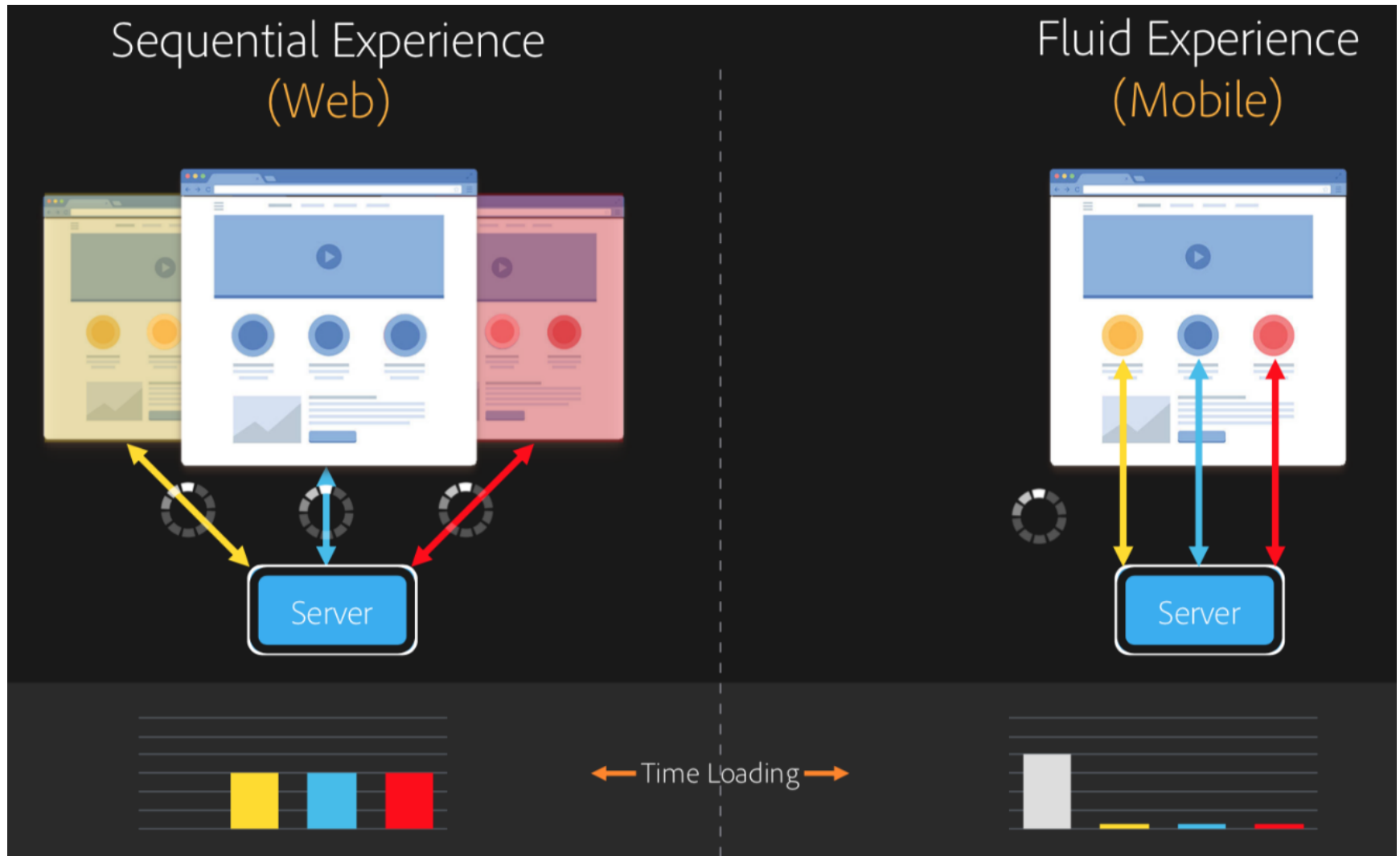
# SPA vs MPA

---

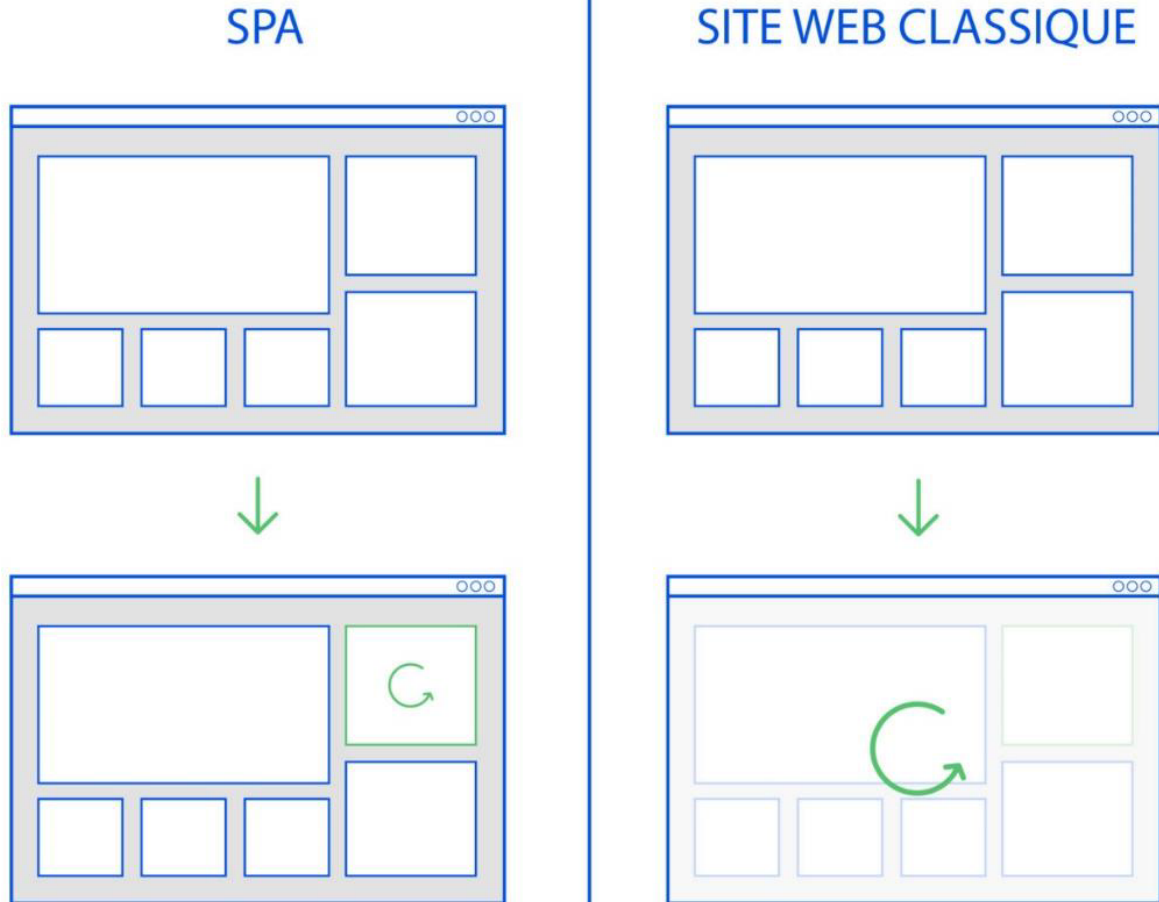
## ■ SPA – Ajax et JavaScripts

Le cœur d'un SPA est basé sur Ajax, un ensemble de techniques de développement qui permet au client d'envoyer et de récupérer des données du serveur de manière asynchrone (en arrière-plan) sans interférer avec l'affichage et le comportement de la page web. Ajax permet aux pages web et, par extension, aux applications web, de modifier le contenu de manière dynamique sans avoir à recharger la page entière.

# SPA vs MPA



# Fonctionnement du SPA



# Avantages et inconvénients

---

## ■ **Les avantages**

- **Vitesse**
- **Expérience utilisateur (UX)**
- **Capacités de mise en cache**
- **Débogage**

## ■ **Les inconvénients**

- **Historique du navigateur**
- **Optimisation du référencement**
- **Problèmes de sécurité**

# Comprendre les concepts de React JS



# Présentation de React

---

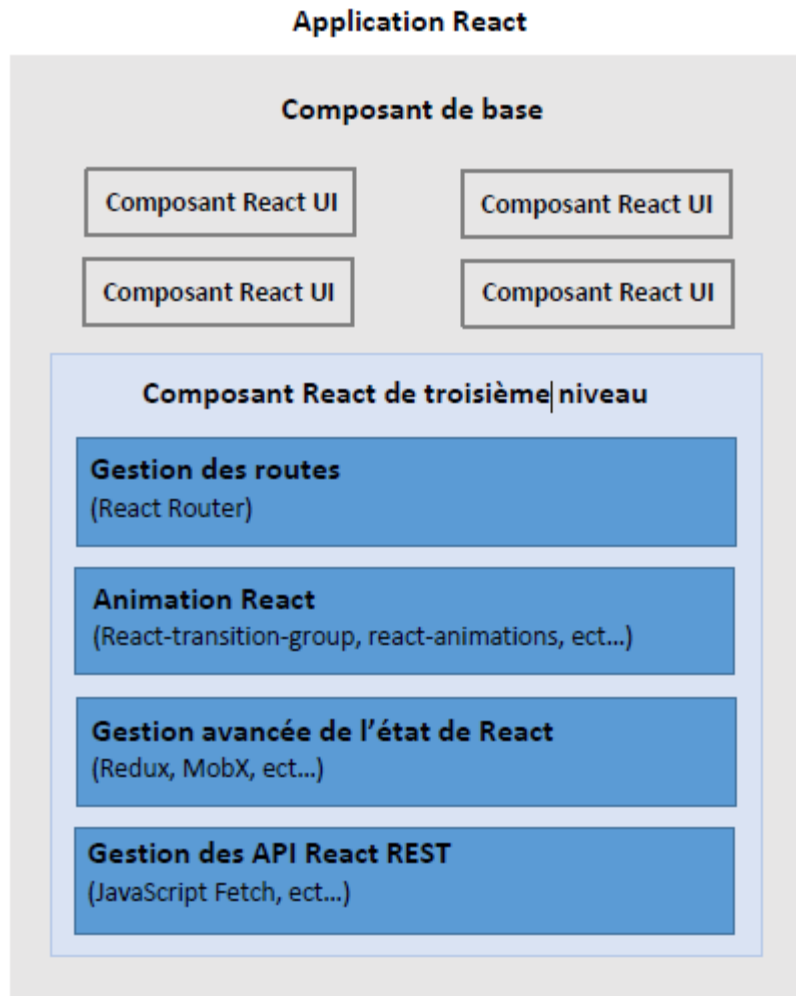
- React (Ou ReactJS, React.js) est une bibliothèque JavaScript développée par Facebook en 2013, utilisée pour créer des composants d'affichage réutilisables.
- React n'est pas seulement une bibliothèque permettant d'effectuer des affichages sur le Web, elle peut également servir à produire des applications natives iPhone et Android, en utilisant une variante nommée React Native.

# L'écosystème React

---

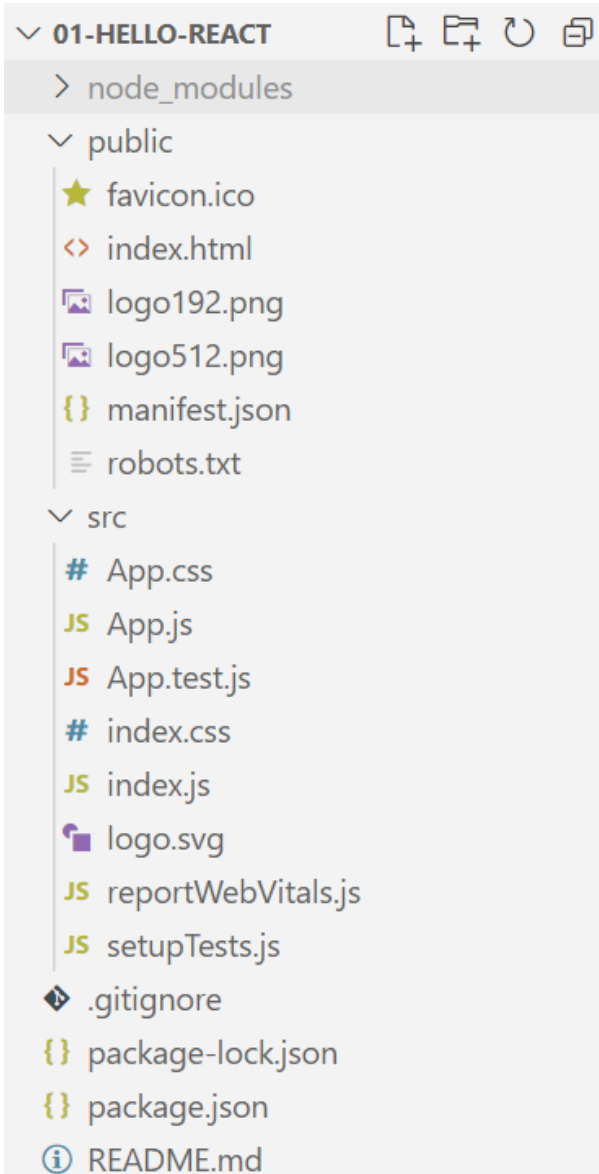
- **NPM:** npm fait deux choses : gérer vos dépendances et lancer des scripts. C'est l'équivalent de pip pour Python
- **ECMAScript6:** ES6, est un standard de langage de programmation, il définit : La syntaxe, Les types de variable et encore pas mal d'autres choses...
- **Babel:** Afin de faire de l'ES6 dès aujourd'hui, il faut le compiler, ou le transpiler. Pour cela il y a notamment **Babel** qui va transformer pour vous le code en ES5
- **Bundlers:** Le bundler condense tout le code de votre application pour n'avoir qu'un seul fichier javascript à importer dans votre page HTML.

# Architecture d'une application React



1. L'application React commence à démarrer avec un seul composant racine ;
2. Le composant racine est créé grâce à un ou plusieurs composants ;
3. Chaque composant est formé comme un composant de composants plus petits ;
4. La grande partie des composants sont des composants d'interface utilisateur ;
5. L'application React a la possibilité d'ajouter un composant tiers à des fins spécifiques comme le routage, l'animation, la gestion de l'état, etc.

# Structure de projet



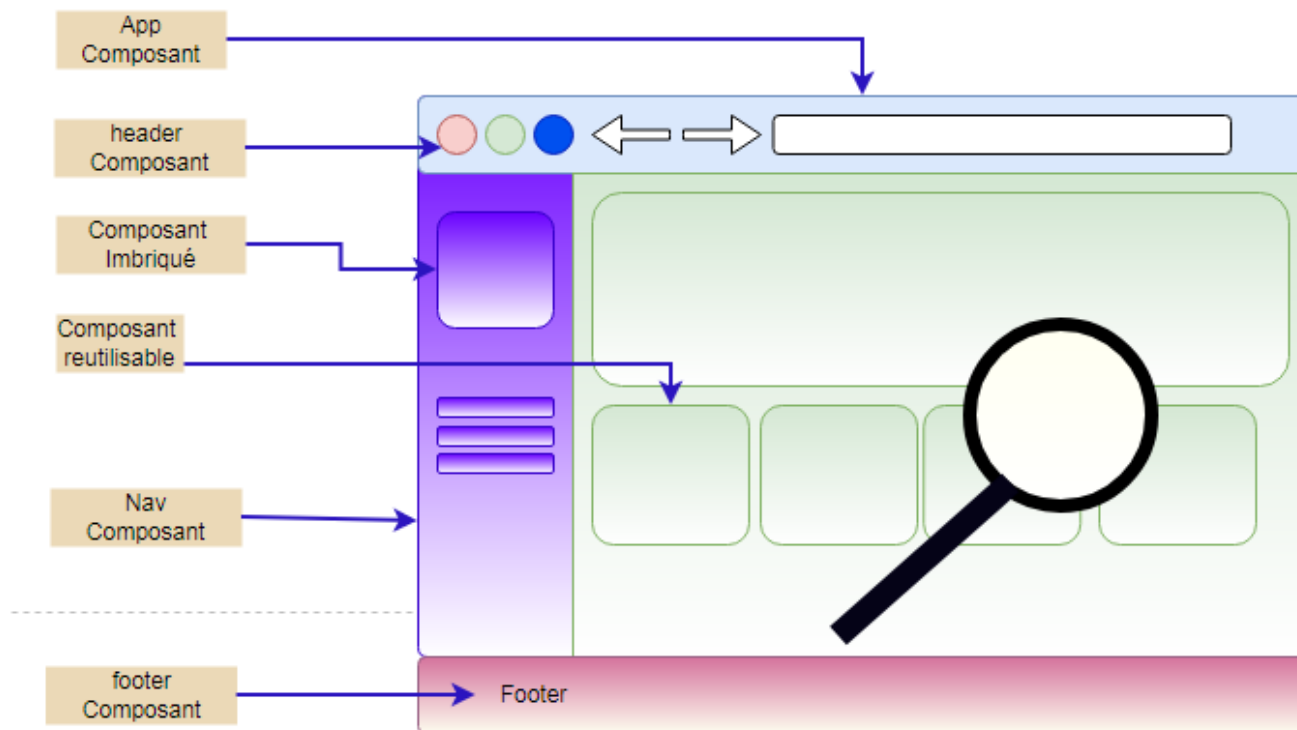
- Le dossier **node\_modules** contient toutes les librairies javascript dont vous aurez besoin
- Le dossier **public** vous y trouverez le fichier **index.html** qui est le point de départ
- Le fichier **package.json** est certainement le fichier qui contient plusieurs informations sur votre application React notamment les scripts, les dépendances.
- Le fichier **Readme** permet de documenter votre app en Markdown.
- Le dossier **src** est l'endroit où toute la magie de votre application va opérer, on y retrouve le fichier **index.js** qui est le fichier qui charge notre premier composant le composant App.

# TP1: Installation et configuration de l'environnement de développement

# Composants en React

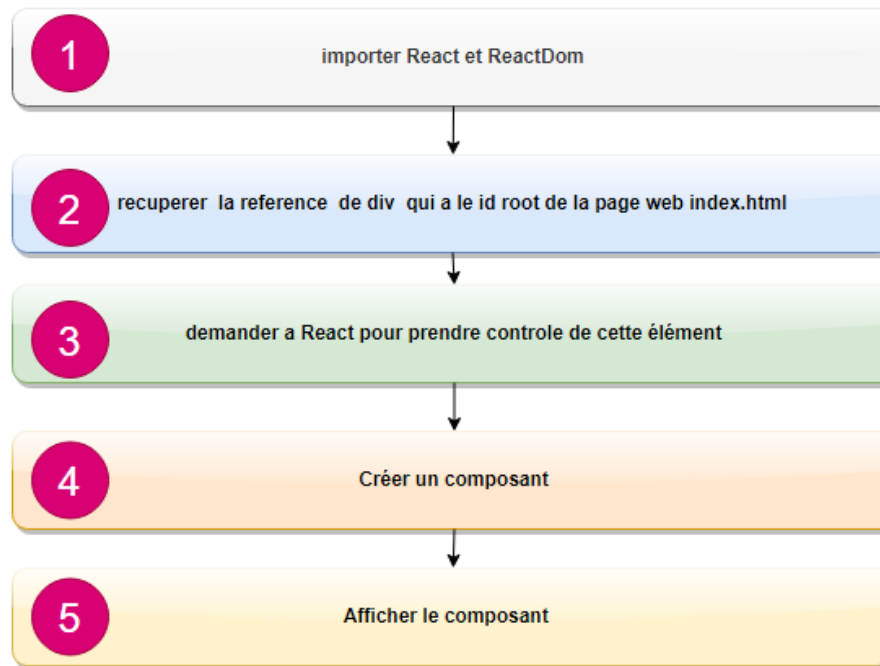
# Composants

- Chaque application React est constitué d'un ensemble de composants.
- Les composants sont des éléments autonomes qui encapsulent une partie de l'interface utilisateur et peuvent être combinés pour créer des interfaces utilisateurs plus complexes.



# Etapes pour créer un composant

- Un composant est une **fonction** qui retourne de JSX
- Ou une **classe** qui hérite de la classe **React.Component** possédant la fonction render qui retourne du JSX





# Etapes pour créer un composant

## Etape 1 :

```
// 1) importer React et ReactDOM
import React from 'react';

import ReactDOM from 'react-dom/client'
```

**React**

bibliothèque qui définit ce qu'est un composant et comment plusieurs composants fonctionnent ensemble

**ReactDOM**

bibliothèque qui permet d'afficher un composant dans le navigateur

Le package **react-dom/client** fournit des méthodes spécifiques au client utilisées pour initialiser une application sur le client. La plupart de vos composants ne devraient pas avoir besoin d'utiliser ce module.

# Etapes pour créer un composant

## Etape 2 :

//2) faire reference à div id=root de index.html

```
const element=document.getElementById("root");
```

on crée une référence vers l'élément div id='root' de la page public/index.html (la page index.html contient `<div id= 'root'></div>` )

C'est dans cette élément que React va injecter le code HTML

```
<body>  
<noscript>You need to enable JavaScript to run this app.</noscript>  
<div id="root"></div>  
</body>
```

# Etapes pour créer un composant

## Etape 3 :

//3) prendre le controle de l'element par React

```
const root=ReactDOM.createRoot(element)
```

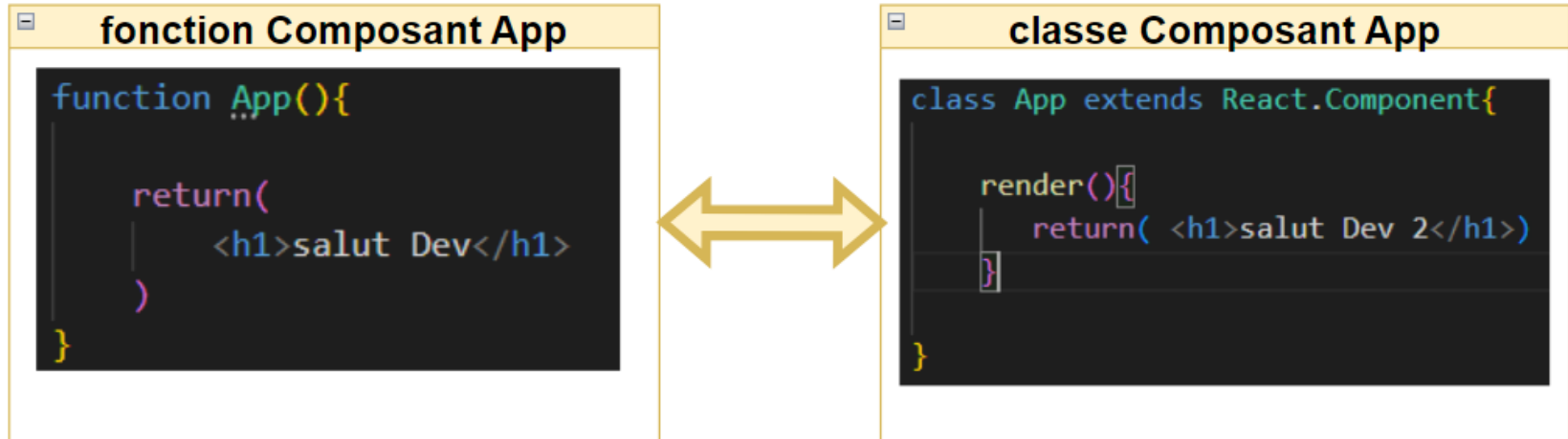
On prend le contrôle de l'élément div id='root' de la page public/index.html par React  
Créez une racine React pour le conteneur fourni et renvoyez la racine. La racine peut être utilisée pour restituer un élément React dans le DOM avec render :

```
root.render(<App/>)
```

# Etapes pour créer un composant

## Etape 4 :

On crée un composant qui s'appelle **App**, comme on avait expliqué avant un composant est une fonction qui retourne de JSX, il peut être aussi une classe qui hérite de la classe **React.Component** possédant la fonction **render** qui retourne du JSX,



Pour une bonne organisation de l'application chaque composant doit être dans fichier js portant le même nom de composant.

# Etapes pour créer un composant

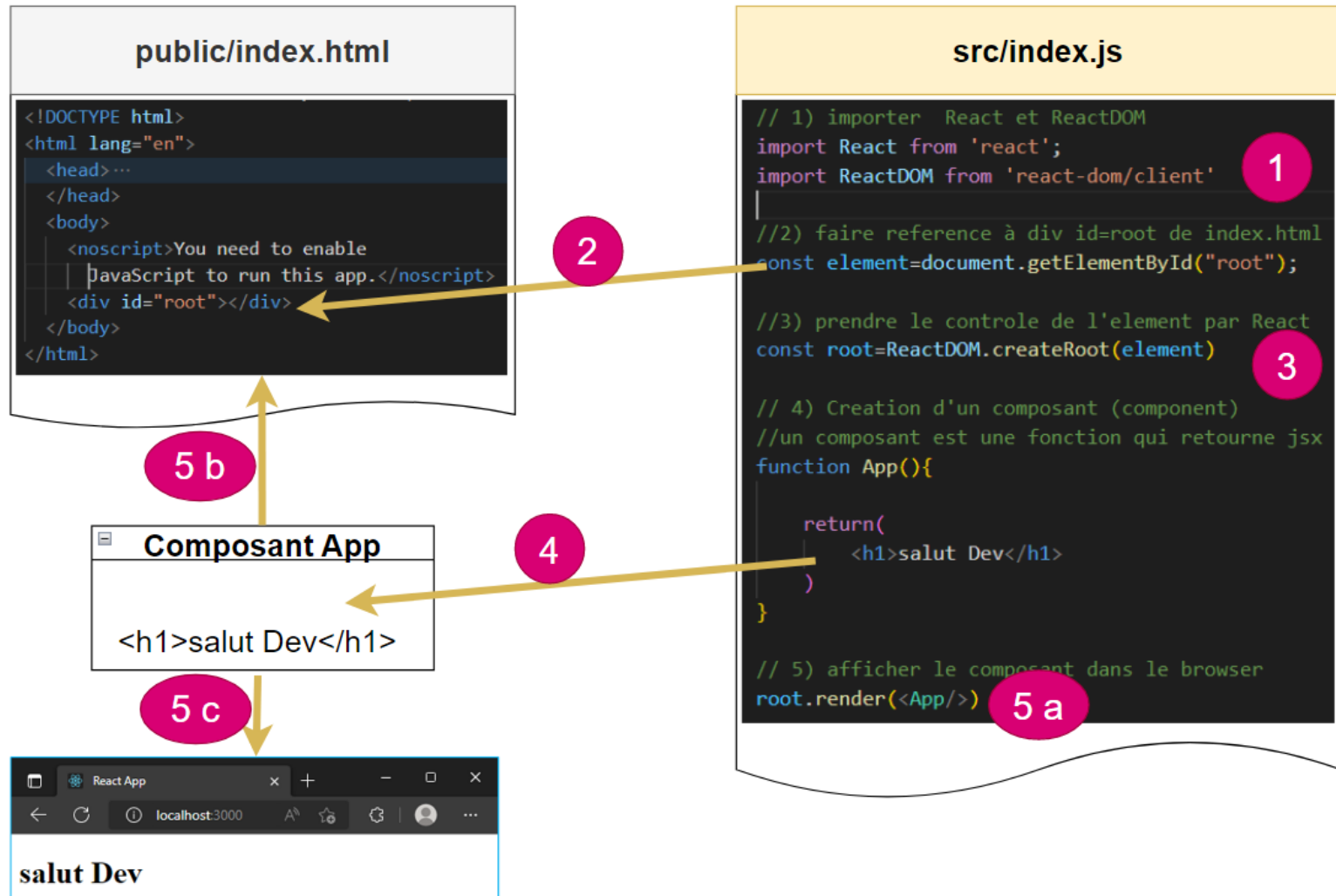
---

## Etape 5 :

```
// 5) afficher le composant dans le browser  
root.render(<App/>)
```

on passe App composant comme JSX élément a la méthode root.render()  
permet d'afficher le contenu du composant App dans le <div id='root'></div> de la  
page indx.html

# Etapes pour créer un composant



# Le langage JSX

---

```
<h1>salut Dev</h1>
```

- Cette drôle de syntaxe n'est ni une chaîne de caractères ni du HTML.
- Ça s'appelle du JSX, et c'est une extension syntaxique de JavaScript. Utilisée avec React afin de décrire à quoi devrait ressembler l'interface utilisateur (UI). JSX vous fait sûrement penser à un langage de balisage, mais il recèle toute la puissance de JavaScript.
- JSX produit des « éléments » React.

# Le langage JSX

---

JSX est une forme d'écriture des éléments React, plus simple à lire et à écrire que les instructions **React.createElement()**. Cette syntaxe est donc abondamment utilisée dans les programmes React.

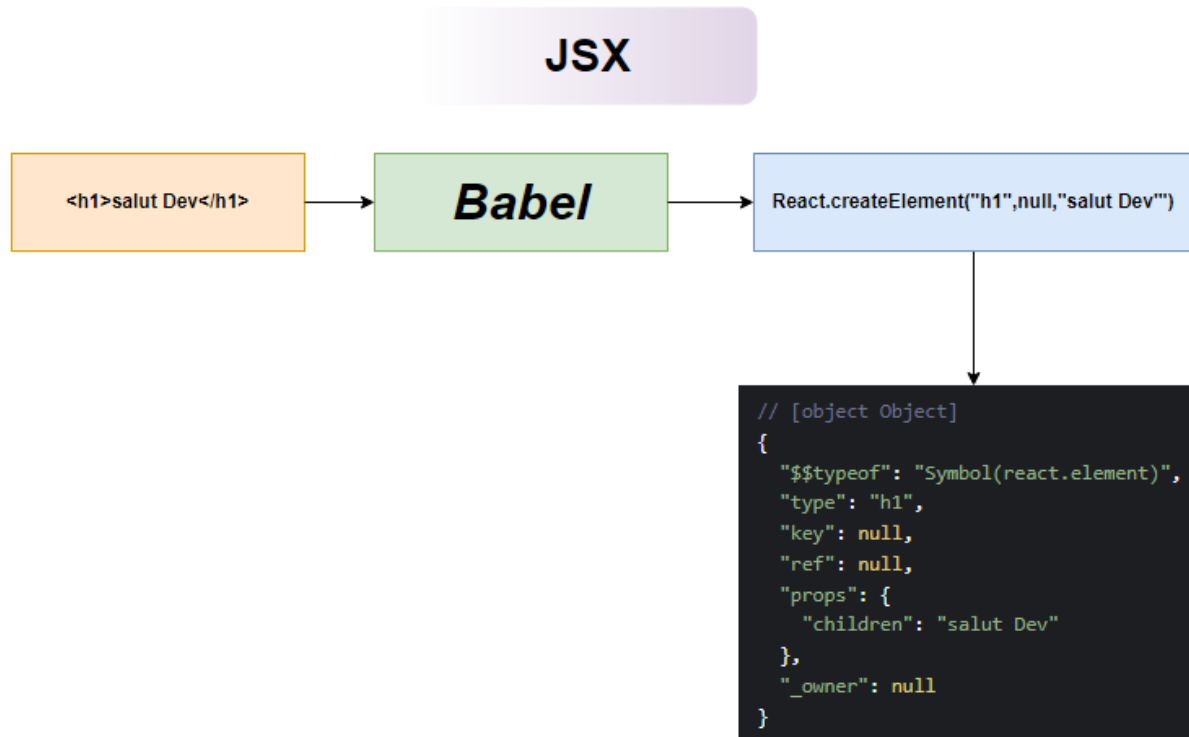
Au fur et à mesure, vous verrez que le JSX est un langage très intuitif à utiliser. Voici les deux propriétés de base pour ce qui est des balises utilisées :

- Toute balise commençant par une minuscule (div, span, label, etc.) est réservé aux éléments HTML. Ces éléments sont déclarés par React DOM, et vous obtiendrez une erreur si vous utilisez un élément inexistant.
- Toute balise commençant par une majuscule (Greetings, App, etc.) doit être déclarée explicitement, ce doit donc être un élément du scope courant : fonction déjà déclarée, composant importé d'une bibliothèque ou d'un autre fichier...



# Le langage JSX

## Comment se fait l'interprétation de JSX



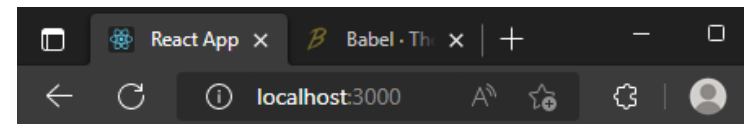

# Le langage JSX

## Ajouter de variables et expression java script dans JSX

- Si on veut utiliser une variable ou expression js dans JSX on utilise les accolades {},ici on passe la variable nom dans JSX
- `<h1>salut {nom}</h1>`

Exemple1:

```
function App(){  
  const nom='RAMI'  
  return(  
    <h1>salut {nom}</h1>  
  )  
}
```



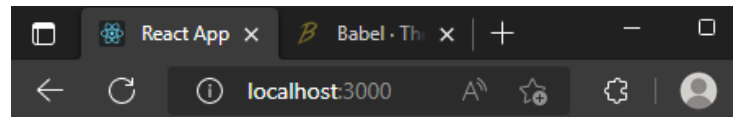
**salut RAMI**

# Le langage JSX

## Ajouter de variables et expression java script dans JSX

### Exemple2:

```
function App(){  
  const nom='RAMI'  
  const time=new Date().toLocaleDateString();  
  return(  
    <h1>salut {nom} la date {time}</h1>  
  )  
}
```



**salut RAMI la date 22/07/2022**

# Le langage JSX

## Ajouter de variables et expression java script dans JSX

### Exemple3:

- Si le nombre d'article acheté est supérieur a 5 remise 2% si non remise 0%

```
function App(){
  const nbArticle=6
  let remise=0
  if(nbArticle>=5){
    remise=2
  }
  return(<div>
    <h2> Remise </h2>
    <p> votre remise est: {remise} %</p>
  </div>)
}
```



# Le langage JSX

## ■ Exercice d'entraînement :

On souhaite afficher le nom, le prénom et l'âge dans l'élément p du composant suivant :

Pour calculer l'âge utiliser la fonction getAge()

```
import React from 'react';
import ReactDOM from 'react-dom/client';
const element=document.getElementById("root");
const root=ReactDOM.createRoot(element)
//fonction getAge reçoit la date en format DD/MM/YYYY
function getAge(dateNaissance){
    let from = dateNaissance.split("/");
    let birthdateTimeStamp = new Date(from[2], from[1] - 1, from[0]);
    let cur = new Date();
    let diff = cur - birthdateTimeStamp;
    // difference en milliseconds
    let currentAge = Math.floor(diff/31557600000);
    // Division par 1000*60*60*24*365.25
    return currentAge;
}
function App(){
    const nom='RAMI'
    const prenom='AHMED'
    const dateNaissance='23/03/2000'
    return(<div>
        <h2> Informations </h2>
        <p> -----</p>
    </div>
    )
}
root.render(<App/>)
```

**Informations**  
RAMI AHMED votre age est:22

# Le langage JSX

## ■ Solution:

code permettant de calculer les valeurs qu'on souhaite afficher dans notre JSX

Le contenu qu'on va afficher dans le composant App

```
function App(){  
  const nom='RAMI'  
  const prenom='AHMED'  
  const dateNaissance='23/03/2000'  
  const years = getAge(dateNaissance);  
  return(<div>  
    <h2> Informations </h2>  
    <p> {nom} {prenom}  votre age est:{years}</p>  
    </div>  
  )  
}
```

# Le langage JSX

- **Solution en utilisant classe composante :**

```
default class App extends React.Component{
  constructor(props) {
    super(props)
    this.nom="RAMI";
    this.prenom="AHMED";
    this.dateNaissance="10/03/2000"
  }
  render(){
    return(<div>
      <h2> Informations </h2>
      <p> {this.nom} {this.prenom} votre age est
        {getAge(this.dateNaissance)} ans</p>
    </div>
    )
  }
}
```

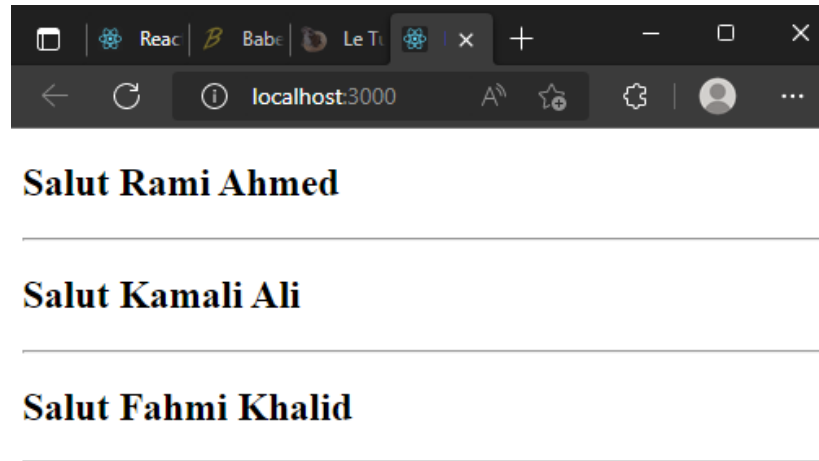
**TP2:**



# Manipuler les propriétés et gérer les états

# Utilisation des propriétés

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

# Utilisation des propriétés

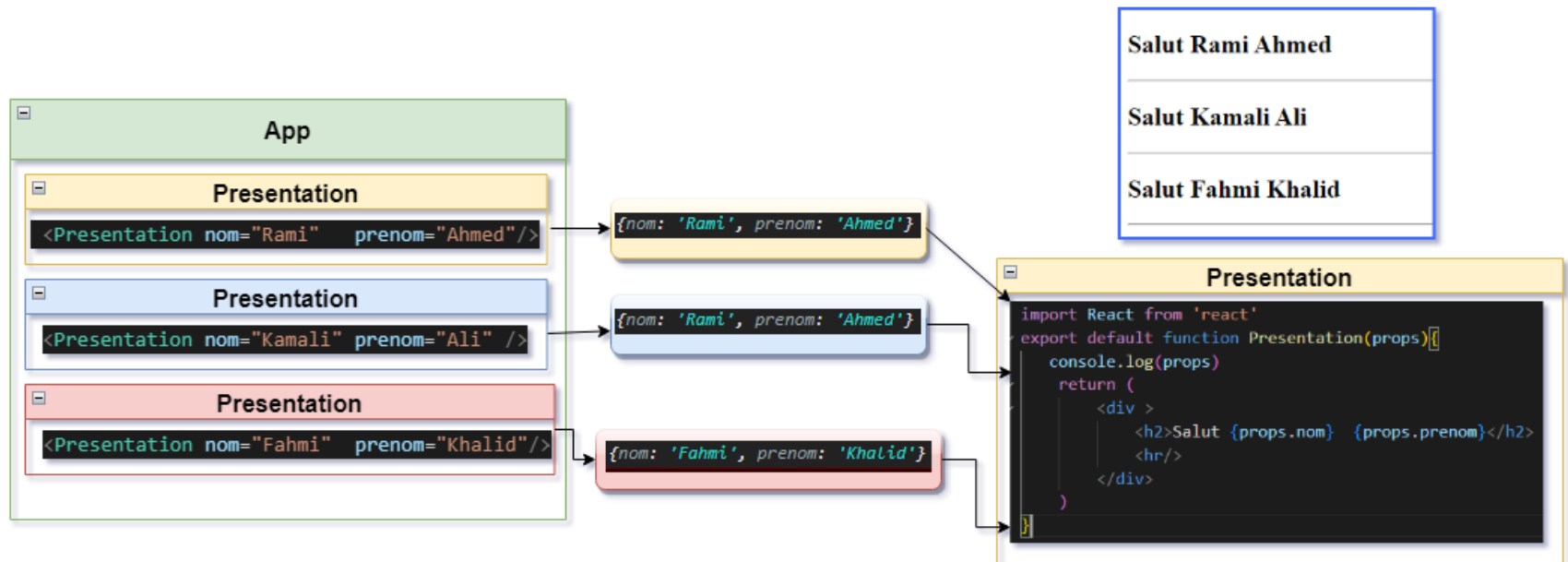
## Presentation.js

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

## 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 />);
```

# Utilisation des propriétés



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>
  ) }
```

```
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>
  ) }
```

# Passer un Array dans props

```
function App(){
  let personne1={nom:"Rami",prenom:"Ahmed"}
  let diplomes=["Bac","Licence","Master"]
  return(
    <div>
      <Presentation personne={personne1}
        diplomes={diplomes} />
    </div>
  )}

```

```
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>
  )}

```

# Passer un Array dans props

```
function App(){
  let personne1={nom:"Rami",prenom:"Ahmed"}
  let diplomes=["Bac","Licence","Master"]
  return(
    <div>
      <Presentation personne={personne1}
        diplomes={diplomes} />
    </div>
  )}

```

```
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>
  )}

```

# Passer un contenu dynamique

```
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>  
  );  
}
```

**Salut Rami Ahmed**

ce ci est un children props

---

**Salut Kamali Ali**

quitter

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



# Etat des composants

- L'état (ou "**state**") est un concept important en React, qui permet aux composants de maintenir leur propre état interne et de modifier leur apparence en fonction de cet état. L'état est utilisé pour stocker des données qui peuvent changer au cours du temps, comme l'état d'une checkbox, le contenu d'un champ de saisie, etc.
- Les composants de classe sont les seuls à pouvoir avoir un état dans React. Pour définir un état initial dans un composant de classe, le constructeur de la classe peut être utilisé :

```
import React from 'react'
export default class Counter extends React.Component{
  constructor(){
    super()

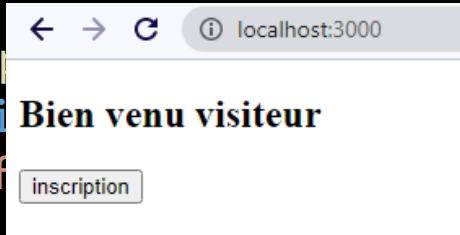
    this.state={count:0};
  }
  render(){
    return(<div> le compteur est à : {this.state.count}
    </div> ;)
  } }
```

# Etat des composants

```
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>)
  }
}
```

# Etat des composants

```
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>)
  }
}
```



# Etat des composants

## regles de state

- 1 → utilisé dans un classe composant
- 2 → Attention de confondre props et state
- 3 → state est un java script objet contenant des données relative au composant
- 4 → la mise à jour de 'state' sur le composant provoque le rendu instantané du composant
- 5 → state doit être initialisé quand le composant est créé
- 6 → state doit être mise à jour uniquement par la fonction **'setState'**