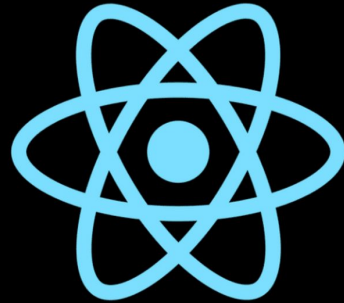


# Développement WEB



**React JS**

# PROGRAMME

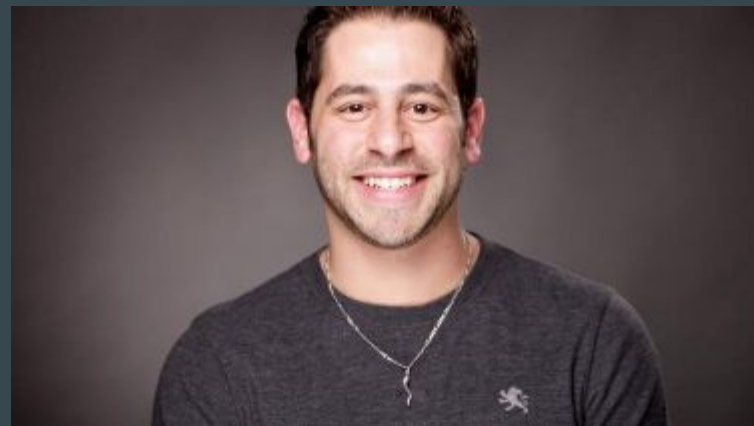
- Histoire
- Qu'est ce que REACT ?
- Fonctionnement
- Fondamentaux
- initialisation
- 1er projet
- JSX
- Composants
- Le state
- Hooks



# Un peu d'histoire ...



créé par Jordan Walke un ingénieur au sein de la société Facebook à la fin de l'année 2011



Pete Hunt, ingénieur travaillant sur Instagram est intéressé par la bibliothèque et assiste Walke



Développé par

Facebook, Instagram et la communauté

Christian DELORME



# Qu'est ce que REACT ?



- REACT est une bibliothèque Javascript utilisant du code “moderne”
- REACT s'utilise côté client (navigateur)(ne gère ni la base de donnée ni le serveur)
- REACT n'utilise pas le système MVC, mais une logique basée sur les composants

# AVANTAGES

- Découpage de l'interface utilisateur en morceaux réutilisables
- Évite de recharger tout le contenu d'une page et permet de ne rafraichir que les composants qui en ont besoin
- Peut être utilisé sur une partie ou sur l'ensemble d'un site web ou application



# Fonctionnement

# Index.js

```
import React from 'react'
import ReactDOM from 'react-dom'
import App from './App'

ReactDOM.render(<App />, document.getElementById('root'))
```



# index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- ..... -->
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
  </body>
</html>
```



# Fondamentaux

Documentation : <https://fr.reactjs.org/docs/getting-started.html>

React developer tool sur chrome : Installer (ajout de components et de profiler )

Dans vs code : installation de Babel javascript / oceanic next



# Initialisation



# 1er projet

Création de notre première application

✓ OPEN EDITORS

✓ TESTRRACT

> node\_modules

✓ public

- ★ favicon.ico
- index.html
- logo192.png
- logo512.png
- manifest.json
- robots.txt





> src

- .gitignore
- package-lock.json
- package.json
- README.md

# Structure

npx create-react-app projet0

✓ OPEN EDITORS

✓ TESTRR...    

> node\_modules

> public

✓ src

- App.css
- App.js
- App.test.js
- index.css
- index.js
- logo.svg
- reportWebVitals.js
- setupTests.js
- .gitignore
- package-lock.json
- package.json
- README.md



# Le JSX

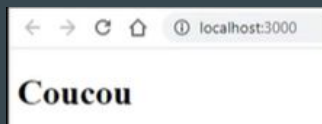


le jsx permet de simplifier l'écriture de code HTML en Javascript:

```
ReactDOM.render(<h1>coucou</h1>, document.getElementById("root"));
```



```
ReactDOM.render(React.createElement("h1", null, "coucou"), document.getElementById("root"));
```





Avantage : on peut écrire des balises HTML dans du code javascript

Suivant les conventions : ex

- class → className
- onclick → onClick

class css

JSX



# Les composants




# les composants simples



# Composant “complexe”

La fonction “render “ ne peut retourner qu’un seul composant (une seul balise)

Un composant peut contenir une seul structure complexe JSX (plusieurs balises)



```
function App() {  
  return (  
    <h1>coucou</h1>  
    <p> Mon prénom est Christian </p>  
  );  
}
```



```
function App() {  
  return (  
    <div>  
      <h1>coucou</h1>  
      <p> Mon prénom est Christian </p>  
    </div>  
  );  
}
```

## Fragmentation

```
import React, { component, Fragment } from "react";

function App() {
  return (
    <Fragment>
      <h1>coucou</h1>
      <p> Mon prénom est Christian </p>
    </Fragment>
  );
}
```



```
import React, { component } from "react";

function App() {
  return (
    <>
      <h1>coucou</h1>
      <p> Mon prénom est Christian </p>
    </>
  );
}
```

Nécessite une version récente de REACT



# Des composants dans des composants

Le composant App contient le composant Personne

```
class App extends component {  
  render() {  
    return (  
      <>  
        <Personne />,  
        <Personne />,  
        <Personne />  
      </>  
    );  
  }  
}
```



# Passage d'information a un composant

```
const personne = props => {  
  return (  
    <>  
      <h1> {props.nom} </h1>  
      <div>Age : {props.age}</div>  
      <div>Sex : {props.sex} </div>  
    </>  
  );  
}
```



```
class App extends component {  
  render() {  
    return (  
      (  
        <>  
          <Personne nom="joe" age="31" sex="homme " />,  
          <Personne nom="janne" age="25" sex="femme " />,  
          <Personne nom="William" age="43" sex="homme " />  
        </>  
      )  
    );  
  }  
}
```





# Les classes

```
class Personne extends Component {  
  render() {  
    return (  
      <>  
        <h1> {props.nom} </h1>  
        <div>Age : {props.age}</div>  
        <div>Sex : {props.sex} </div>  
      </>  
    );  
  }  
};
```



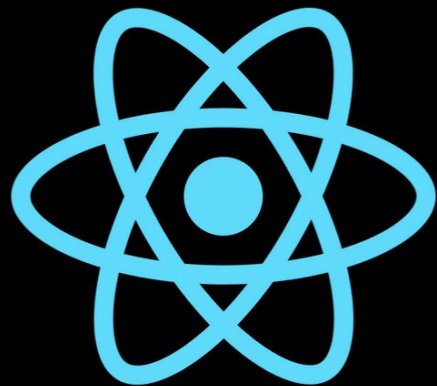
# Synthèse



- **Composant** : un composant permet de regrouper du code JSX réutilisable à plusieurs endroits dans une application
- Deux écritures :
  - **Sous forme de fonction (fléchée)** : composants basiques sans logique intérieur (stateless)
  - **Sous forme de class** : composant complexe manipulant des states, fonction , autre composants (statefull)
- Pour passer des informations aux “components”, il faut utiliser le système de propriétés “ props ”.
  - Fonctions fléchées : **props.nomdePropriété**
  - Classes : **this.props.nomdePropriété**



# le states



**React JS**  
*State*

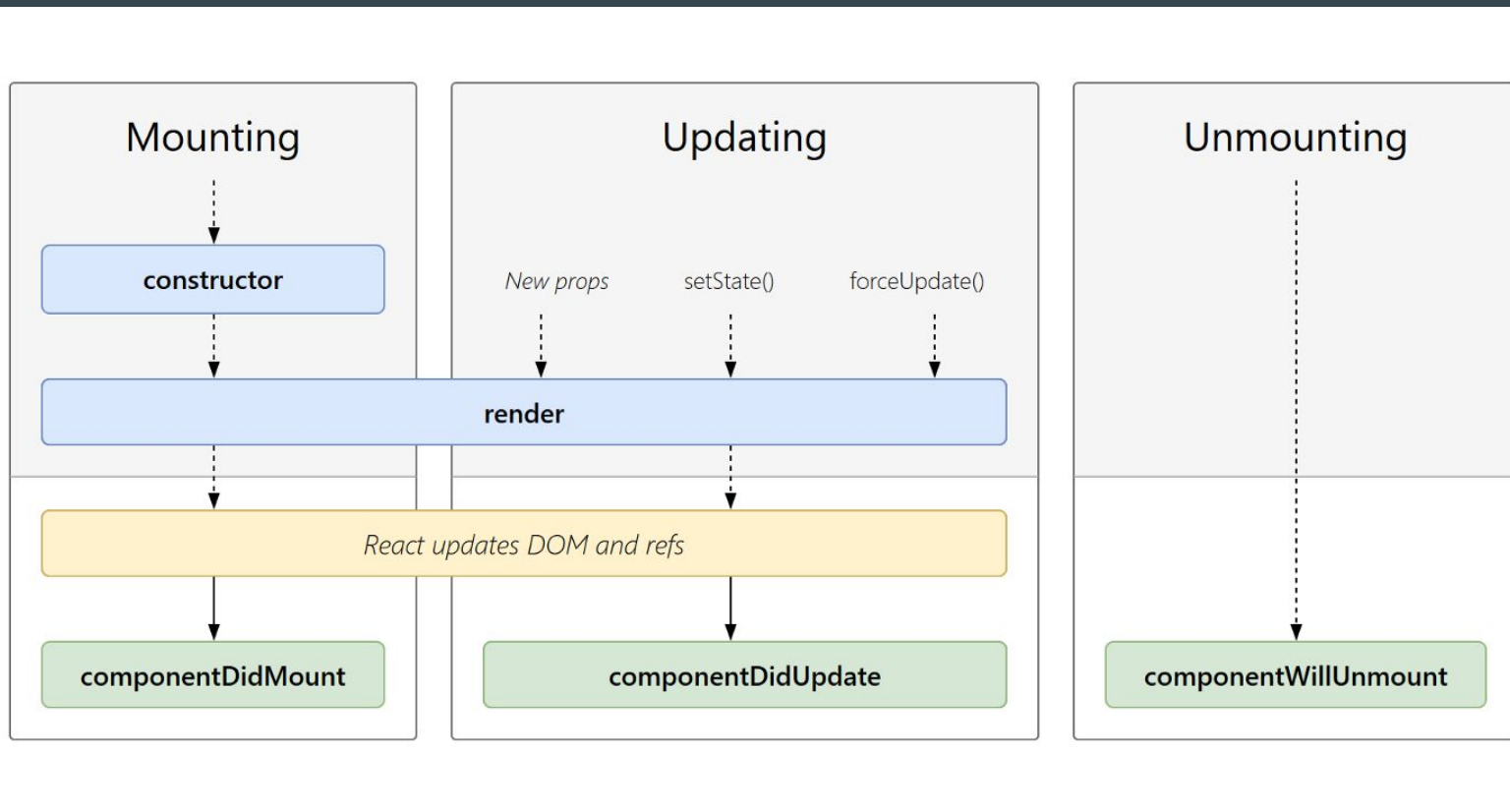


# Principe

- Pour modifier des informations, on passe par une classe et un attribut spécial “**states**” prévu par la classe “ Component ”.
- Rappel : il n’est pas possible de modifier des “ props ”.
- pour bien comprendre le fonctionnement, nous allons mettre en place une horloge



# Cycle de vie



# state

```
class Horloge extends Component {  
  state = {  
    date: new Date(),  
  };  
}
```

# setState

```
class Horloge extends Component {  
  state = {  
    date: new Date(),  
  };  
  componentDidMount() {  
    setInterval(() => this.setState({ date: new Date() }), 1000);  
  }  
  
  render() {  
    console.log("composant affiché");  
    return <h2>Horloge : {this.state.date.toLocaleTimeString()} </h2>;  
  }  
}
```





# Synthèse



- **State** : les “states” permettent de stocker, véhiculer et manipuler des données dans une application REACT au travers de différents composants.
- **setState** : la fonction setState permet de mettre à jour les données dans le “state”, c’est l’écriture à utiliser pour s’assurer que les composants impactés soient rafraîchis.
- **Cycle de vie** : de fonctions sont disponibles pour suivre l’évolution d’un composant:
  - **componentDidMount** : lancé après le montage
  - **componentDidUpdate** : lancé après une mise à jour
  - **componentWillUnmount** : lancé avant le “démontage”

# Événement et state



# React les Hooks

...



# Qu'est ce qu'un Hook ?

Un hook est une fonction qui permet de se “brancher sur des fonctionnalités Réact”.

ex: useState est un Hook qui permet d'ajouter l'état local React à des fonctions composants

<https://fr.reactjs.org/docs/hooks-intro.html>



# Quand utiliser un Hook ?

Les Hooks sont des fonction Javascript, mais ils imposent 2 règles supplémentaires:

Appeler les Hooks uniquement au niveau racine . N'appellez pas Hooks à l'intérieur de boucles , de code conditionnel ou de fonctions imbriquées.

Appeler les Hooks uniquement depuis des fonctions composants React .

N'appellez pas les Hooks depuis des fonctions Javascript classique .



# useState

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

```
function Test() {  
  const state = useState(0);  
  return (  
    <div>  
    </div>  
  )  
}
```

destructuring



```
function Test() {  
  const [age, setAge] = useState(18);  
  return (  
    <div>  
    </div>  
  )  
}
```

`useState` retourne un tableau qui retourne 2 choses :

1. la valeur de départ (ici 0).
2. et une fonction => (en réalité le `setState` des classes), qui permet de modifier cette valeur





# useState et les objets

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

function Test() {
  const [personne, setPersonne] = useState({
    age: 18,
    sexe: true,
    couleur: "noire"
  })

  return (
    <div>
    </div>
  )
}
```

# useEffect

permet de gérer les 3 fonctions du cycle de vie:

`componentDidMount`

`componentDidUpdate`

`componentWillUnmount`

Lorsque vous appelez `useEffect`, vous dites à React de lancer votre fonction d'effet après qu'il ait mis à jour le DOM.

Les effets étant déclarés au sein du composant, ils ont accès à ses props et son état.

Par défaut, React exécute les effets après chaque affichage, y compris le premier

Permet de gérer les effets de bord

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

```
function Test() {  
  
  const [cpt, setCpt] = useState(0)  
  
  useEffect(() => {  
    const timer = setInterval(() => {  
      setCpt(cpt => cpt + 1)  
    }, 1000);  
  
    return function () {  
      clearInterval(timer)  
    }  
  }, [])  
  
  return (  
    <div>  
      timer : {cpt}  
    </div>  
  )  
}
```

dépendance : permet d'éviter la répétition lors de la modification du composant

dans cette exemple permet de lancer le timer uniquement au montage du composant

fonction appelé au démontage du composant

On peut utiliser autant de Useffect que nécessaire

React recommande d'utiliser un useEffect par fonctionnalité différentes.