

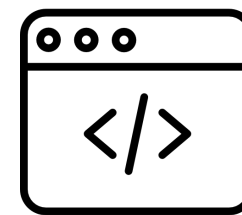
REACT.JS

Module 28h

Présenté & écrit par Jeremy Nohile

MEET MR. Jeremy Nohile

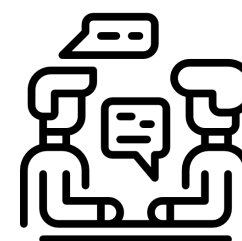
Co-Dirigeant NJG Connect



Développement



Design



Conseil & Expertise

PROGRAMME

REACT.JS

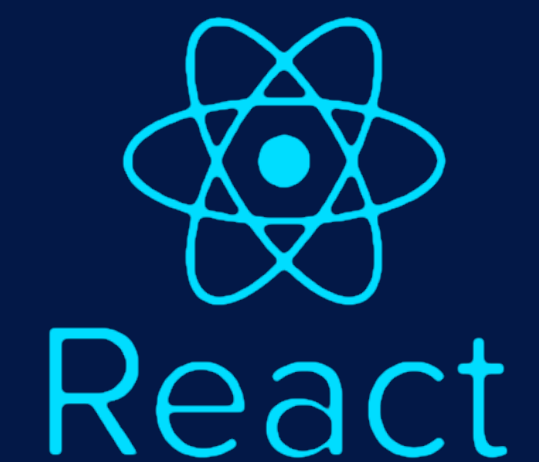
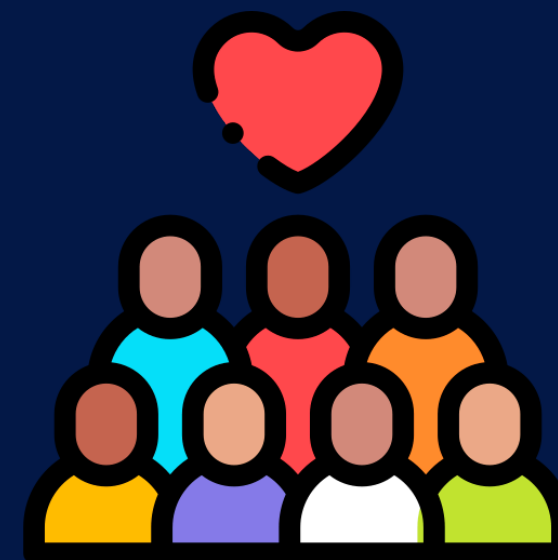


React

c'est quoi?

C'est une **Librairie** JavaScript
développé par Facebook depuis 2013

Objectif : Construction d'interfaces utilisateur



JSX

Extension syntaxique de JavaScript

« Un semblant de HTML »

Facilite l'écriture de ces structures.

« HTML in JS »

La syntaxe `<div />` est transformée à la compilation en `React.createElement('div')`



```
1 // Avec JSX
2
3 const element = () => (
4   <div>
5     <h2>Titre</h2>
6     <p>Contenu</p>
7   </div>
8 )
9
10 // Sans JSX
11
12 React.createElement(
13   'div',
14   null,
15   React.createElement('h2', null, 'Titre'),
16   React.createElement('p', null, 'Contenu')
17 )
18
```

Comment ça fonctionne ?

Index.js le point d'entrée de notre App

2 Packages Important:

- react... pour faire du React
- React-dom => render notre App

```
1 import React from 'react'
2 import ReactDOM from 'react-dom'
3 import App from './App'
4
5 ReactDOM.render(<App />, document.getElementById('root'))
```

Render va ajouter notre appli sur l'élément du DOM avec l'id root
(public/index.html)

Attention



Toutes les propriétés & attributs sont en camelCase

Puisque `for` est un mot réservé en JavaScript, les éléments React utilisent plutôt `htmlFor`.

Doublon dans le DOM il faut rajouter un attribut `key` mais attention à ne pas mettre l'index si vous jouez avec avec la position des éléments

L'attribut `style` accepte un objet JavaScript avec des propriétés en camelCase

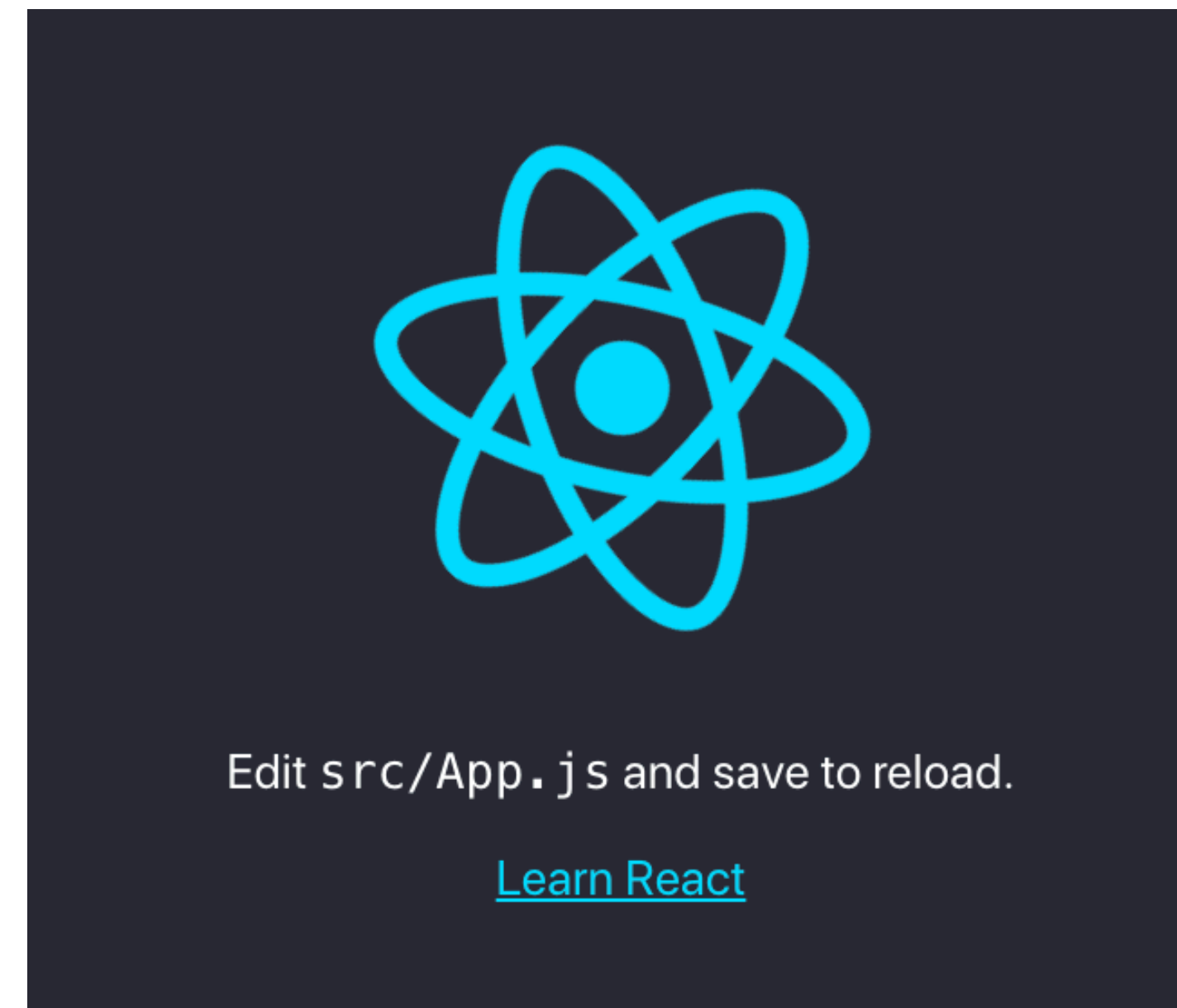
sans react	avec react
class	className
onclick	onClick
for	htmlFor

```
1 const divStyle = {
2   color: 'blue',
3   backgroundImage: 'url(' + imgUrl + ')',
4 }
5
6 function HelloWorldComponent() {
7   return <div style={divStyle}>Bonjour, monde !</div>
8 }
9
```

Hello world



```
npx create-react-app mon-app  
cd mon-app  
npm start
```



Remarque : npx sur la première ligne n'est pas une faute de frappe
c'est un exécuter de paquets qui est inclus dans npm 5.2+.

Components



Un objectif Par composant

Exemple : Un bouton, un input...



Réutilisable

Un composant peut-être utilisé plusieurs fois même si ses paramètres changent



Test plus simple

Les tests sont isolés et plus facile à réaliser



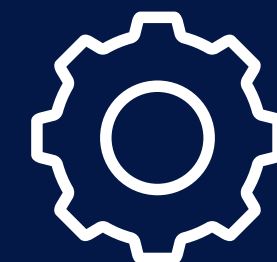
Visibilité plus claire

Permet de savoir l'intérêt du composant uniquement par son nom



Coder moins

l'objectif est de supprimer les doublons.
Et ainsi d'éviter de se répéter.



Maintenabilité simplifiée

Simple pour fixer ou améliorer son composant

Classe & Fonction

Classe

```
1 import React, { Component } from "react";
2
3 class ClassComponent extends Component {
4   render() {
5     return <h1>Class Component</h1>;
6   }
7 }
8
```

Fonction

```
1 import React from 'react'
2
3 function FunctionalComponent() {
4   return <h1>Functional Component</h1>
5 }
6
```

Prend la relève

React améliore davantage cette pratique

Props

Les props sont les paramètres du composant
(c'est la contraction de « propriétés »)

Props == paramètres == propriétés

```
1 const React from "react"
2
3 const User = (props) => <p>{props.name}</p>;
4
5 const HelloUser = () => {
6   return (
7     <div>
8       <User name="Fred" />
9     </div>
10  );
```

State | État Local

Les states == l'état d'un composant

Inaccessible pour les autres composants.

Quand l'état change, le composant répond en se rafraîchissant.



```
1 import React, { useState } from 'react';
2
3 function Example() {
4   // Déclare une nouvelle variable d'état, que l'on va appeler « count »
5   const [count, setCount] = useState(0);
6   return (
7     <div>
8       <p>Vous avez cliqué {count} fois</p>
9       <button onClick={() => setCount(count + 1)}>
10         Cliquez ici
11       </button>
12     </div>
13   );
14 }
```

Bouton

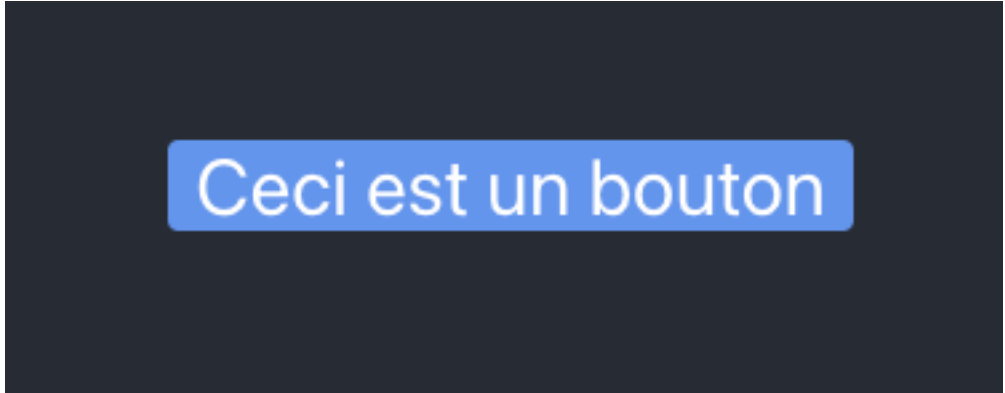


1°) Créer un Component Button.js

Props :

- value
- onClick

Comportement de Base



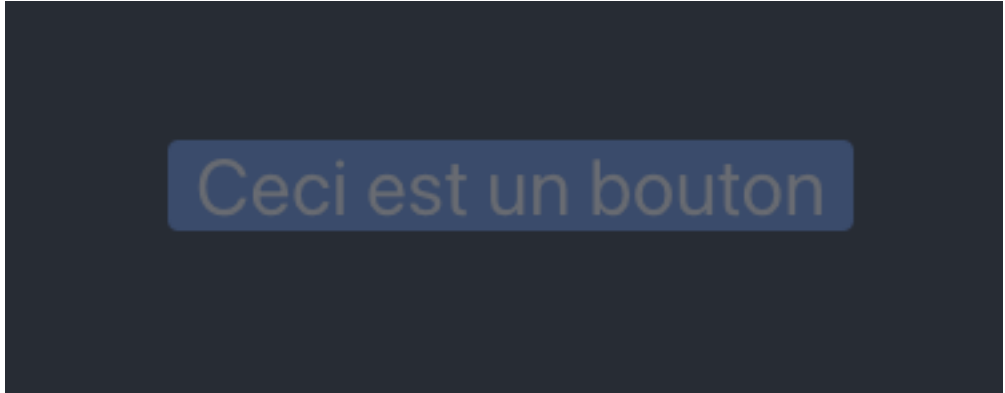
Ceci est un bouton

2°) Le styliser

Style demandé

- Couleur de fond
- Au survol de la souris grisé ou changer de couleur

Comportement au Survol



Ceci est un bouton

Changement d'icône

1°) Afficher une image avec l'api avatars.dicebear

Url de l'image :

<https://avatars.dicebear.com/api/adventurer-neutral/example.svg>

2°) Styliser l'image SVG sans utiliser l'attribut className

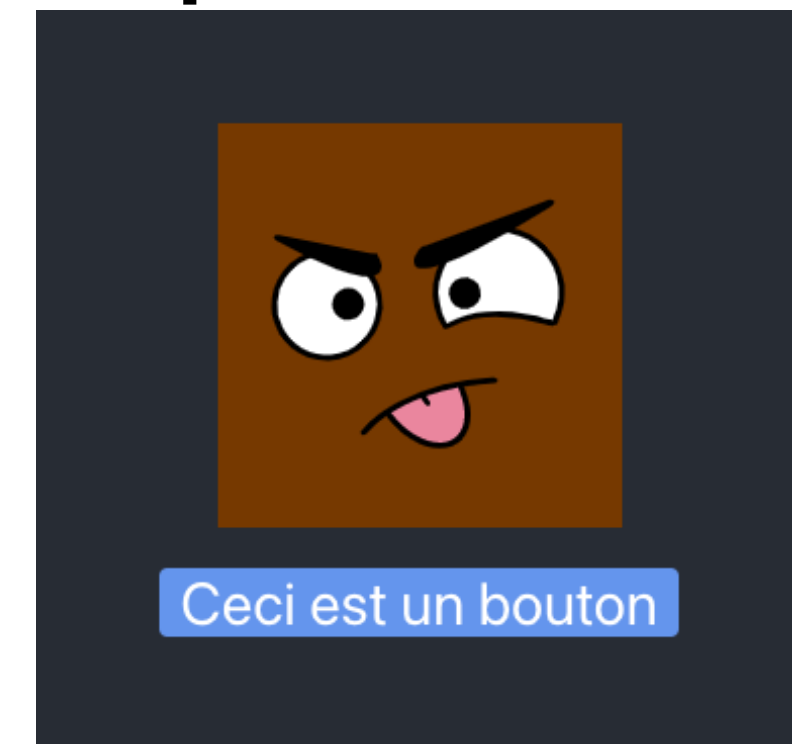
Style demandé

- width : 200
- height: 200
- margin du bas 20

3°) Utiliser le bouton pour trigger une valeur random

Cette valeur random va changer la valeur « [exemple](#) » de notre image

Comportement initial



Après l'appuis du bouton



État & Cycle de vie

Classe

```
1 componentDidMount()  
2 // S'exécute quand le composant est monté  
3  
4 componentWillUnmount()  
5 // S'exécute quand le composant va être retiré de DOM  
6
```

Fonction

```
1 useEffect(() => {  
2   // Anything in here is fired on component mount.  
3   return () => {  
4     // Anything in here is fired on component unmount.  
5   }  
6 }, [])  
7
```

Mounting



Updating



Unmounting



Appel API



1°) useState()

- Représente la Data qu'on va récupérer
- Son état est undefined au depart

2°) useEffect()

- S'appelle quand le composant se monte
- Encapsuler d'une IIFE (Immediately Invoked Function Expression)
- Récupérer la Data et la mettre dans notre State

3°) fetch()

- Function pour récupérer les informations d'une API tiers

```
1 import { useEffect, useState } from 'react'
2
3 function App() {
4   const [data, setdata] = useState(undefined)
5   useEffect(() => {
6     (async () => {
7       const newData = await getData()
8       setdata(newData)
9     })()
10    return () => {}
11  }, [])
12
13  const getData = async () => {
14    const dataJson = await fetch('https://randomuser.me/api/')
15    return await dataJson.json()
16  }
17
18  if (!data) {
19    return <p>waiting</p>
20  }
21  return <p>{data.results[0].name.first}</p>
22 }
23 export default App
24
```


chifoumi 🤘🤚👊



1°) Afficher les 3 symboles

Tips:

- utiliser un map()

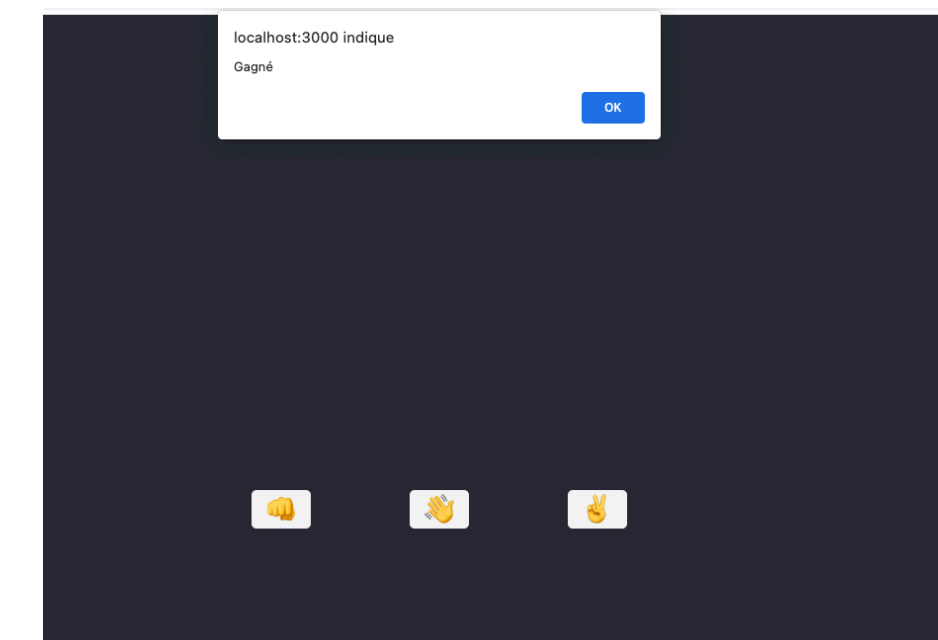
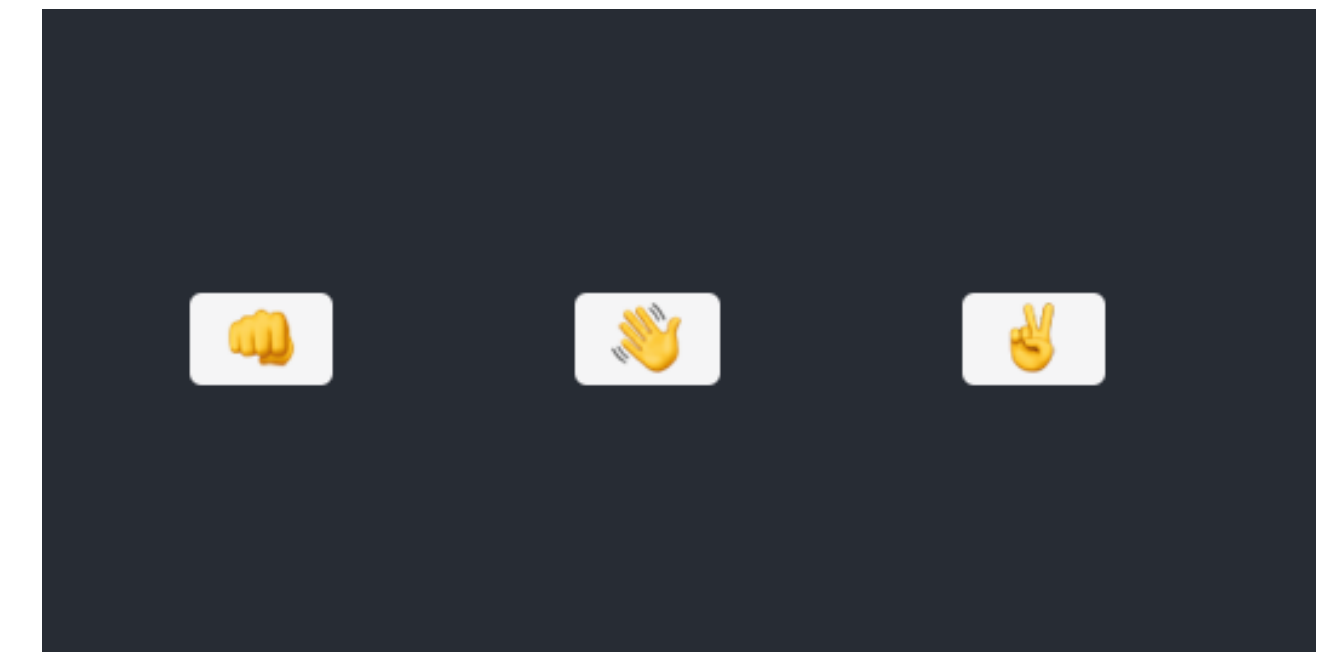
2°) Ajouter la logique du chifoumi

L'algorithme determine un choix aléatoire

3°) Afficher si tu as gagné ou Perdu

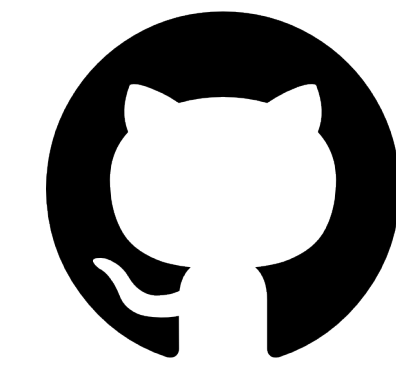
Affiché avec la méthode alert()

Comportement initial



Après l'appui d'un choix

Ajouté des packages



Ajouter des packages devient simple

Multitudes de Component disponible

Gain de temps

Attention à ne pas ajouter tout et n'importe quoi !!

1°) Ajout d'un package

```
$ yarn add react-toastify
```

1°) Utilisation d'un package

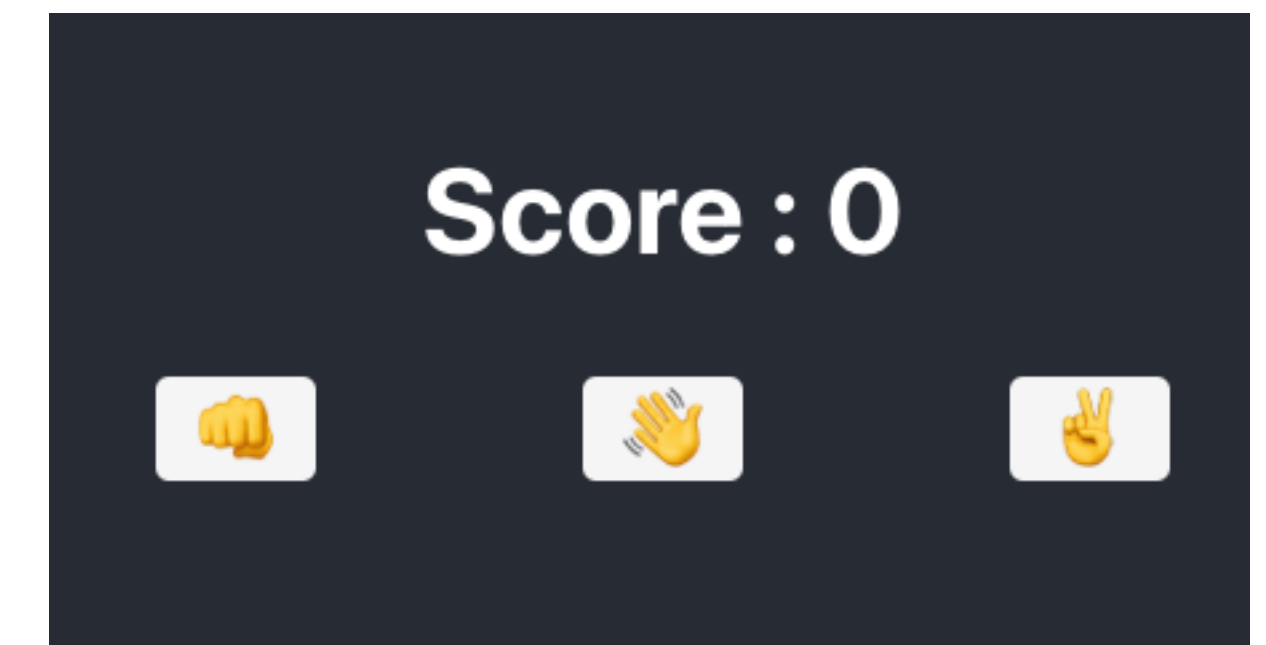
```
1 import React from 'react'
2
3 import { ToastContainer, toast } from 'react-toastify'
4 import 'react-toastify/dist/ReactToastify.css'
5
6 function App() {
7   const notify = () => toast('Wow so easy!')
8
9   return (
10     <div>
11       <button onClick={notify}>Notify!</button>
12       <ToastContainer />
13     </div>
14   )
15 }
16
```

chifoumi + 🤞🤚👊



1°) Rajouter un titre 2 avec le score de victoire

Comportement initial



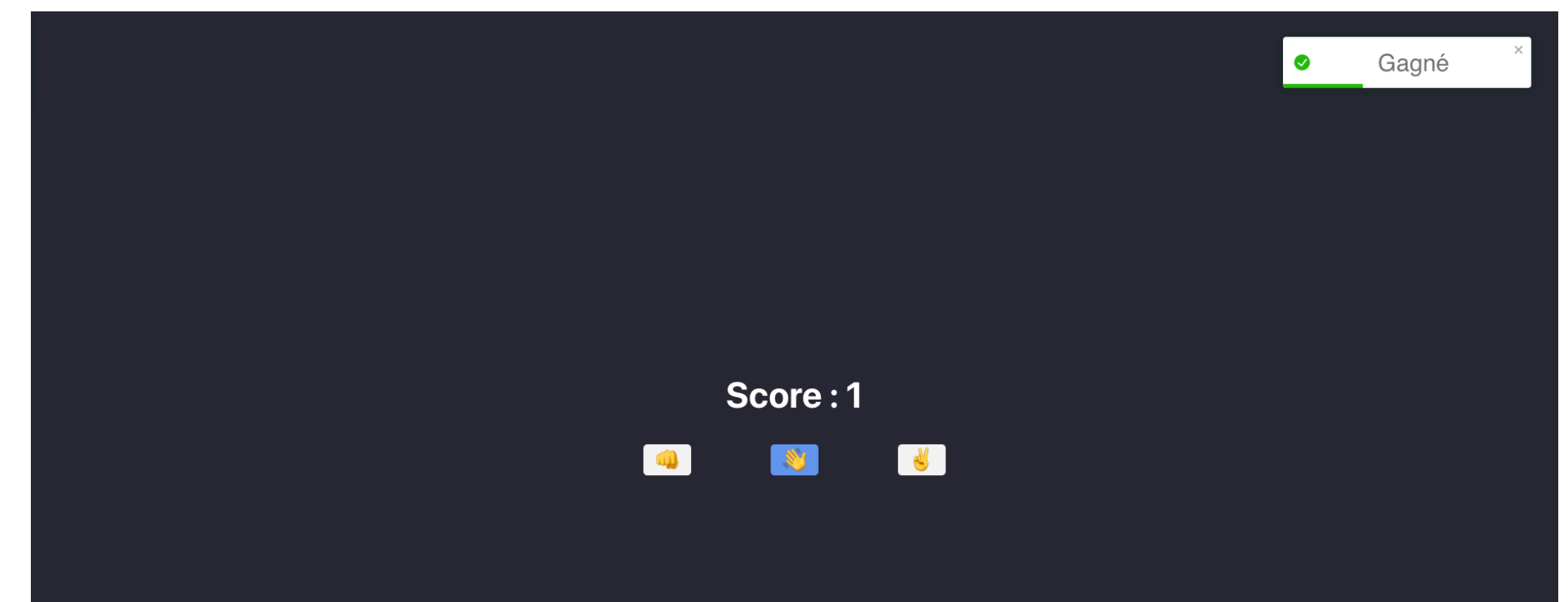
2°) Installer la lib de toast (react-toastify)

3°) Remplacer la méthode alert par des toasts

Afficher le toast :

- Rouge = Perdu
- Vert = Gagné
- Couleur de ton Choix = Égalité

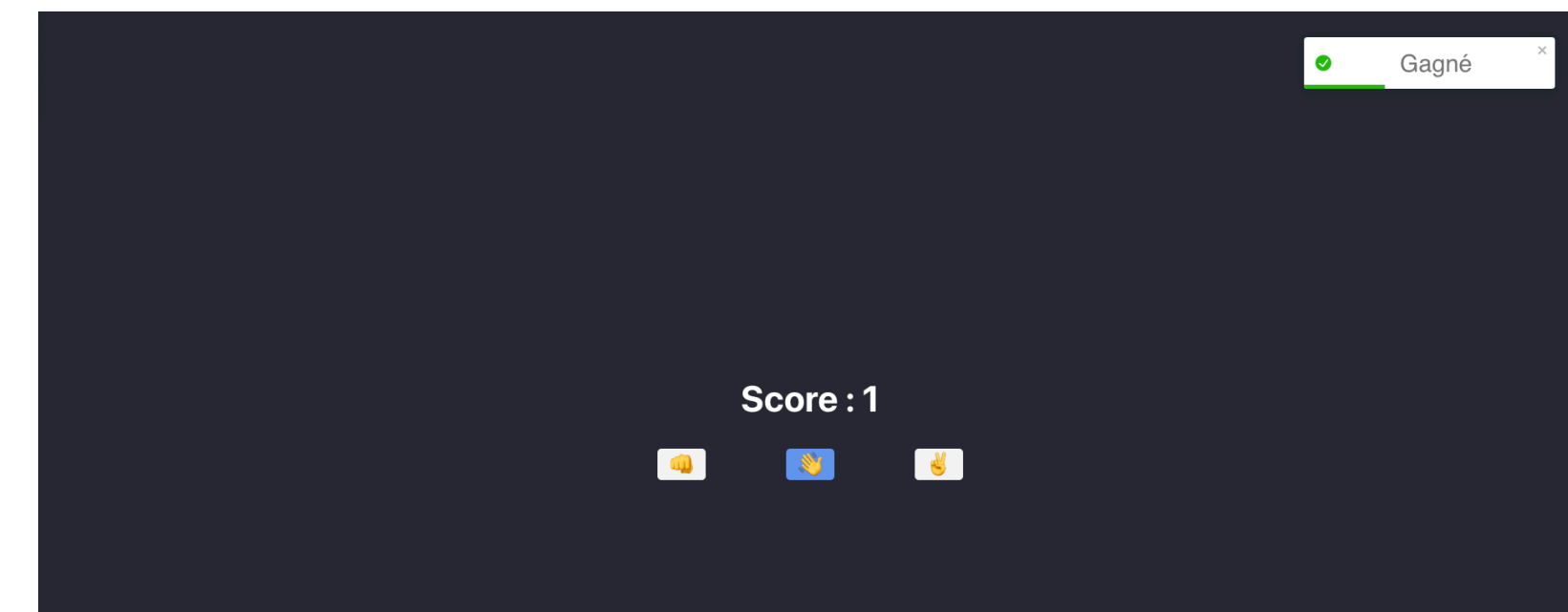
Après l'appui d'un choix



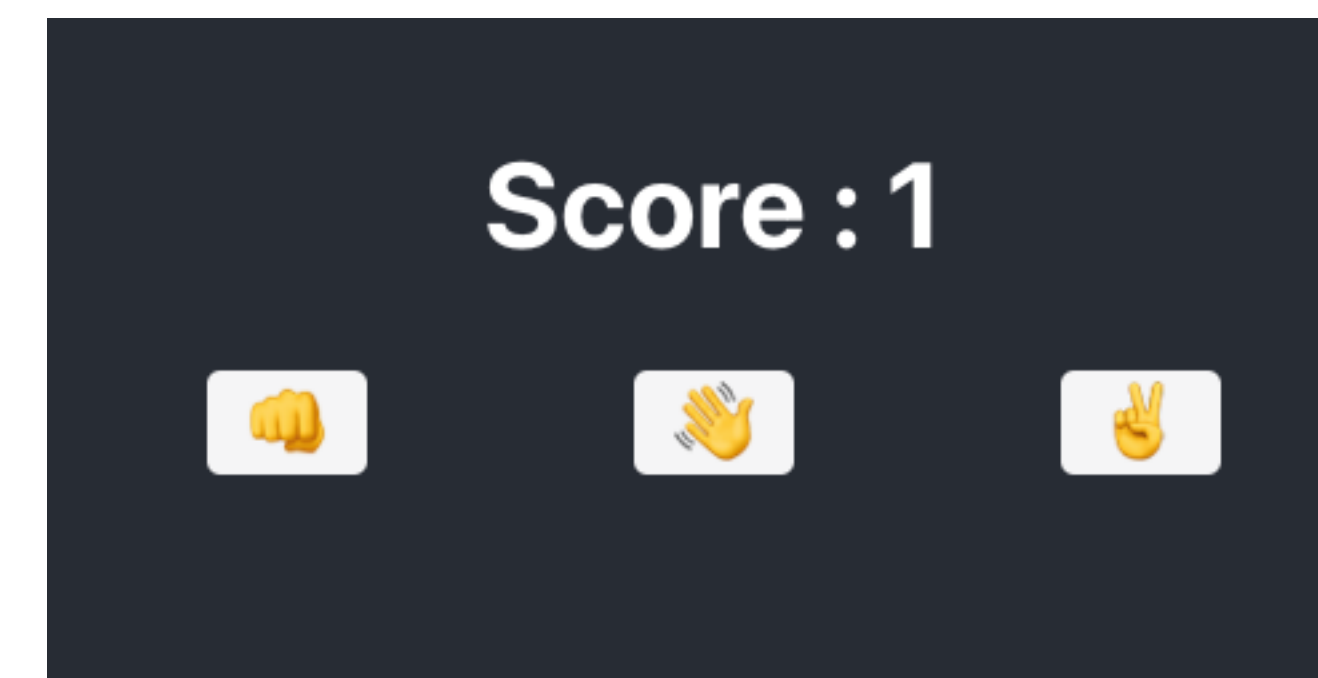
chifoumi ++



Comportement initial



Après Refresh du Navigateur



1°) Stocker le nombre de victoire en storage

2°) Quant-on rafraichi la page le score est gardé