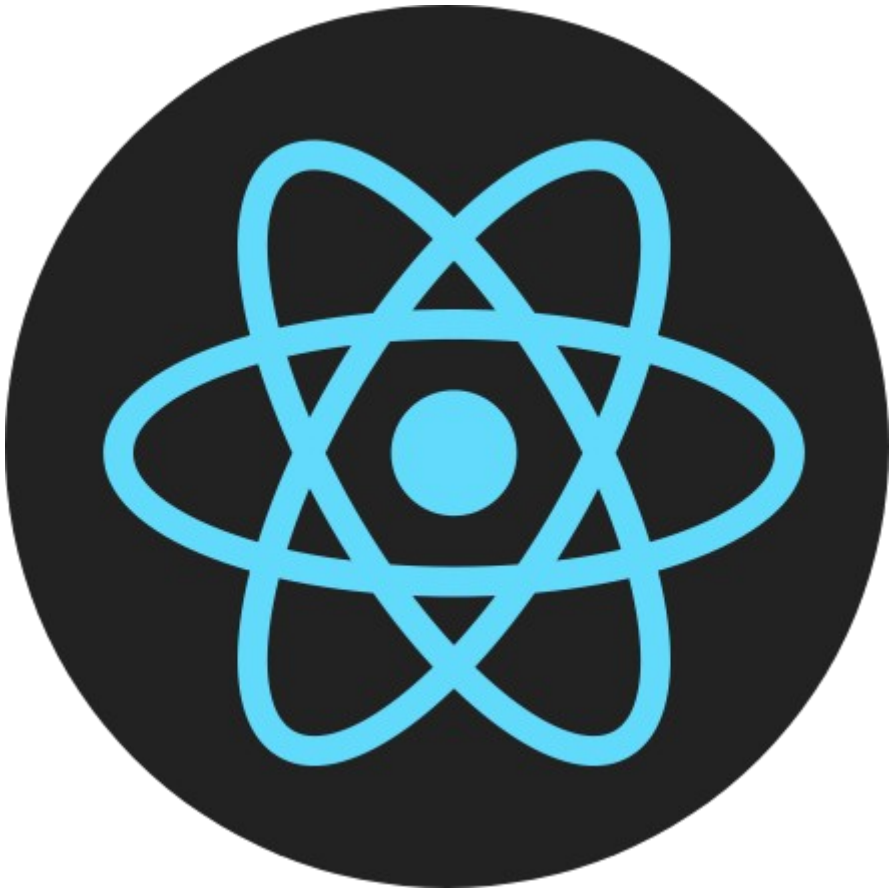
# Projet 11 Découverte de la librairie ReactJS (JavaScript)



## Définition :

***React*** (aussi appelé **React.js** ou **ReactJS**) est une bibliothèque JavaScript libre développée par Facebook depuis 2013. Le but principal de cette bibliothèque est de faciliter la création d'application web monopage, via la création de composants dépendant d'un état et générant une page (ou portion) HTML à chaque changement d'état.

*ReactJS* est une bibliothèque qui ne gère que l'interface de l'application, considéré comme la vue dans le modèle MVC. Elle peut ainsi être utilisée avec une autre bibliothèque ou un framework MVC comme AngularJS. La bibliothèque se démarque de ses concurrents par sa flexibilité et ses performances, en travaillant avec un DOM virtuel et en ne mettant à jour le rendu dans le navigateur qu'en cas de nécessité.

La bibliothèque est utilisée par Netflix (une migration de la partie client vers du JavaScript pur a permis d'augmenter les performances de 50%), Yahoo, Airbnb, Sony, Atlassian ainsi que par les équipes de Facebook, pratiquant l'autoéquipement sur le réseau social éponyme, Instagram ou encore WhatsApp. À la fin de 2015, WordPress.com annonce Gutenberg, une interface pour les éditeurs de sites WordPress, développée en JavaScript avec Node.js et React

## Historique :

React est 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 afin de retirer les portions dépendantes de Facebook. Ceci permet à React d'être publié sous licence Apache 2.0 le 29 mai 2013. React s'inspire de XHP (en), une bibliothèque également développée par Facebook, permettant l'inclusion de HTML au sein de PHP.

En octobre 2014, la version 0.12.0 est publiée sous licence BSD modifiée, avec une note associée PATENTS permettant l'utilisation des brevets de Facebook associé à React. Cependant, la licence BSD est mise à jour en avril 2015, avec la version 0.13.1, pour éviter les confusions.

La dernière version de React est 18.2.0(14 juin 2022), elle met de coté les classes (POO Javascript) au profit des hooks (programmation fonctionnelle)

## Fonctionnalités :

React a été conçu comme étant une bibliothèque et non un framework MVC, comme peuvent l'être ses concurrents. Ainsi, React encourage la création de composants réutilisables, avec en entrée des données, pouvant changer au cours du temps

Par ailleurs, React n'utilise pas de système de templates et ne fonctionne qu'avec du JavaScript, permettant une insertion complète du composant au sein d'une unique classe ou de fonction  hooks. Pour faciliter l'écriture de la vue, l'équipe initiale chez Facebook a développé un langage, JSX, qui permet de générer des objets Javascript avec une notation similaire à HTML.

## Un DOM Virtuel :

Un DOM Virtuel est une représentation du DOM en JavaScript. Au lieu de générer le DOM lui-même comme avec un langage de templating, c'est-à-dire au lieu de dialoguer avec les API du navigateur pour construire le DOM, on ne génère qu'une arborescence d'objets JavaScript en mémoire.

## React Native pour des applications Hybrides multiplatformes (Windows, Linux MacOs)

En 2015 React Native fait son apparition. Ce framework est basé sur React et permet de créer toujours en Javascript des applications multi-plateformes Android et iOS.

ReactJS fonctionne en créant des “components”, correspondant à un composant dans votre page HTML. Voici un exemple de création de composant :

Quelque chose peut vous surprendre : on écrit du XML (HTML) dans du JavaScript ! Il s’agit du langage JSX , pour “JavaScript Xml”. Cela nous permet de gérer le rendu de notre composant, dans la méthode “render” en y injectant directement le HTML que nous voulons voir apparaître. On peut bien entendu se passer de cette notation, qui peut déstabiliser, en utilisant du JavaScript pur :

L’intérêt de ReactJS est que ses composants correspondent à un élément de la page, avec son état propre et ses actions associées. Le tout géré directement à l’intérieur du composant lui-même ! Ils ont leur propre état, leurs propres données et sont

indépendants les uns des autres. Autant dire que créer une bibliothèque en React correspond simplement à créer un composant auquel on peut donner une configuration !

### Différentes méthodes et champs sont disponibles pour créer un composant :

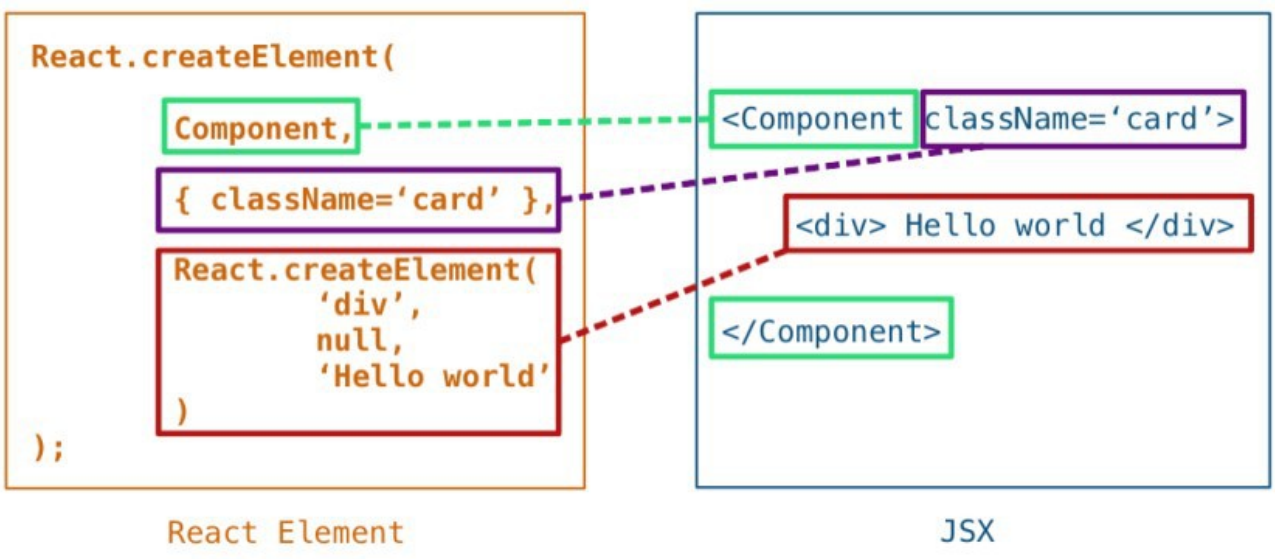
- **props** : ce sont les propriétés qui sont données en paramètre du composant. Il s'agira souvent d'une configuration.
- **state** : contient l'état interne du composant. Celui-ci peut être la liste des éléments à afficher, une valeur ou une configuration.
- **getInitialState** : méthode permettant d'initialiser le "state" interne du composant. La méthode retourne un objet qui sera ensuite mappé sur this.state.
- **setState** : méthode permettant simplement de mettre à jour le "state". Elle est principalement appelée de l'extérieur pour mettre à jour les données.
- **componantDidMount** : appelée lors de la création du composant, elle permet d'initialiser des méthodes et données.
- **componantWillUnmount** : Appelé lors de la destruction du composant. Peut être utilisé pour transmettre des données lors de cette phase.
- **render** : se charge de l'affichage du composant (comme vu auparavant).

Il existe encore d'autres méthodes, mais ce sont ici les principales. On peut alors créer des composants plus compliqués, comme une ToDoList qui garde la liste des todos dans son état interne

### LE JSX (Javascript extension XML)

Le JSX est une extension syntaxique de JavaScript. Nous recommandons de l'utiliser 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.

Au lieu de séparer artificiellement les *technologies* en mettant le balisage et la logique dans des fichiers séparés, React sépare les *préoccupations* via des unités faiblement couplées appelées « composants », qui contiennent les deux.



## Exemple :

```
const name = 'Clarisse Agbegnenou';const element = <h1>Bonjour, {name}</h1>;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

## Le rendu des elements :

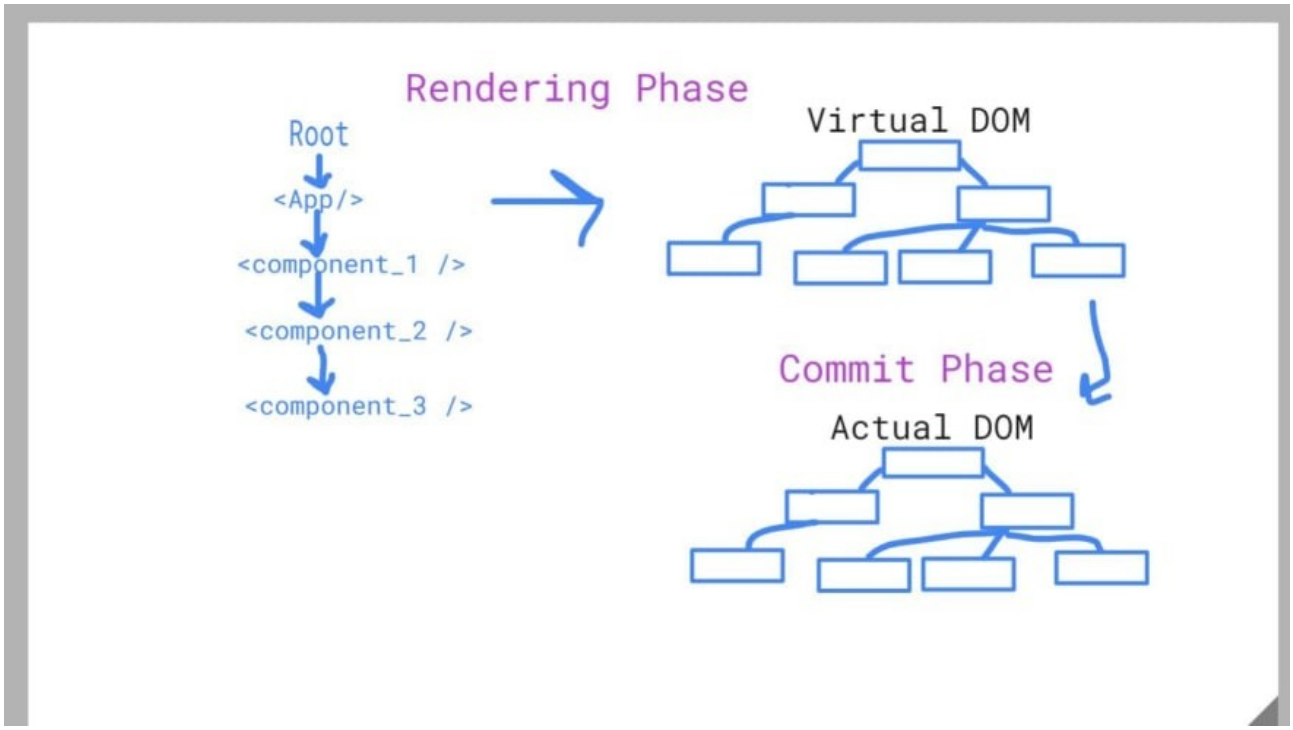
Contrairement aux éléments DOM d'un navigateur, les éléments React sont de simples objets peu coûteux à créer. React DOM se charge de mettre à jour le DOM afin qu'il corresponde aux éléments React.

Nous parlons de nœud DOM « racine » car tout ce qu'il contient sera géré par React DOM.

Les applications développés uniquement avec React ont généralement un seul nœud DOM racine. Si vous intégrez React dans une application existante, vous pouvez avoir autant de nœuds DOM racines isolés que vous le souhaitez.

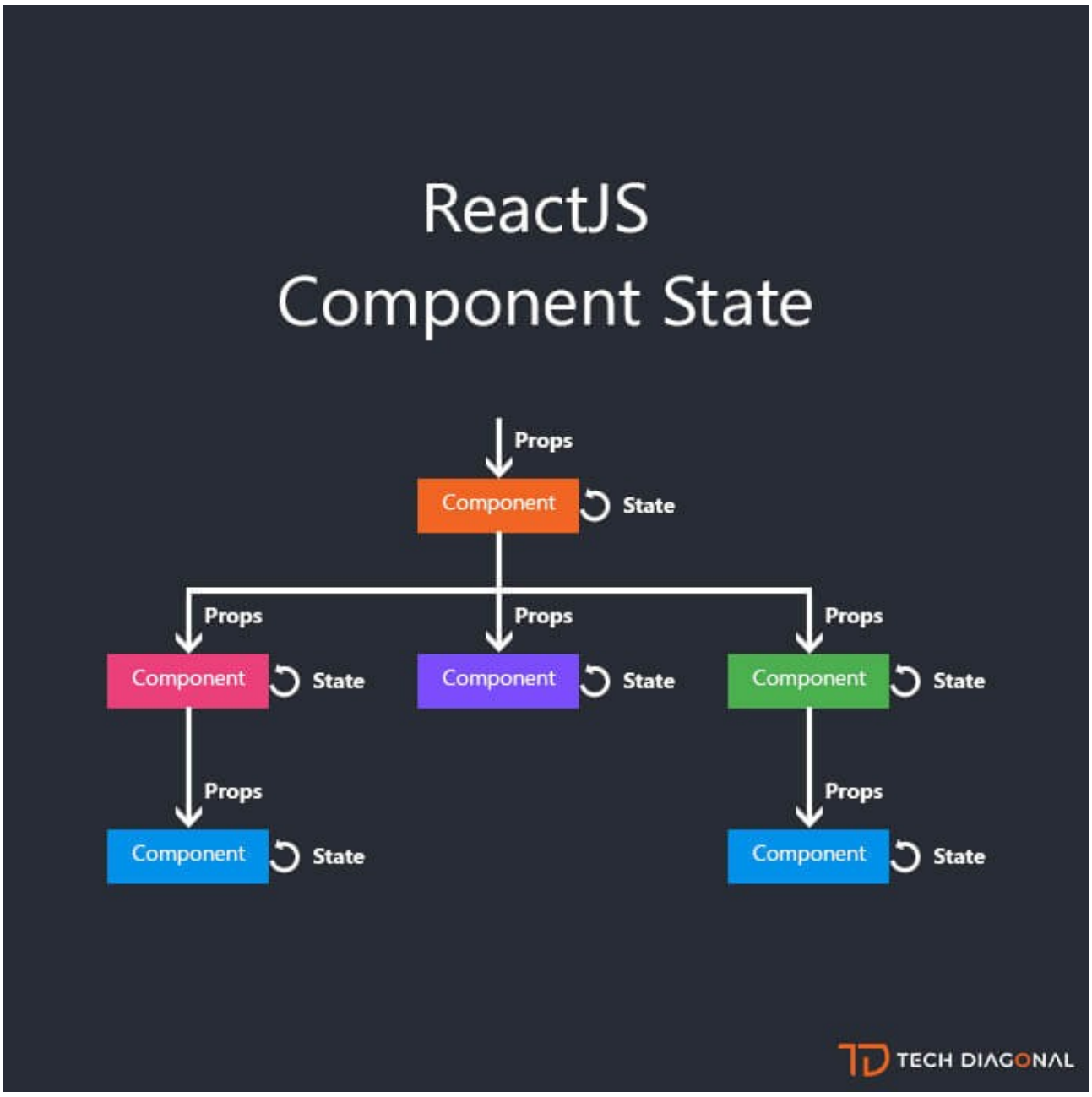
Pour faire le rendu d'un élément React dans un nœud DOM racine, passez les deux à la méthode ReactDOM.render() :

Les éléments React sont immuables. Une fois votre élément créé, vous ne pouvez plus modifier ses enfants ou ses attributs. Un élément est comme une image d'un film à un instant T : il représente l'interface utilisateur à un point précis dans le temps.



## Les composants et les props (properties)

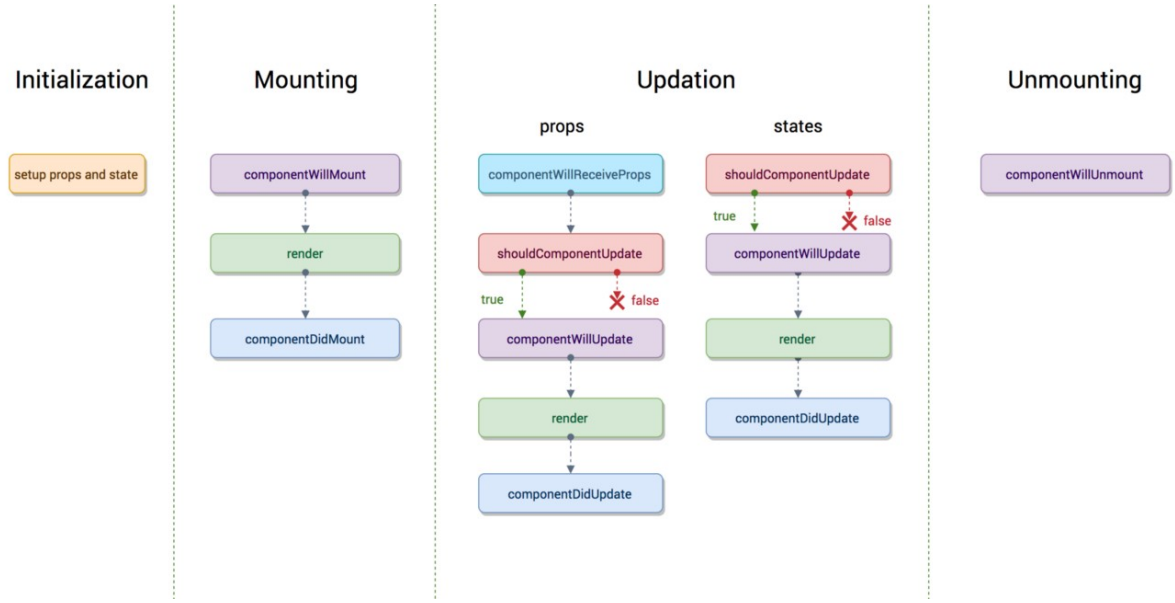
Conceptuellement, les composants sont comme des fonctions JavaScript. Ils acceptent des entrées quelconques (appelées « props ») et renvoient des éléments React décrivant ce qui doit apparaître à l'écran.



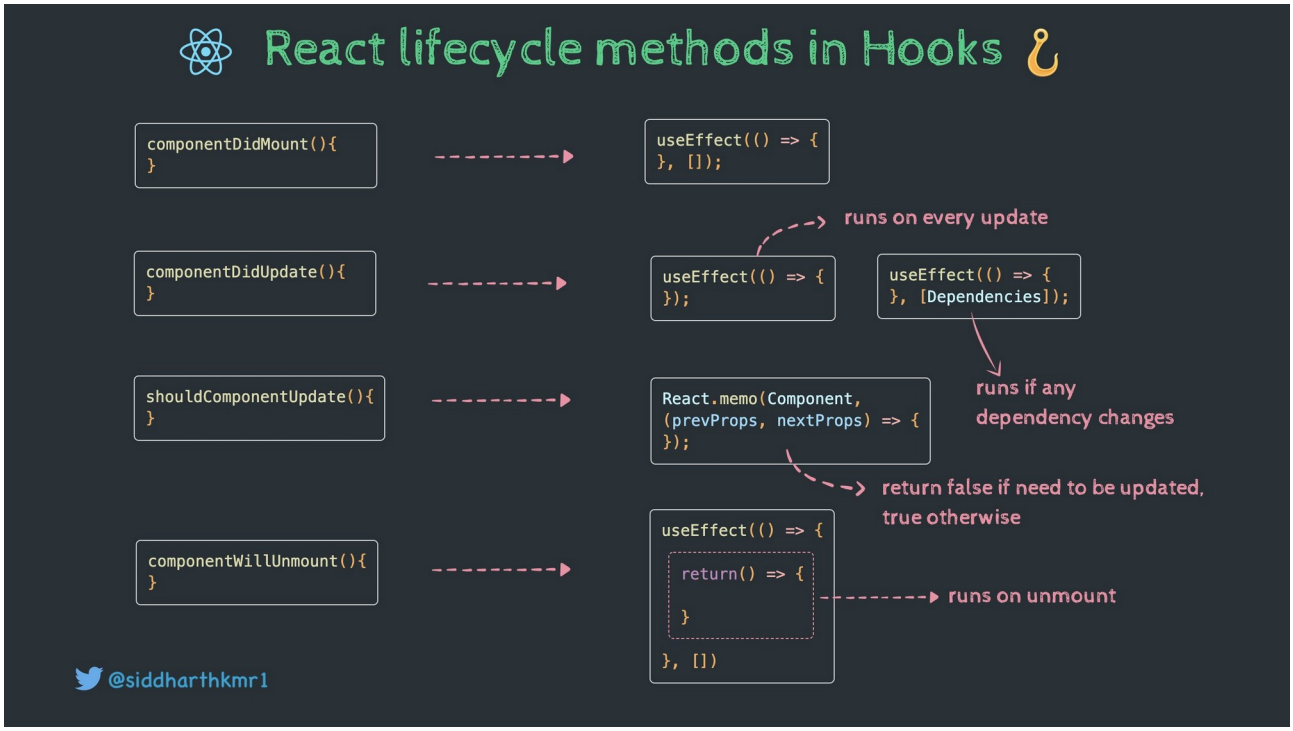

## Exemple :

```jsx
import './App.css';

function App() {

  //On declare une variable = String
  let prenom = "";
  //Un composant simple = fonction + props en paramètre de la fonction = unePropriété
  const ComposantBasePlusProps = (unePropriete) =>{
    return prenom = "Michael"
  }

  //Lors de l'appel du composant dans JSX: on passe la props = un paramètres de la fonctions = ma variable String

  return (
    <div className="App">
      <p>SALUT</p>
      <ComposantBasePlusProps unePropriete={prenom}/>
    </div>
  );
}

export default App;
```

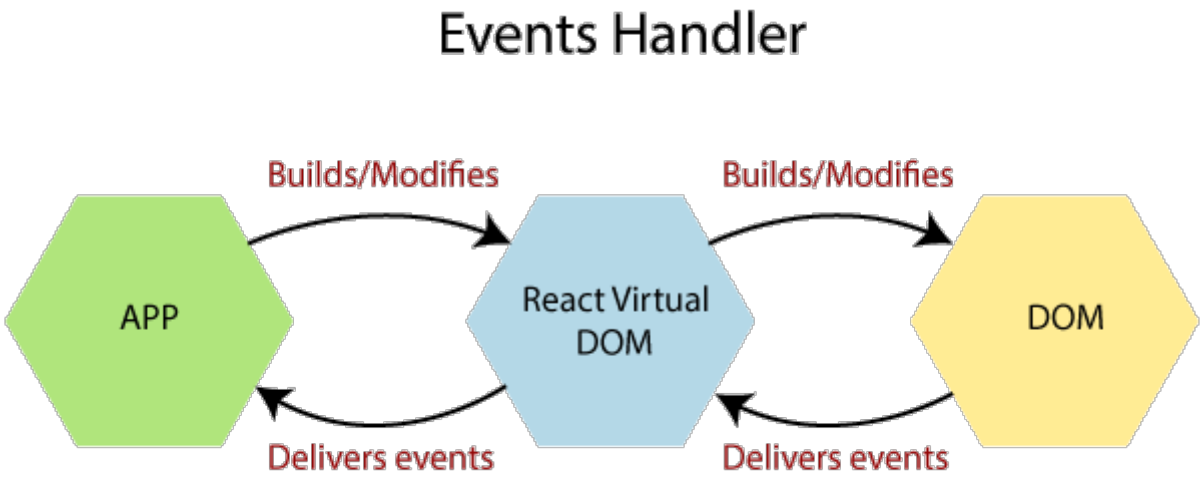
## Le cycle de vie générale de React :



## Le cycle de vie des hooks avec  React :



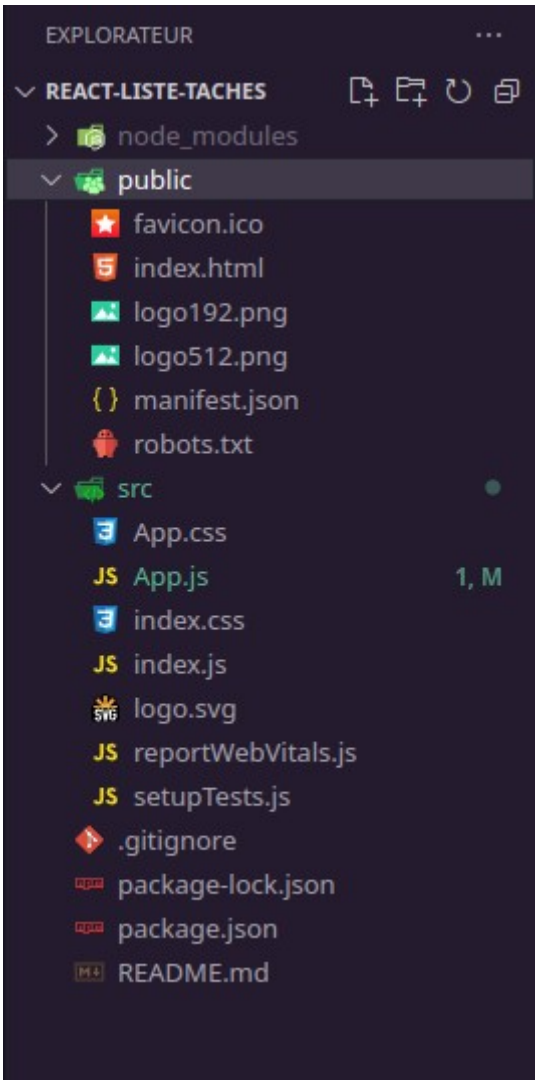

## La gestion des événements :



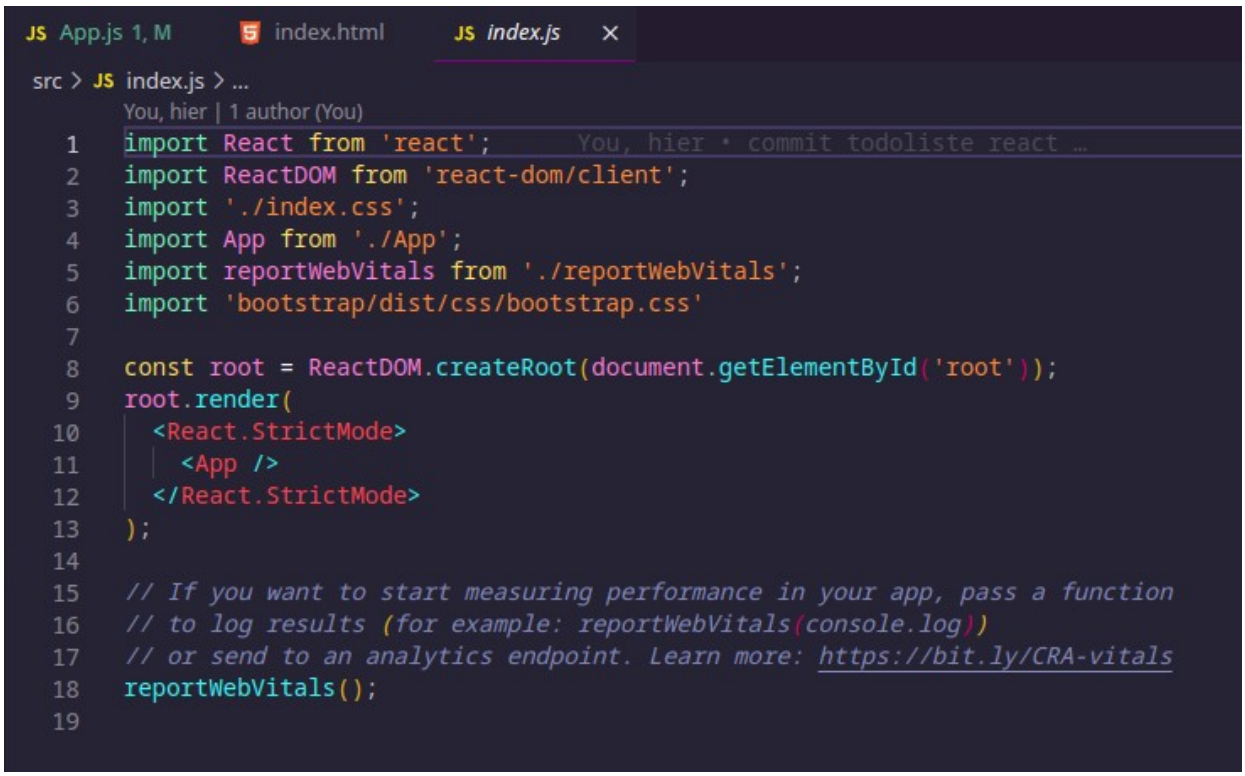

## Les bases de React : Une liste de tache

### Projet 11-A: Une liste de tache simple avec React

Sources : https://github.com/michelonlineformapro/Projet-11-A-Decouverte-React-Hooks

1. Créer votre application React : **npx create-react-app liste-tache**
2. La structure du projet :
3. 

4. Le dossier public/ contient un fichier index.html qui est le template HTML 5 de base de chaque composants
5. React est crocheté sur la 
6. Dans le dossier src/ : le point d'entrée index.js créer l'instance de la librairie react




7. Installer bootstrap 5 a l'aide de votre terminal via la commande :
8. npm i `bootstrap@5.2.0` et `npm i bootstrap-icons`
9. Importer le framework css dans src/index.js via :
10. import 'bootstrap/dist/css/bootstrap.css (ligne 6)
11. (ligne 8) une constante greffe la librairie React a id='app' du fichier public/index.html via ReactDOM.createRoot(document.getElement Id('root')
12. Le composant App.js est importer (ligne 4)
13. Et ce dernier est appelé dans un 1er render ligne 11
14. Ainsi chaque composant créer sont des enfants de App.js
15. Dans le fichier src/App.js
16. Supprimer le contenu JSX (ne garder que la  parente)
17. Lancer votre application via : npm start (rendez-vous sur l'URL localhost:3000)
18. Dans la fonction App() : on commence par créer un tableau d' objet :

```
//Chaque <li> de notre liste est un objet composé (id + tache)
let produitObjet = [
  {
    id:1,
    tache:"Apprendre Javascript",
  },
  {
    id:2,
    tache:"Tester React JS",      You, il y a 11 minutes • Uncommit
  },
]
```

19. On ajoute ensuite un hook d'état pour inspecter l'état du tableau a l'instant T
20. DOCS fr : https://fr.reactjs.org/docs/hooks-state.html

```
//un hook d'etat du tableau de tache
//Ici on peux dire que let produits = produitObjet[]
//et setProduit = (produits) => {fait des trucs}
let [taches, setTaches] = useState(produitObjet)
```

21. Les hooks React utilise l'affectation par décomposition
22. Ajouter une tache

```
//Ajout d'une tache
//function + paramètre = mutateur (setters) du hook (setTache)
//A l'interieur: spreadOpéreator https://geeklecode.com/loperateur-spread-en-javascript-va-vous-simplifier-la-vie/
//cet element realise une copie de tableau taches et separe chaque index et en second paramètre on ajoute la tache entrée dans input
//Ceci est equivalent a taches.push({id: taches.lenght + 1, tache: })
const ajouterTache = (tache) => setTaches([...taches, {tache}]);
```

23. Pour bien comprendre le spread Operator :
24. https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Spread_syntax
25. Et ce code en commentaire :

```
/*

//le spreadOperator (affectation par decomposition)
const tableau = [15,5,10];
function addition(nb1,nb2,nb3){
  return nb1 + nb2 + nb3;
}

let resultat = addition(tableau[0], tableau[1], tableau[2]);
let memeChose = addition(...tableau)

console.log(resultat);
console.log(memeChose)

//Eclater le tableau
console.log(...tableau)

*/
```

26. Le tableau est éclaté par index (rappel : dans tous les langage, l'index des tableaux commencent par 0 )

27. Ajouter la partie HTML via JSX

```
  //ici on realise une boucle sur le tableau de taches avec .map()
  //chaque <li> accède au tableau taches et au objet present
  //On peu donc acceder a chaque element des objets du tableau de taches
  //En bas du code : on appel le composant formulaire ecrit ci-dessus
return (
  <div className="container shadow p-3 mt-5 w-25" id='main-container'>
    <h2 className='text-center text-white'>LES TACHES DU JOUR</h2>

      <ul className='list-group'>
        {taches.map((uneTaches, index) => (
          <li key={index} className='mt-3 list-group-item d-flex justify-content-between'>
              {uneTaches.tache}
              <span className='text-danger icone-poubelle' onClick={supprimerTache}>
                <svg xmlns="http://www.w3.org/2000/svg" width="32" height="32" fill="currentColor" className="bi bi-trash-fill" viewBox="0 0 16 16">
                  <path d="M2.5 1a1 1 0 0 0-1 1v1a1 1 0 0 0 1 1H3v9a2 2 0 0 0 2 2h6a2 2 0 0 0 2-2V4h.5a1 1 0 0 0 1-1V2a1 1 0 0 0-1-1H10a1 1 0 0 0-1-1H7a1 1 0 0 0-1
                </svg>
              </span>
            </li>
      ))}
      </ul>
      <AjouterTacheFormulaire ajouterTacheProps={ajouterTache}/>
  </div>
);
```

28. Nous allons désormais créer le Composant <AjouterTacheFormulaire + sa props/>

29. Créer une constante qui retourne une fonction anonyme qui prend un objet en paramètre (aussi appelé argument d'une fonction)

30. Ce paramètre s' appel ajouterTacheProps et sera passer en attribut lors de l'appel du composant dans le JSX

```
//Le formulaire = function + props passee en paramètre lors de l'appel de ce dernier
//ex: <AjouterTacheFormulaire ajouterTacheProps={quelque chose}/>
//On passe des propriétés de ce composant a la liste (ici la valeur du champ <input> et son etat)
const AjouterTacheFormulaire = ({ajouterTacheProps}) => {
```

31. Le paramètres = props  est un objet

32. Créer un hook (crochet) qui au ra pour rôle de récupérer les valeur de <input>

```
    //un hook d'etat des valeurs du champ <input>
    //inputValue = '' String
    //setInputvalue = () => {} = mutation (setter) de inputValue[]
    let [inputValue, setInputValue] = useState('')
```

33. Rappel : inputValue est une variable string

34. setInputValue est une fonction (un mutateur (setters) capable de modifier l 'état et la valeur d'un élément)

35. Créer une fonction appelée a la soumission du formulaire via l'attribut onClick

```
  //Fonction qui inspecte le changement d'etat du champ <input>
 const soumissionFormulaire = (e) =>{
   //Supprimer le comportement par defaut de la page web = ici evite le rechargement de la page syncrone
   e.preventDefault();
   //La string du hook + le paramètre de la fonction a appelé <AjouterTacheFormulaire ajouterTacheProps={}/>
   inputValue && ajouterTacheProps(inputValue)
   //On modifie l'etat du champ input = reinitialisation = vide le champ
   setInputValue('')
   //Debug console f12
   console.log(setInputValue)
 }
```

36. On passe a la props (ajouterTacheProps) la valeur de l'input

37. Puis on vie de champs via le mutateur (setters)

38. On ajoute le formulaire au JSX de cette fonctionne

```jsx
//JSX HTML
//A la soumission du formulaire on appel la fonction onSubmit={soumissionFormulaire}
//Le champ input value prend la valeur init du hook
//Quand utilisateur entre des lettres => onChange detecte un changement d'etat et recupère la valeur de ce dernier
//equivalent a : document.getElementsByTagName('input').value
//Ce composant est appeleé dans le parent <ListeTache/>
return(
  <div className="mt-3" id='main-container'>
    <h2 className='text-center text-white'>AJOUTER UNE TACHE</h2>
    <form onSubmit={soumissionFormulaire}>
      <input
        type="text"
        placeholder='Votre taches'
        value={inputValue}
        onChange={e => setInputValue(e.target.value)}
        className='form-control'/>

        <span>
          <button type='submit' className='btn btn-warning mt-3 text-success btn-ajouter'>
            <span>
              <svg xmlns="http://www.w3.org/2000/svg" width="32" height="32" fill="currentColor" className="bi bi-plus-circle-fill" viewBox="0 0 16 16">
                <path d="M16 8A8 8 0 1 1 0 8a8 8 0 0 1 16 0zM8.5 4.5a.5.5 0 0 0-1 0v3h-3a.5.5 0 0 0 0 1h3v3a.5.5 0 0 0 1 0v-3h3a.5.5 0 0 0 0-1h-3v-3z"/>
              </svg>
            </span>
          </button>
        </span>
    </form>
  </div>
)      You, hier • commit todoliste react …
}
```

39. La balise <from> prend un attribut onSubmit=’’ qui appel la fonction créer ci-dessus
40. Le champ <input> prend un paramètre value={inputValue} = la variable du hook
41. Et un attribut onChange=’’ qui appel le mutateur du hook
42. Ce dernier récupère la valeur du champ a l’aide de e.target.value
43. Appeler le composant :
44. <AjouterTacheFormulaire ajouterTacheProps={ajouterTache}/>
45. Dans le JSX de la fonction App()
46. Faite des tests
47. Supprimer une Tache
48. Dans chaque <li> possède une icône bootstrap
49. Au clic sur cette icône on appel une fonction suprimerTache

```jsx
<span className='text-danger icone-poubelle' onClick={supprimerTache}>
  <svg xmlns="http://www.w3.org/2000/svg" width="32" height="32" fill="currentColor" className="bi bi-trash-fill" viewBox="0 0 16 16">
    <path d="M2.5 1a1 1 0 0 0-1 1v1a1 1 0 0 0 1 1H3v9a2 2 0 0 0 2 2h6a2 2 0 0 0 2-2V4h.5a1 1 0 0 0 1-1V2a1 1 0 0 0-1-1H10a1 1 0 0 0-1-1H7a1 1 0 0 0-
  </svg>
</span>
```

50. La fonction supprimerTache

```jsx
//Supprimer une tache a l'aide de son index
const supprimerTache = (index) => {
  //On eclate le tableau de taches
  const unetache = [...taches];
  //.splice() recupère l'index concerné et en second paramètre le nombre d'index a supprimer
  unetache.splice(index, 1);
  //On met a jour l'etat du  tableau avec le mutateur du hook
  setTaches(unetache)
  //Cette fonction est appelé au clic sur l'icon poubelle de chaque <li>
}
```

51. Le rendu finale


52. Bravo vous connaissez les base de React + Hooks d'état

## Projet 11-B : Un Chat avec React – ExpressJs et Socket.io

Sources : https://github.com/michelonlineformapro/Projet-11-B-Chat-React-Socket.io

le tutoriel : https://www.cril.univ-artois.fr/~boussemart/react/chapter01.html

## Projet 11-C : Un CRUD avec React – Json-server

1. Créer une nouvelle application React : dans un terminal
2. `npx create-react-app react-crud-hooks`
3. Entrer dans votre dossier : cd `react-crud-hooks`
4. `Lancer l'application : npm start`

5. Rendez-vous sur l'url : http://localhost:3000/
6. Supprimer le contenu de la fonction App()
7. Ajouter un exemple de jsx

```jsx
import './App.css';

function App() {

  //Exemple de base de JSX

  return (
    <div className="App">
      <p>SALUT React</p>

    </div>
  );
}

export default App;
```

8. Un exemple de base de jsx

9. Installer bootstrap 5 a l'aide de votre terminal via la commande :
10. npm i bootstrap@5.2.0 et npm i bootstrap-icons
11. Importer le framework css dans src/index.js via :
12. import 'bootstrap/dist/css/bootstrap.css
13. POUR LE BACKEND
14. On utilise le middleware json-server
15. https://www.npmjs.com/package/json-server
16. dans votre terminal : npm i json-server
17. Générer votre fichier db.json
18. npx json-server –watch db.json –port 3001
19. Ici on ajoute le drapeau –port pour changer le port du serveur
20. En effet le port:3000 est déjà pris par react
21. Back : port :3001 = json-server
22. Front : 3000 = React
23. Modifier le fichier db.json (générer a la racine du projet)
24. Exemple :

App.js U  db.json U  exemple.json U

exemple.json > [ ] livres > {} 1

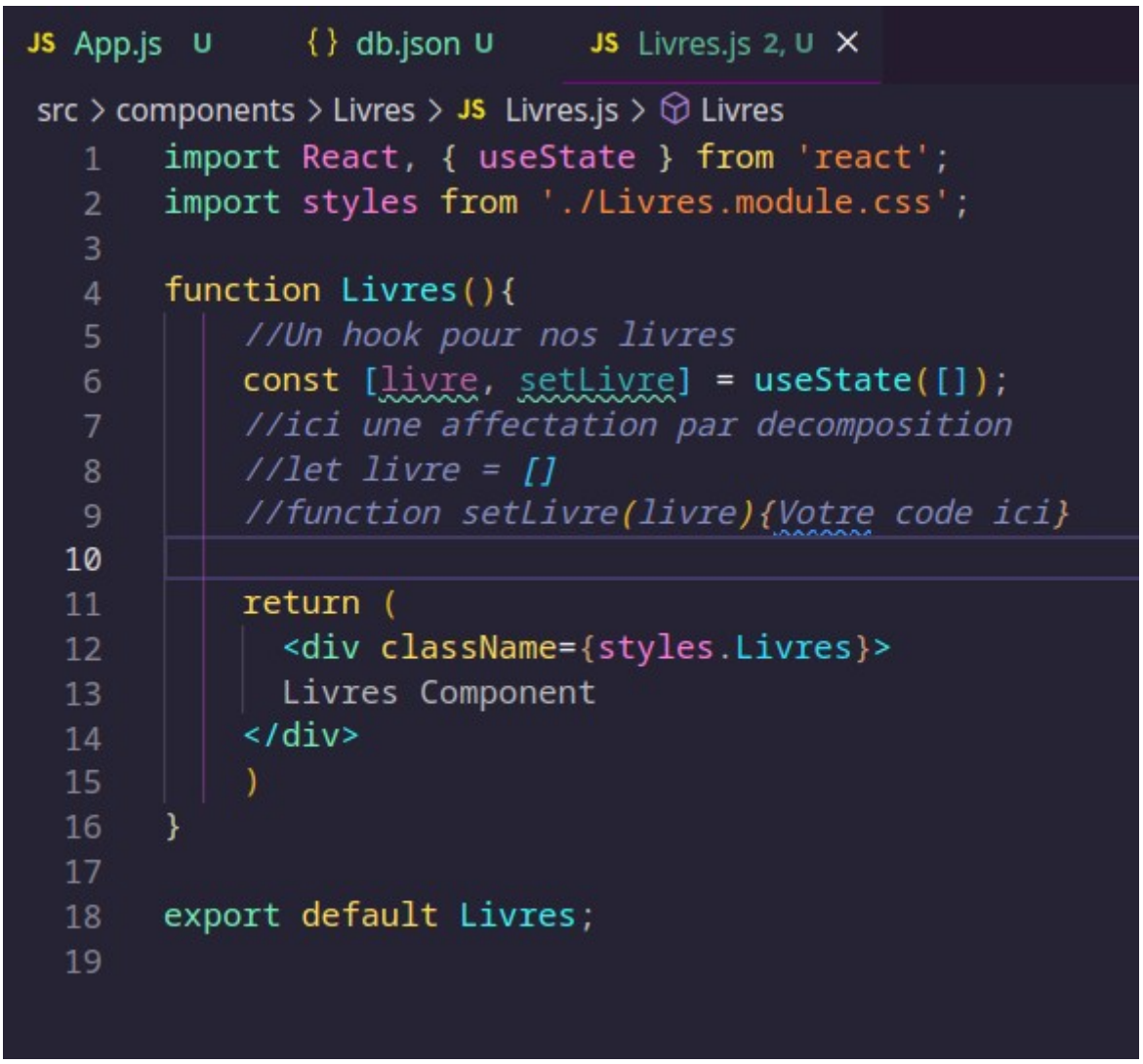
```
{
    "livres": [
      {
        "id": 1,
        "nomLivre": "Livre javascript",
        "descriptionLivre": "JavaScript est un langage de programmation de scripts principalement emplo
        "prixLivre": 12.5,
        "imageLivre": "https://static.fnac-static.com/multimedia/Images/FR/NR/0d/4a/99/10045965/1507-1/t
        "categoriesLivre": "programmation"
      },
      {
        "id": 2,
        "nomLivre": "Livre PHP",
        "descriptionLivre": "PHP: Hypertext Preprocessor, plus connu sous son sigle PHP, est un langage
        "prixLivre": 45458.25,
        "imageLivre": "https://m.media-amazon.com/images/I/319uThTlOSL.jpg",
        "categoriesLivre": "enfant"
      }
    ]
}
```

25. Rendez-vous sur URL : http://localhost:3001/livres
26. Votre backend est terminé
27. POUR LE FRONTEND
28. Générer des composants :
29. https://www.npmjs.com/package/generate-react-cli
30. `npx generate-react-cli component Livres`
31. Répondez aux questions

```
mic-kubuntu@mic-kubuntu ~/Documents/Web_dev/react-crud-produits  master ± npx generate-react-cli component Livres
Need to install the following packages:
  generate-react-cli@7.3.0
Ok to proceed? (y)

----------------------------------------------------------------------------------------------------------------------------------------------------------------
It looks like this is the first time that you're running generate-react-cli within this project.

Answer a few questions to customize generate-react-cli for your project needs (this will create a "generate-react-cli.json" config file on the root level of this project).
----------------------------------------------------------------------------------------------------------------------------------------------------------------

? Does this project use TypeScript? No
? Does this project use CSS modules? Yes
? Does this project use a CSS Preprocessor? css
? What testing library does your project use? None
? Set the default path directory to where your components will be generated in? src/components
? Would you like to create a corresponding stylesheet file with each component you generate? Yes
? Would you like to create a corresponding test file with each component you generate? No
? Would you like to create a corresponding story with each component you generate? No
? Would you like to create a corresponding lazy file (a file that lazy-loads your component out of the box and enables code splitting:
https://reactjs.org/docs/code-splitting.html#code-splitting) with each component you generate? No

The "generate-react-cli.json" config file has been successfully created on the root level of your project.

You can always go back and update it as needed.

Happy Hacking!


Livres.js was successfully created at src/components/Livres/Livres.js
Livres.module.css was successfully created at src/components/Livres/Livres.module.css
mic-kubuntu@mic-kubuntu ~/Documents/Web_dev/react-crud-produits  master ±
```

32. Cette commande à créer un dossier src/components/
33. A l'intérieur : un fichier Livres.js + Livres.module.css (chaque composant a son propre fichier css)
34. Modifier le contenus de App.js (composant parent)


35. Ainsi le composant parent App.js appel le composant Livres.js
36. Dans votre composant Livres.js et la fonction Livres()
37. Créer un hook livres :



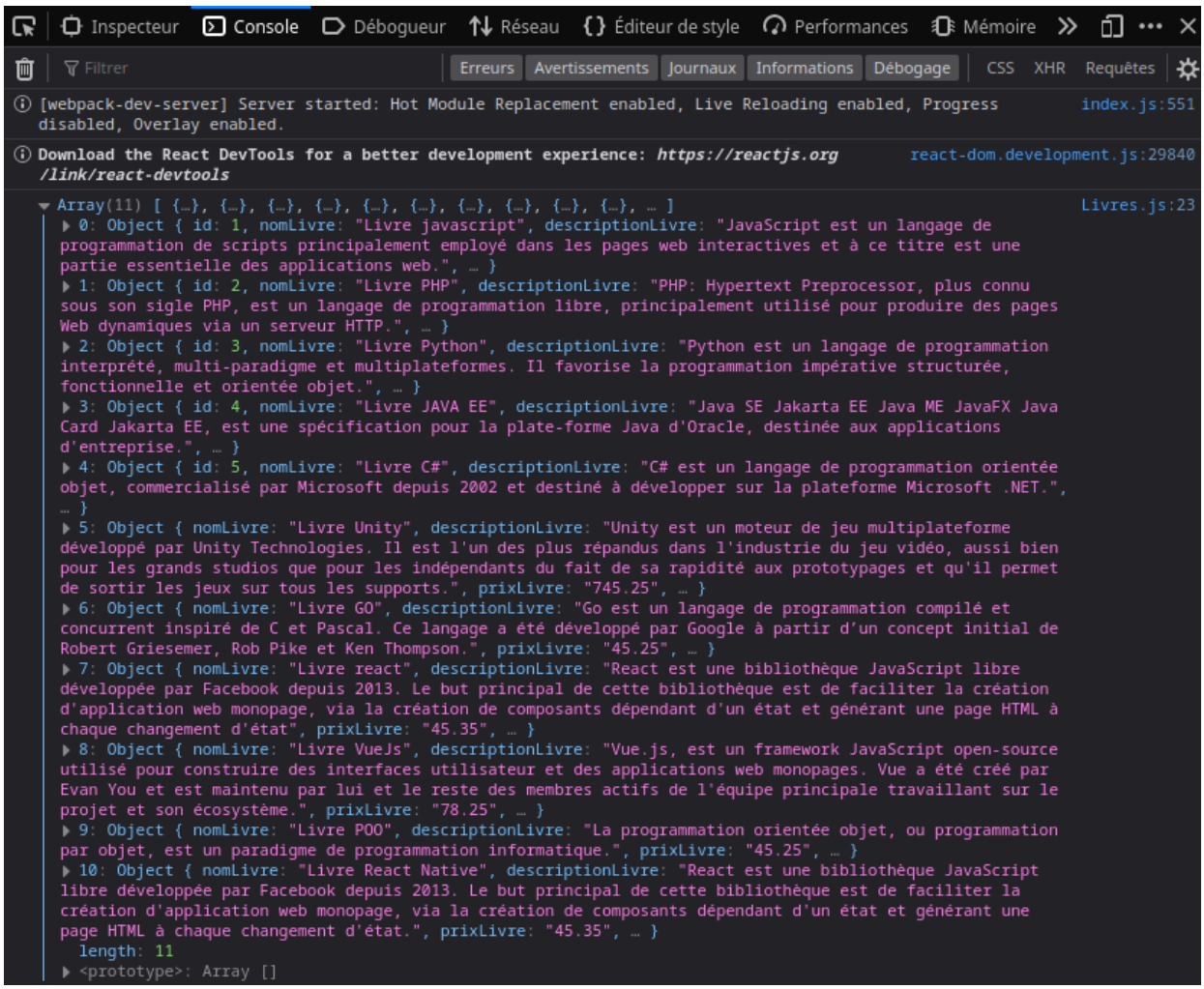

38. Pour effectuer notre 1er requête : on utilise le middleware axios
39. https://www.npmjs.com/package/axios
40. npm i axios
41. importer axios a l'aide de webpack
42. import axios from 'axios'
43. Créer une fonction afficherLivres()
44. Écrire la requête HTTP à l'aide de axios

```
//la fonction pour afficher les livres
const afficherLivres = () => {
  //La requète HTTP avec axios + methode GET
  axios.get('http://localhost:3001/livres')
  //la promesse = reponse de json-server
  .then(reponse => {
    //On utilise le mutateur du hook (setter) et on passe l'objet reponse en paramètre
    setLivre(reponse.data);
    console.log(reponse.data);
  })
  //Si la promesse n'est pas tenue = on affiche une erreur
  .catch(erreur => {
    console.error('Erreur de requète HTTP ' + erreur)
  })
}
```

45. Pour appeler cette fonction : on utilise le hook d'effet similaire a componentDidMount et componentDidUpdate dans une classe
46. C'est a dire quand le composant Livre est monté et a jour
47. Après donc l'appel du 1er render()
48. https://fr.reactjs.org/docs/hooks-effect.html

```
useEffect(() => {
  afficherLivres()
}, [])
```

49. On passe un tableau en second paramètre pour éviter des appel infinis de la fonction
50. Afficher le debug du navigateur f12 + onglet console




51. Coté JSX (afficher le tableau d'objet)

```jsx
return (
  <div className={styles.Livres}>
    <div className='row'>
      {livre.map((book, index) =>
          <div className='col-md-4 col-sm-12 mt-5' key={index}>
            <div id='carte' className='card'>
            <img className='p-3' src={book.imageLivre} alt={book.nomLivre} title={book.nomLivre}/>
              <div className='card-body'>
                <h5 className='text-danger'>{book.nomLivre}</h5>
                <p className='card-text'>{book.descriptionLivre}</p>
                <p>PRIX: {book.prixLivre}</p>
                <p>{book.categoriesLivre}</p>
              </div>
            </div>
          </div>
      )}
    </div>
  </div>
)
```

52. Rendu :


53. Afficher les détails d'un livre
54. Créer 2 nouveaux hook

55. le premier pour récuperer le livre concerné et le second on pour récupere l'index de l'objet dans le tableau json

```
//Recuperer un seul livre
const [livreConcerner, setLivreConcerner] = useState(null);
//L'index d'un objet dans le tableau json
const [livreIndex, setLivreIndex] = useState(-1);
```

56. Créer une fonction qui affiche un seul livre à l'aide des hook

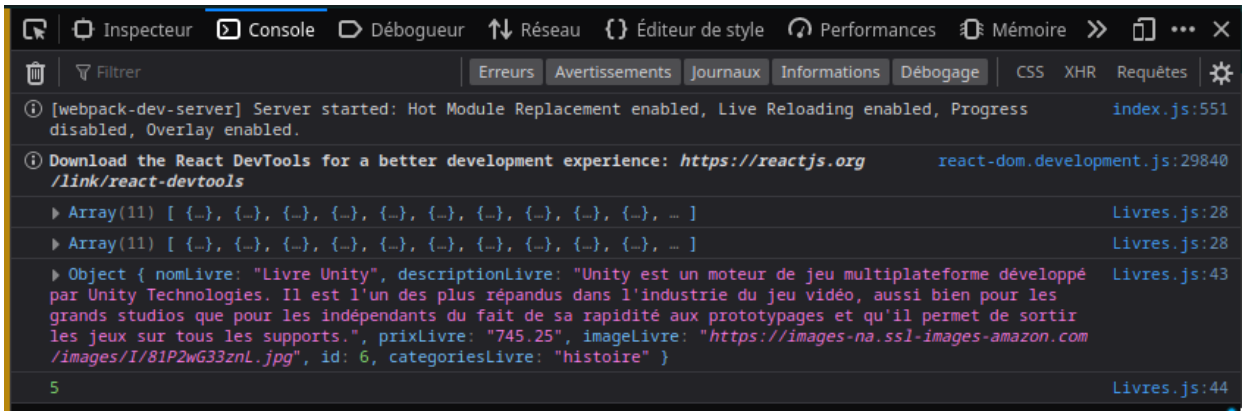
```
//afficher un seul livre : on passe en paramètre les livres du hook + index
const livreParId = (livre, index) => {
  //On utilise le mutateur du hook livre concercerner et on assigne les livres du hook livre
  setLivreConcerner(livre);
  //idem pour l'index
  setLivreIndex(index);
  //Debug
  console.log(livre);
  console.log(index);
}
```

57. Enfin ajouter un bouton dans le JSX qui appel la fonction livreParId() a l'aide de l'attribut onClick(paramètre, paramètre)

```
<p>{book.categoriesLivre}</p>
<button className='btn btn-warning' onClick={() => livreParId(book, index)}>Plus d'infos</button>
```

58. Au clic sur un bouton d'un des livres, inspecter le résultat dans votre console de debug du navigateur

59. Exemple : au clic dur le bouton du livre id = 6




60. On obtient bien les données de l'objet du tableau json id = 6 et l'index du tableau est bien : 5

61. Afficher un seul livre au clic

62. On utilise l'affichage a l'aide des structure conditionnel

63. https://fr.reactjs.org/docs/conditional-rendering.html

64. On peu donc scinder l'affichage a l'aide d'une condition booléen

65. Si livre concerner retourne une valeur : on affiche un bloc JSX

66. Sinon on affiche tous les livres par défaut

67. Le code pour un exemple de base

```
return (
  <div className={styles.Livres}>

    {/* si livre concerner retourne une valeur */}
    {livreConcerner ?(
      <div className='container w-50 shadow'>
          {/* ICI LE CODE POUR AFFICHER UN SEUL LIVRE */}
      </div>
    ):(
      <div className='row'>
          {/* Sinon on afficher tous les livres */}
      {livre.map((book, index) =>
          <div className='col-md-4 col-sm-12 mt-5' key={index}>
            <div id='carte' className='card'>
            <img className='p-3' src={book.imageLivre} alt={book.nomLivre} title={book.nomLivre}/>
              <div className='card-body'>
                <h3 className='text-danger'>{book.nomLivre}</h3>
                <p className='card-text'>{book.descriptionLivre}</p>
                <p>PRIX: {book.prixLivre}</p>
                <p>{book.categoriesLivre}</p>
                <button className='btn btn-warning' onClick={() => livreParId(book, index)}>Plus d'infos</button>
              </div>
            </div>
          </div>
      )}
    </div>
    )}
  </div>
)
```

68. Un fois les détails d'un livre afficher on ajoute les bouton ÉDITER et SUPPRIMER

69. Le code complet :

```
    {/* si livre concerner retourne une valeur */}
    {livreConcerner ?(
      <div className='container w-50 shadow'>
            <div id='carte' className='card'>
            <img className='p-3' src={livreConcerner.imageLivre} alt={livreConcerner.nomLivre} title={livreConcerner.nomLivre}/>
              <div className='card-body'>
                <h3 className='text-danger'>{livreConcerner.nomLivre}</h3>
                <p className='card-text'>{livreConcerner.descriptionLivre}</p>
                <p>PRIX: {livreConcerner.prixLivre}</p>
                <p>{livreConcerner.categoriesLivre}</p>
                <button className='btn btn-info'>EDITER</button>
                <button className='btn btn-danger mx-3'>SUPPRIMER</button>
                <button className='btn btn-success' onClick={() => window.location.reload()}>RETOUR</button>
              </div>
            </div>
      </div>
    ):(
      <div className='row'>
          {/* Sinon on afficher tous les livres */}
```

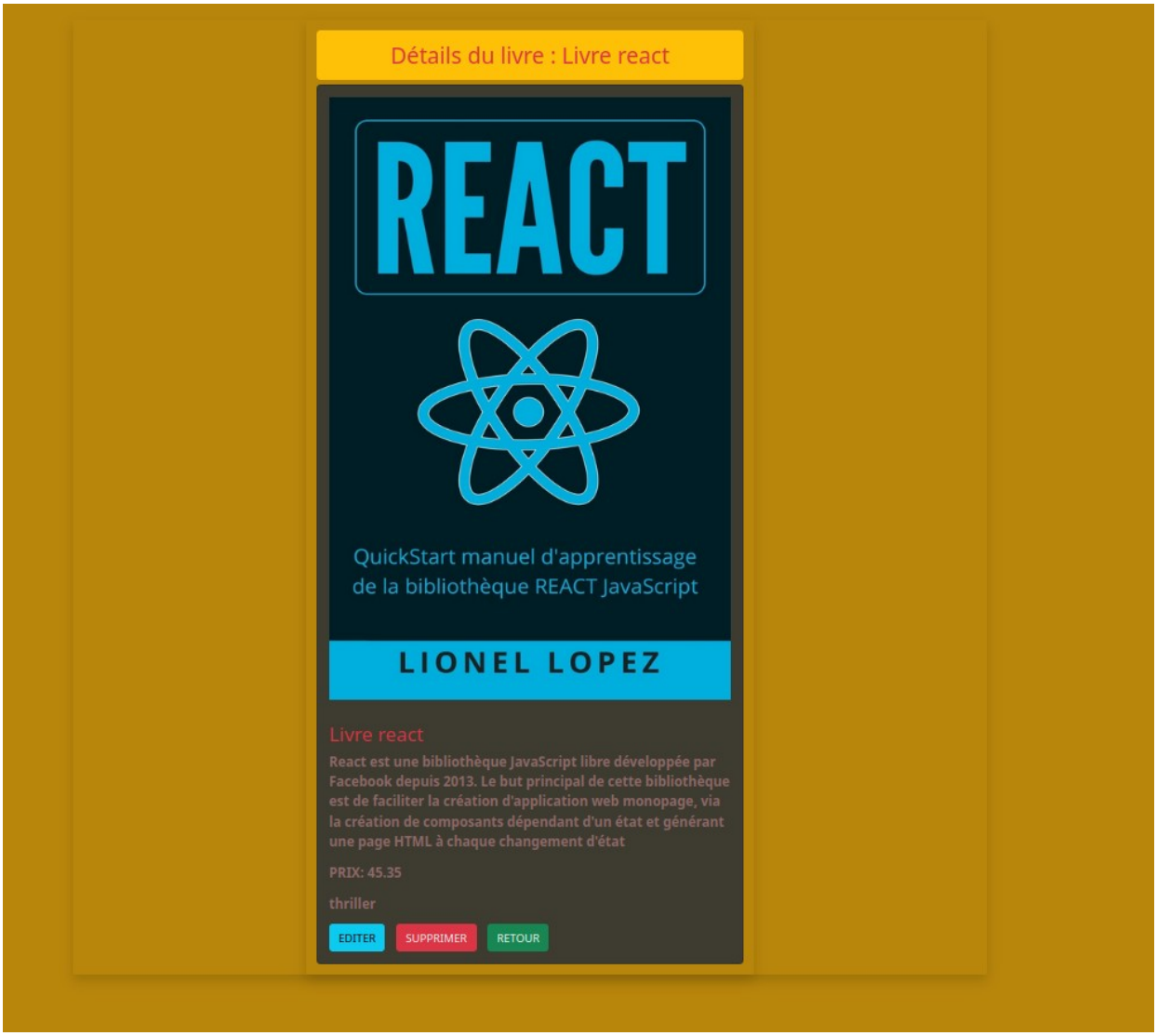
70. Pour l'instant les bouton éditer et supprimer ne servent a rien

71. Le bouton retour rafraîchit la page a l'aide de la fonction du BOM javascript

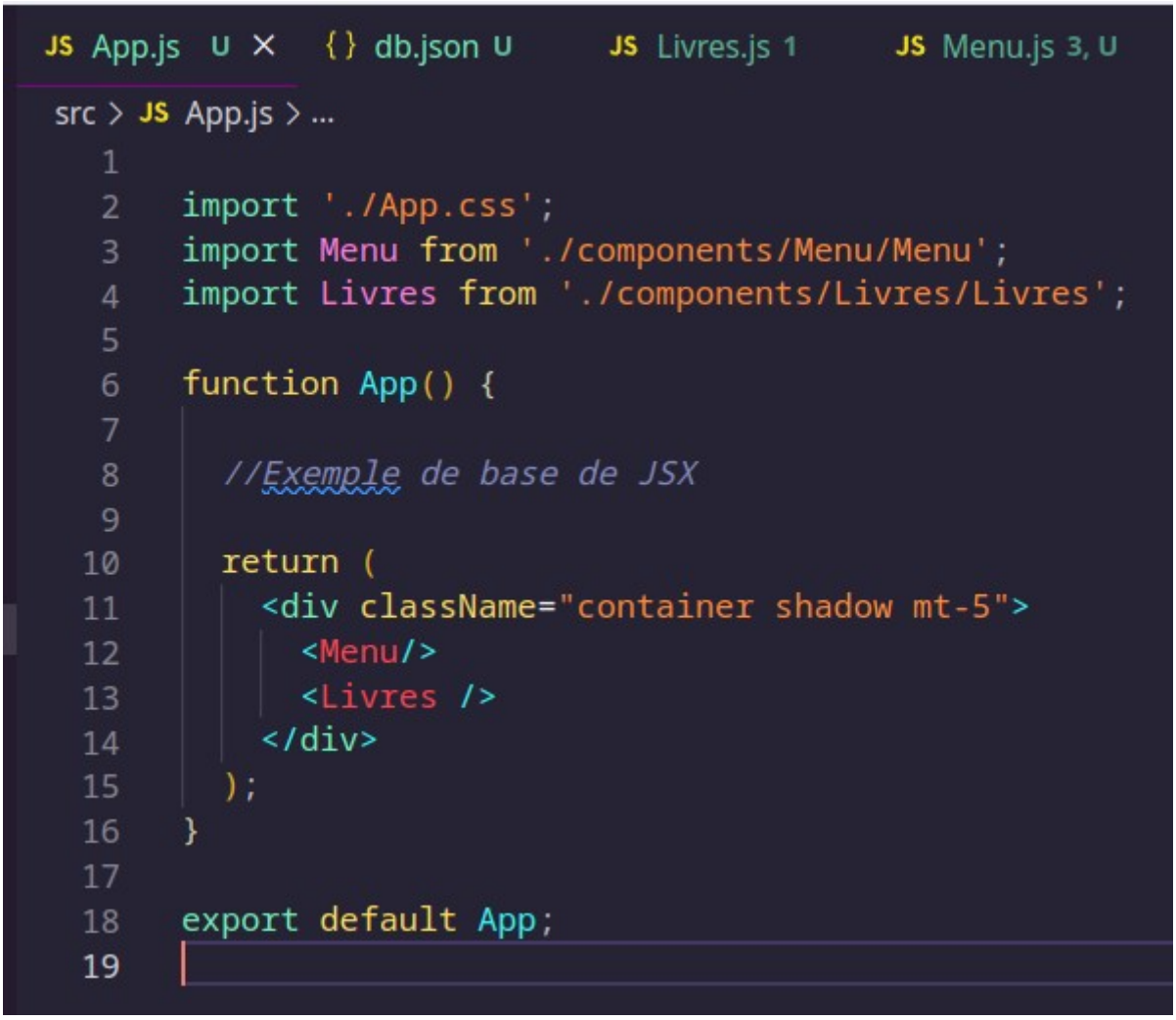
72. window.location.reload()

73. Ceci a pour effet de simuler un retour a la page d'accueil

74. Résultat :




75. Ajouter un livre
76. POUR CETTE ÉTAPE : IL FAUT CRÉER UN ROUTER
77. Un composant <Menu/> sera présent sur chaque page
78. React – Router – DOM : https://reactrouter.com/en/main/getting-started/installation#basic-installation
79. Installer le package : npm install react-router-dom@6
80. Générer un nouveau composant :
81. npx generate-react-cli component Menu
82. Importer et afficher le composant dans le composant parent App.js




83. Avant d'écrire le code de la navbar : générer un nouveau composants

84. npx generate-react-cli component AjouterLivre
85. Dans le fichier Menu.js : en haut de la page importer les elements suivants :

86. import { Routes, Route, Link } from "react-router-dom";
87. Dans le JSX : ajouter le code de la barre de navigation horizontale

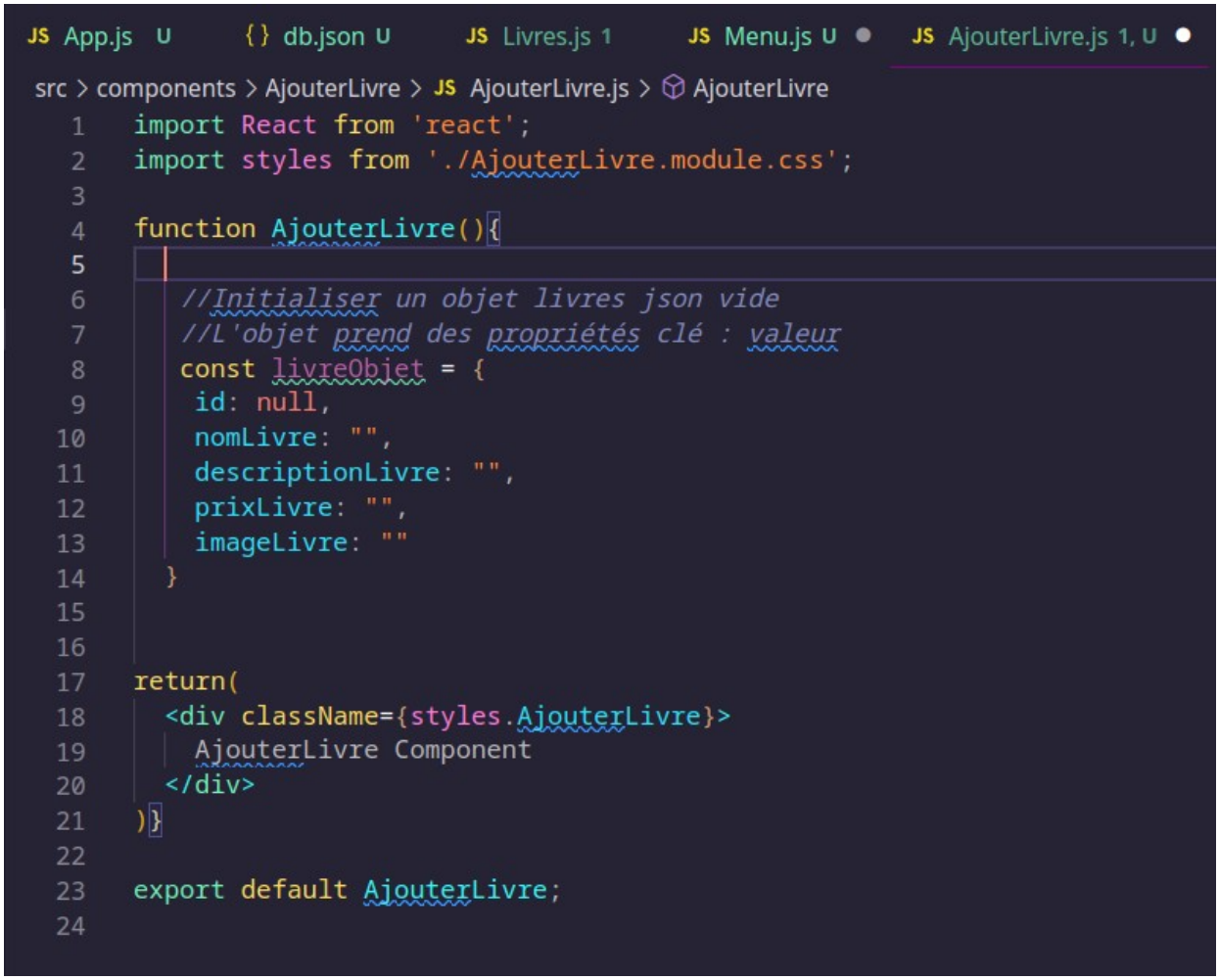
88. Le code du composant <Menu/>

```
JS App.js U   { } db.json U   JS Livres.js 1   JS Menu.js U ●   Livres.module.css 1   App.css U
src > components > Menu > JS Menu.js > Menu
1   import React from 'react';
2   import styles from './Menu.module.css';
3
4   //Import des module de react-router-dom
5   import { Routes, Route, Link, BrowserRouter } from "react-router-dom";
6
7   //Import des composants <Livres/> et <AjouterLivre/>
8   import Livres from '../Livres/Livres';
9   import AjouterLivre from '../AjouterLivre/AjouterLivre';
10
11  //Fonction a exporter et importer dans App.js
12  const Menu = () => (
13    <div className={styles.Menu}>
14      {/* Les liens = <a href=''>Liens</a> */}
15      <BrowserRouter>
16      <nav class="navbar navbar-light bg-warning rounded shadow mt-3 p-3">
17          <div class="container-fluid">
18              <Link to='/'>NOS LIVRES</Link>
19              <Link to='/ajouter-livre'>AJOUTER UN LIVRE</Link>
20          </div>
21        </nav>
22        {/* chaque route to appel un chemin (path) => appel un composant importer */}
23        <Routes>
24          <Route path='/' element={<Livres/>}/>
25          <Route path='/ajouter-livre' element={<AjouterLivre/>}/>
26        </Routes>
27      </BrowserRouter>
28    </div>
29  );
30
31  export default Menu;
32
```

89. Résultat :


90. Au clic sur le lien AJOUTER LIVRE : on affiche le contenu du composant <AjouterLivre />

91. Le composant <AjouterLivre />
92. L'objectif est de créer un formulaire et de parser les données de chaque champ au fichier db.json
93. Transformer votre constante AjouterLivre en fonction
94. On initialise un objet vide :




95. Créer 2 hooks :
96. Le premier pour créer un livre et le second est un booléen pour repérer l'état du formulaire (soumis ou non)

```
//2 hooks
//1er pour creer un nouvel objet livre

//le 2nd pour l'etat du formulaire (booleen soumis ou non)

//ici let livre = livreObjet

//et function setLivre(livre) {votre code}

const [livre, setLivre] = useState(livreObjet);

//Etat du formulaire

//let soumis = false;

//et function setSoumis(soumis){votre code}

const [soumis, setSoumis] = useState(false);
```

97. On créer ensuite une fonction qui détecte les changement d'état du DOM virtuel
98. A chaque entrée dans un champ du formulaire le DOM virtuel détecte un changement d'état
99. La fonction handleInputChange()

```
//fonction de tracking pour reperer les changements dans les inputs et les changements d'etats du DOM
const handleInputChange = event => {
  //Recupération des attributs input name et value
  //Ceci est un affectation par decomposition soit :
  //const name = event.target
  //const value = event.target
  const {name, value} = event.target;

  //spread Operator (extrait des données d'un tableau ou un objet et creer un nouveau tableau)
  //Appel du mutateur setLivre
  //On decompose (eclate) l'objet livre puis l'attribut name du champ <input> est egale a l'attribut value = event.target
  setLivre({...livre, [name]: value})
  console.log(setLivre)
}
```

100. La fonction la plus importante :
101. La liaison des champ du formulaire avec le fichier db.json du back et la sauvegarde des données
102. Le code de la fonction sauverLivre()

```
//Lier les données du fromulaire avec les objets du tableau json + sauvegarde
const sauverLivre = () => {
  //On creer un objet vide rempli avec les champ du formulaire
  let livreFormData = {
        nomLivre: livre.nomLivre,
        descriptionLivre: livre.descriptionLivre,
        prixLivre: livre.prixLivre,
        imageLivre: livre.imageLivre
  }

  //La requete HTTP (importer axios via webpack)
  //fetch = axios + methode + url + requete + reponse + objet livreFormData
  axios.post('http://localhost:3001/livres', livreFormData)
  //La promesse = reponse du serveur (json-server) a la requete fetch (soit axios)
  .then(reponse => {
    //A l'aide du hook : on lie les données du formulaire a notre objet vide livreFormData
    setLivre({
        id: reponse.data.id,
        nomLivre: reponse.data.nomLivre,
        descriptionLivre: reponse.data.descriptionLivre,
        prixLivre: reponse.data.prixLivre,
        imageLivre: reponse.data.imageLivre
    });
    //On change le status du booleen pour afficher 2 condition bloc dans le JSX
    //{soumis ?(bloc jsx 1 ):(bloc jsx 2)}
    setSoumis(true);
    //Debug
    console.log(reponse.data);
  })
  //Si la promesse n'est pas tenue : on affiche une erreur
  .catch(erreur => {
    console.error("Erreur lors de l'ajout du livre ! " + erreur);
  });
}
```

103. Le JSX 2 conditions

104. Si le formulaire est valide et soumis

105. Dans l'objet return dans une 

```
{/* SI LE FORMULAIRE EST SOUMIS soumis = true */}
{soumis ?(
  <div className='alert alert-success'>
      <h4 className='text-center text-white'>Le livre a bien été ajouté !</h4>
      <button className='btn btn-success mt-3'>Ajouter un autre livre ?</button>
      <button className='btn btn-warning mt-3' onClick={() => window.location.reload()}>Fermer</button>
  </div>
):(

  <div>
      {/* SINON AFFICHE LE FORMUAIRE  soumis = false*/}      You, il y a 1 seconde • Uncommitted changes
```

106. Sinon on affiche le formulaire

```
{/* SINON AFFICHE LE FORMUAIRE  soumis = false*/}      You, il y a 1 seconde • Uncommitted changes
            <div className="mt-3">
                <label className="label">Nom du livre</label>
                <input
                    type="text"
                    className="form-control"
                    id="nomLivre"
                    name="nomLivre"
                    placeholder="Nom du livre"
                    required
                    value={livre.nomLivre}
                    onChange={handleInputChange}
                />
            </div>

            <div className="mt-3">
                <label className="label">Description du livre</label>
                <textarea
                    className="form-control"
                    id="descriptionLivre"
                    name="descriptionLivre"
                    required
                    rows="5"
                    placeholder="Description du livre"
                    value={livre.descriptionLivre}
                    onChange={handleInputChange}
                />
            </div>

            <div className="mt-3">
                <label className="label">Prix du livre</label>
                <input
                    type="number"
                    step='0.01'
                    className="form-control"
                    id="prixLivre"
                    placeholder="Prix du livre"
                    name="prixLivre"
                    required
                    value={livre.prixLivre}
                    onChange={handleInputChange}
                />
            </div>

            <div className="mt-3">
                <label className="label">Image du livre</label>
                <input
                    type="text"
                    className="form-control"
                    id="imageLivre"
                    name="imageLivre"
                    placeholder="URL de l'image du livre"
                    required
                    value={livre.imageLivre}
                    onChange={handleInputChange}
                />
            </div>

            <button className="btn btn-success mt-3" onClick={sauverLivre}>Ajouter le livre</button>
```

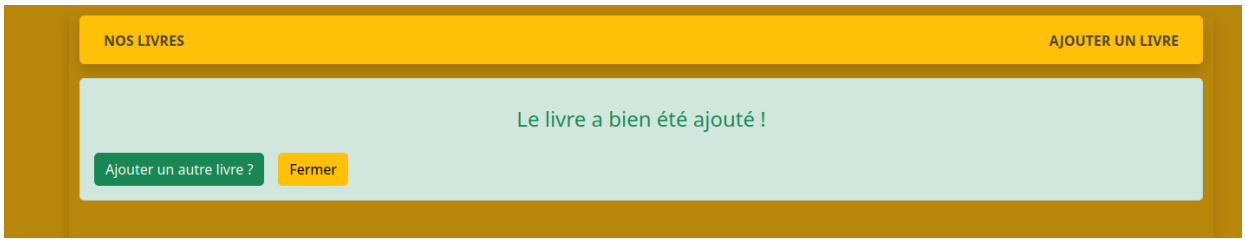
107. Ici chaque value des champs du formulaire sont égales a une propriété de l'objet livreObjet

108. A chaque valeur entrée dans le champs, react détecte un changement d'état à l'aide de l'attribut onChange qui appel la fonction handleInputChange()

109. Le bouton appel la fonction sauverLivre() a l'aide  l'attribut onClick={sauverLivre}

110. Résultat :


111. Si ça marche




112. Il reste une dernière fonction ajouter qui va vide le formulaire et permettre d'ajouter un autre livre

113. La fonction

```
//Vider et afficher le formulaire
const nouveauLivre = () => {
  //On reinitialise le hook avec l'objet vide
    setLivre(livreObjet)
    //L'etat soumis du formulaire est de nouveau a false
    setSoumis(false);       You, il y a 1 seconde • Uncommitted changes
}
```

114. Appel de la fonction depuis un bouton quand le formulaire est soumission

```
{/* SI LE FORMULAIRE EST SOUMIS soumis = true */}
{soumis ?(
  <div className='alert alert-success mt-3'>
      <h4 className='text-center text-success mt-3'>Le livre a bien été ajouté !</h4>
      <button className='btn btn-success mt-3' onClick={nouveauLivre}>Ajouter un autre livre ?</button>      You,
      <button className='btn btn-warning mt-3 mx-3' onClick={() => window.location.reload()}>Fermer</button>
  </div>
):(

  <div>
    {/* SINON AFFICHE LE FORMUAIRE  soumis = false*/}
```

115. SUPPRIMER UN PRODUIT

116. Retour sur le composant Livres.js

117. Lors du clic sur le bouton détails (plus d'infos) : on affiche un seul livre

118. Le bouton supprimer va appeler une fonction a l'aide de l'attribut onClick()

119. La fonction

```
//Supprimer un livre
const supprimerLivre = () => {
  //La requete fetch http (axios) + methodes DELETE + url + id dynamique +
  axios.delete(`http://localhost:3001/livres/${livreConcerner.id}`)
  //La promesse = reponse du serveur (json-server) a la requete
  .then(reponse => {
    //On declenche une confirmation
    window.confirm('Valider le supression de ce livre ?');
    //le debug
    console.log(reponse.data);
    //On rafraichis la page
    window.location.reload();
  })
  //Si la promesse n'est pas tenue on retourne une erreur
  .catch(erreur => {
    console.error('Erreur lors de la supression du livre !' + erreur)
  })
}
```

120. Le JSX dans le bloc livreConcerner != null

```
{/* si livre concerner retourne une valeur */}
{livreConcerner ?(
  <div className='container w-50 shadow p-3'>
    <h2 className='text-center text-danger bg-warning p-3 rounded'>Détails du livre : {livreConcerner.nomLivre}</h2>
      <div id='carte' className='card'>
      <img className='p-3' src={livreConcerner.imageLivre} alt={livreConcerner.nomLivre} title={livreConcerner.nomLivre}/>
        <div className='card-body'>
          <h3 className='text-danger'>{livreConcerner.nomLivre}</h3>
          <p className='card-text'>{livreConcerner.descriptionLivre}</p>
          <p>PRIX: {livreConcerner.prixLivre}</p>
          <p>{livreConcerner.categoriesLivre}</p>
          <button className='btn btn-info'>EDITER</button>
          <button className='btn btn-danger mx-3' onClick={supprimerLivre}>SUPPRIMER</button>      You, maintenant • Uncomm
          <button className='btn btn-success' onClick={() => window.location.reload()}>RETOUR</button>
        </div>
      </div>
  </div>
):(
  <div className='row'>

    {/* Sinon on afficher tous les livres */}
```

121. Lors de la suppression on déclenche une alerte


122. ÉDITER UN PRODUIT

123. Au clic sur le bouton plus d'infos : on affiche les détails d'un livres

124. Ajouter le code suivant a votre JSX dans le bloc livre concerner = true du fichier Livres.js

```jsx
{/* FORMULAIRE EDITER UN LIVRE */}
<div id="edit-form-livre" className="mt-3 container">
        <h2 className="title is-2 has-text-success has-text-centered">EDITER LIVRE</h2>
        <div className="field">
            <label className="label">Nom du livre</label>
            <input
                type="text"
                className="input"
                id="nomLivre"
                name="nomLivre"
                placeholder="Nom du livre"
                required
                value={livre.nomLivre}
                onChange={handleInputChange}      You, il y a 1 seconde • Uncommitted changes
            />
        </div>

        <div className="field">
            <label className="label">Description du livre</label>
            <textarea
                className="input"
                id="descriptionLivre"
                name="descriptionLivre"
                required
                rows="5"
                placeholder="Description du livre"
                value={livre.descriptionLivre}
                onChange={handleInputChange}
            />
        </div>

        <div className="field">
            <label className="label">Prix du livre</label>
            <input
                type="number"
                step='0.01'
                className="input"
                id="prixLivre"
                placeholder="Prix du livre"
                name="prixLivre"
                required
                value={livre.prixLivre}
                onChange={handleInputChange}
            />
        </div>

        <div className="field">
            <label className="label">Image du livre</label>
            <input
                type="text"
                className="input"
                id="imageLivre"
                name="imageLivre"
                placeholder="URL de l'image du livre"
                required
                value={livre.imageLivre}
                onChange={handleInputChange}
            />
        </div>

        <button className="button is-danger" onClick={updateLivre}>Mettre à jour le livre</button>
    </div>
```

125. On doit créer 2 fonctions : handleInputChange() et updateLivre()

126. La fonction handleInputChange()

```jsx
//Etat des champs du formulaire editer un livre      You, il y a 1 seconde • Uncommitted changes
//fonction de tracking pour reperer les changements dans les inputs et les changements d'etats du DOM
const handleInputChange = event => {
 //Recupération des attributs input name et value
 //Ceci est un affectation par decomposition soit :
 //const name = event.target
 //const value = event.target
 const {name, value} = event.target;

 //spread Operator (extrait des données d'un tableau ou un objet et creer un nouveau tableau)
 //Appel du mutateur setLivre
 //On decompose (eclate) l'objet livre puis l'attribut name du champ <input> est egale a l'attribut value = event.target
 setLivreConcerner({
   ...livreConcerner,
   [name]: value
 })
 console.log(setLivre)
}
```

127. La fonctions updateLivre()

```
//Mettre a jour un livre a l'aide du formulaire
function updateLivre(){
  //Creer un bojet qui va remplacer le livre courant
  let remplacerLivre = {
        id: livreConcerner.id,
        nomLivre: livreConcerner.nomLivre,
        descriptionLivre: livreConcerner.descriptionLivre,
        prixLivre: livreConcerner.prixLivre,
        imageLivre: livreConcerner.imageLivre
  }

  //la requete http + methode put + url (id dynamique livre concerner) + requete + reponse
  axios.put(`http://localhost:3001/livres/${livreConcerner.id}`, remplacerLivre)
  //la promesse = reponse
  .then(reponse => {
    //mutateur du hook = setter
    //on eclate l'objet pour acceder a chaque propriétés de ce dernier
    setLivreConcerner({
      ...livreConcerner
    })
    //Debug = f12 navigateur
    console.log(reponse.data)
    //rafraichir la page
    window.location.reload('/');
  })
  //Si la promesse n'est pas tenue on affiche une erreur
  .catch(erreur => {
    console.error('Erreur lors de la mise a jour !' + erreur)
  })
}
```

128. Résultat

129.

130.

131.


# BRAVO VOUS ÊTES UN DÉVELOPPEUR FULLSTACK REACT JS