

Développement Front

Développement Front

2

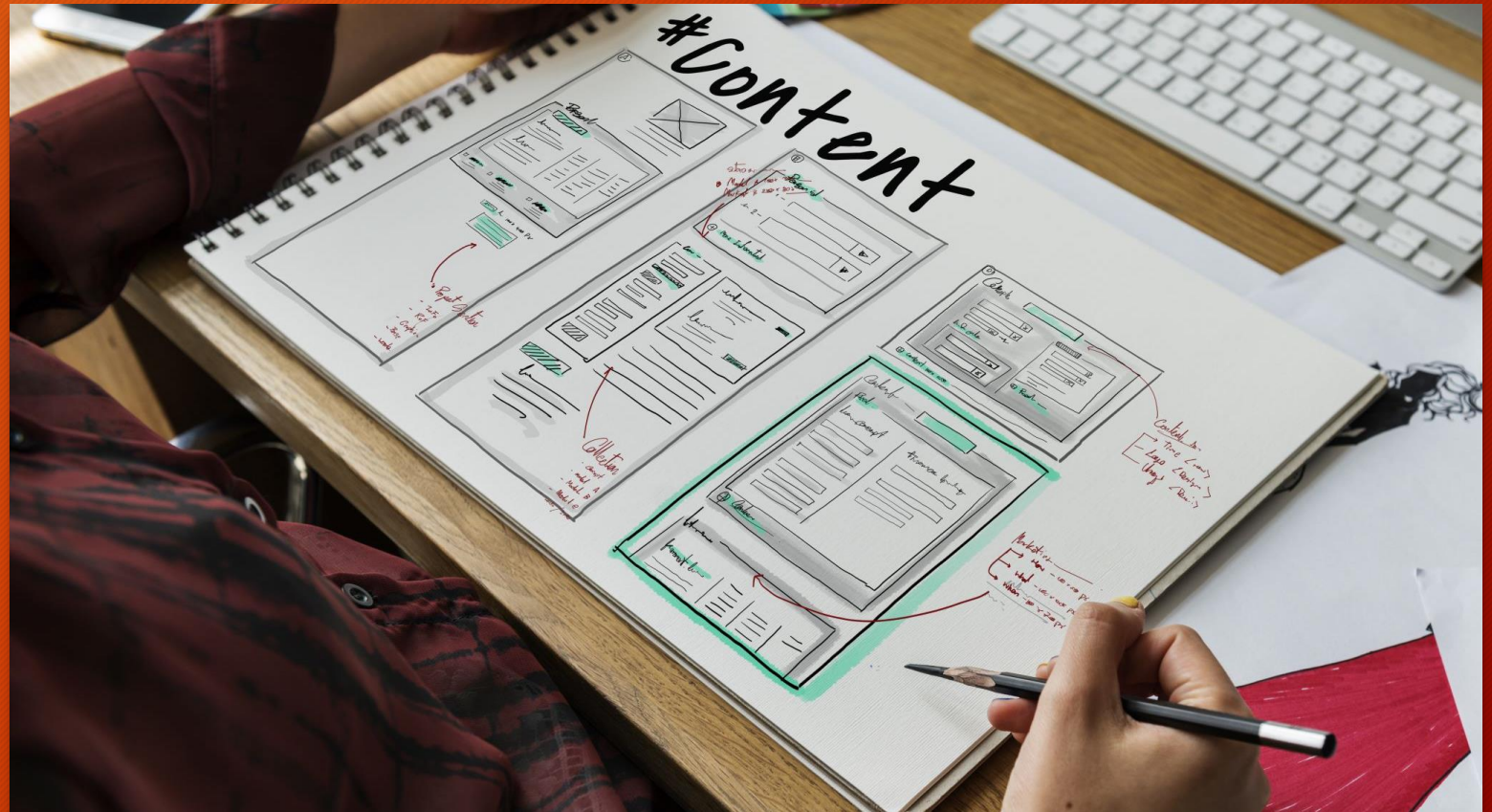
Design

Responsive

Accessibilité

SEO (Search Engine Optimization)

Bonnes pratiques



React js

3

Une bibliothèque de développement Front en js

Objectifs

4

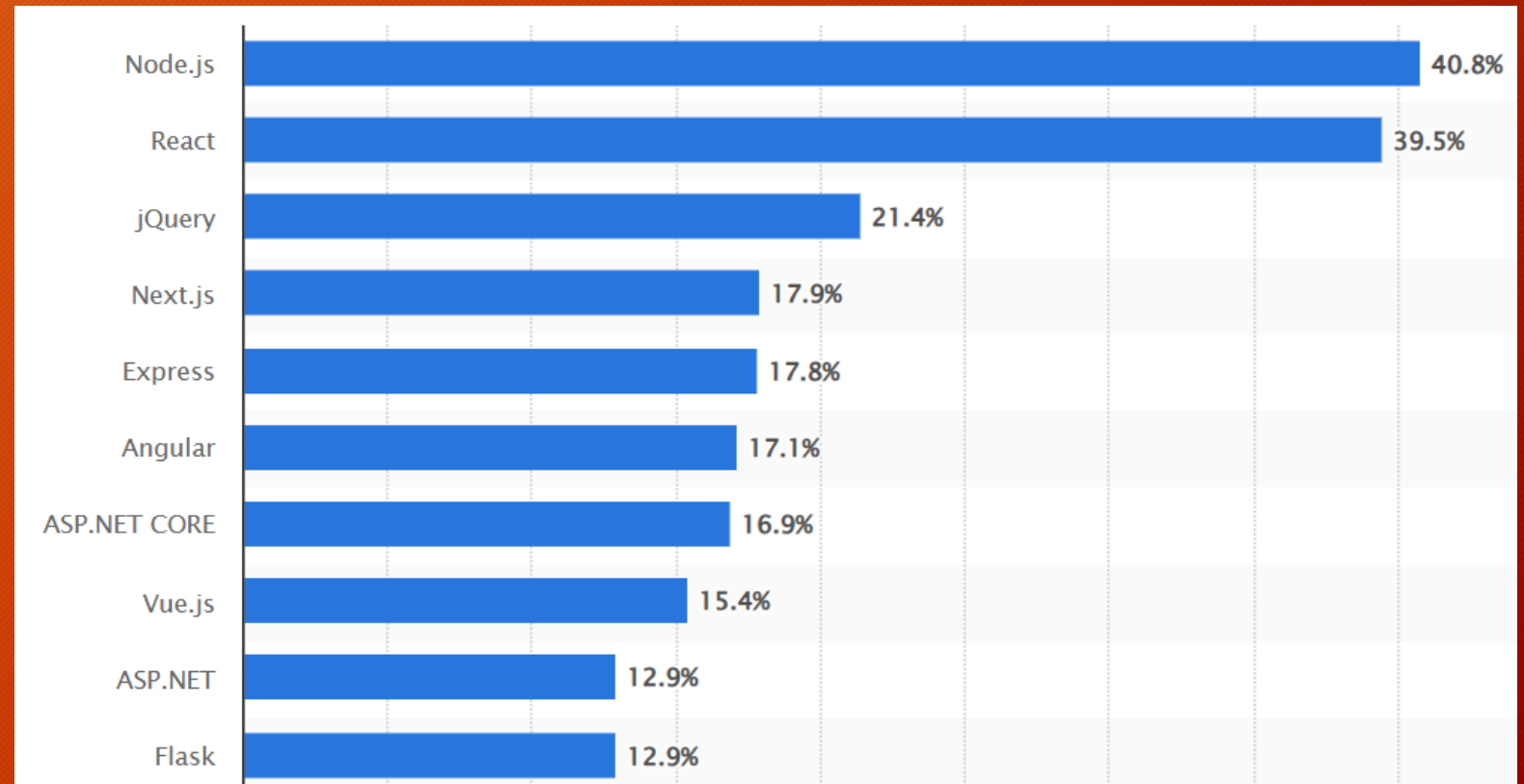
- Maîtriser une bibliothèque js dédiée au front
- Comme les bibliothèques sont très différentes
 - Savoir repérer les éléments importants quand on utilise une bibliothèque
 - Se raccrocher à la théorie

Les framework web les plus utilisés (2024)

5

Source

<https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/>



Top 5 des frameworks frontend web

6

- React
- Next.js (basé sur React, ajoute du traitement côté serveur)
- Angular
- Vue.js
- Svelte

React js

7

- Bibliothèque javascript
- Dédiée au Front
- Fonctionne côté client (Rien côté serveur)
- Doit être compilé en js pour être interprété par un navigateur (transpilation)
- Peut être associé à d'autres éléments dans une stack technique

React js

8

- Au-dessus du bas niveau html, css, js
- Intégration de structures html dans du code javaScript (jsx)
- Système de routage de navigation
- Création de vues dynamiques
- Intégration avec des API externes pour les données
- Possibilité d'animer

Rappels Web

9

DOM

DOM - Document Object Model

10

- Lors du chargement d'une page Web, Le navigateur crée un DOM de la page
- HTML DOM est un objet standard servant d'API pour HTML
- DOM définit
 - Les éléments HTML comme des objets.
 - Les propriétés des éléments HTML
 - Les accesseurs à tous les éléments HTML
 - Les événements attachés aux éléments HTML
- HTML DOM est un standard portant sur comment accéder, modifier, ajouter ou supprimer des éléments HTML
- On retrouve donc les aspects hiérarchiques de HTML (enfants, parent, frères)

Javascript HTML DOM

11

- JavaScript est le langage de programmation du HTML et du Web
- DOM et JavaScript permettent la création de HTML dynamique
 - Modifier/ajouter/supprimer des éléments HTML et leurs attributs
 - Modifier le style CSS des éléments et des attributs
 - Réagir aux événements HTML existants dans la page
 - Créer de nouveaux événements

```
<!DOCTYPE html>
<html>
  <head>
    <title>TITLE GOES HERE</title>
  </head>
  <body>

    <h1>My First JavaScript</h1>

    <button type="button" onclick="document.getElementById('demo').innerHTML = Date()">
      Click me to display Date and Time.
    </button>

    <p id="demo"></p>

  </body>
</html>
```


Le standard est verbeux

12

DOM API (JavaScript)

```
document.getElementById("some-id")  
// <li id="some-id">Unique element</li>  
  
document.getElementsByTagName("p").length;  
// 4  
  
var reds = document.getElementsByClassName("red")  
// [<p class="red">Red Paragraph</p>]  
  
reds[0].innerText  
// "Red Paragraph"
```

D'autres bibliothèques aux dessus de js

13

D3.js

```
d3.select("#some-id")  
// [Array(1)]  
  
d3.selectAll("p").size();  
// 4  
  
var reds = d3.selectAll(".red")  
// [Array(1)]  
  
reds.text()  
// "Red Paragraph"
```

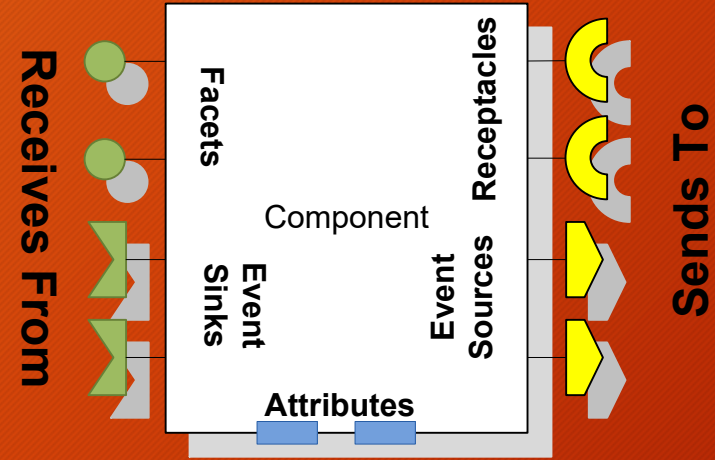
Retour sur React - Approche par composants

14

Ou presque

Modèle de composant

15



Modèle de composant de React

16

- On parle de « composant fonctionnel »
 - Le composant est du code jsx renvoyé par une fonction
- Source et Sink : Événements Javascript
 - On retrouve le fonctionnement des événements DOM classique
 - On utilise généralement des callbacks (ex. onClick)
- Système asynchrone de mise à jour des attributs
- On sort du paradigme objet
 - Tout passe par les paramètres de la fonction
 - Pas d'appel de méthode pour implémenter les facettes
- Le composant est potentiellement recalculé à chaque redessin
 - Par défaut, fonction appelés dans le workflow React

React Dataflow : communication Parents vers Enfants

17

- Sous forme de paramètres

```
function App() {  
  const [isCGUAccepted, setIsCGUAccepted] = useState(true)  
  
  return <form>  
    <CGUComp accepted={isCGUAccepted}/>  
    <button disabled={!isCGUAccepted}>Envoyer</button>  
  </form>  
}  
  
function CGUComp({accepted}) {  
  return <div><label>  
    <input type='checkbox' checked={accepted}/>  
    Accepter les conditions générales d'utilisation (CGU)  
  </label></div>  
}
```


React Reverse Dataflow : communication Enfants vers Parents

18

- Sous forme de callbacks passées paramètres

```
function App() {  
  const [isCGUAccepted, setIsCGUAccepted] = useState(true)  
  
  return <form>  
    <CGUComp accepted={isCGUAccepted} setAccepted={setIsCGUAccepted}/>  
    <button disabled={!isCGUAccepted}>Envoyer</button>  
  </form>  
}  
  
function CGUComp({accepted, setAccepted}) {  
  return <div><label>  
    <input type='checkbox' onChange={(e)=>setAccepted(e.target.checked)} checked={accepted}/>  
    Accepter les conditions générales d'utilisation (CGU)  
  </label></div>  
}
```

jsx

19

- Langage dédié à React js
- Ressemble à
 - Html
 - js
- N'est pas du
 - Html
 - Js
- Donc, il y a des différences de vocabulaire
- N'est pas interprétable par le navigateur
- Langage compilé produisant du js utilisant la bibliothèque React js
- Peut inclure des expression js dans du html-like

Jsx - exemples

20

- `const name = "David";`
- `const welcomeSubtitle = <h2>Bienvenue {name} !</h2>;`

```
const fruits = ["Apple", "Banana", "Cherry", "Date", "Elderberry"];

return (
  <ul>
    {fruits.map((fruit, index) => (
      <li key={index}>{fruit}</li>
    ))}
  </ul>
)
```


Exercice 1 - Affichage d'une carte utilisateur

21

Premiers pas avec jsx

Objectifs - Création d'un objet représentant une personne

22

- Création d'un projet
- Mise en place de variables servant de propriétés (name, age, email)
- Utilisation de collection, boucle, map pour représenter les sections variables (hobbies)
- Conditionnement du style

Structure de données

23

- `const user = {`
- `name: "John Doe",`
- `age: 40,`
- `email: "john.doe@gmail.com",`
- `hobbies: ["Reading", "Traveling", "Gaming"],`
- `};`

Objectif - Une carte magnifique !

24

Div « App »

Header « App-header »

Div « user-card »

- Rouge si < 18
- Orange si < 22
- Vert sinon

Carte Utilisateur

H1

John Doe

H2

Âge: 40 ans

P

Email: john.doe@gmail.com

P

Hobbies:

H3

Div

Reading

Traveling

Gaming

Div « hobby »

Div « hobby »

Div « hobby »

Configuration du projet

25

- Il faut
 - VS Code
 - Nodejs / npm
- Extension React Developer Tools du navigateur
 - Exploration des composants
 - Profiler pour les performances de la page
- On utilise Vite pour créer la structure du projet
 - Boiler-plate
 - <https://vitejs.fr/guide/>

Problème potentiel « Execution Policy »

26

- « Impossible de charger le fichier C:\Program Files\nodejs\npm.ps1, car l'exécution de scripts est désactivée sur ce système »
- Solution (dans powerShell ou en console vsCode)
 - Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser

Création d'un projet

27

- Prérequis
 - NodeJs installé
- Dans un dossier dédié
 - `npm create vite@latest`
 - Utilisation de Vite comme outil de génération front
- Suppression des éléments en trop
 - Dossier assets
 - Les svg
 - Les css
- Dans `main.jsx`
 - Suppression du `import index.css`
- Dans `App.jsx`

```
c > App.jsx > App
1
2
3 function App() {
4   return 'Bonjour'
5 }
6
7 export default App
8
```

Mise en œuvre

28

- Rapatrier la feuille App.css depuis moodle
- Ajouter l'importation dans App.jsx
- Tout le travail se fera dans la fonction App
 - Déclaration de l'objet user
 - Fonction de changement de rendu
 - Fonction de changement de style

Retour sur le fonctionnement de React

29

- Programmation réactive
 - Une modification impacte une partie du projet et pas l'ensemble
 - Asynchrone
- Virtual DOM
 - Changement dans le virtual DOM => changement dans le DOM
 - Evite des refresh inutiles
- Fonctionnement à base de « composants fonctionnels »

Css dans React

31

- Structuré autour des composants
 - Souvent limité au composant
 - Limite les conflits
 - Peut compliquer la cohérence
- Syntaxe différente du css classique
 - ATTENTION : camelCase au lieu de kebab-case
 - Exemple : background-color devient backgroundColor
- Inline styles (attribut style des éléments HTML)
 - Attention, styles complexes ou utilisation de pseudo sélecteur difficiles à gérer
- 1 fichier css par scope local (portée limitée à 1 composant)
 - Problème s'il existe des styles globaux
- Styled components
 - Création d'un composant avec du css dedans
 - Verbeux et crée des dépendances
- Les 3 solutions cohabitent

Framework css

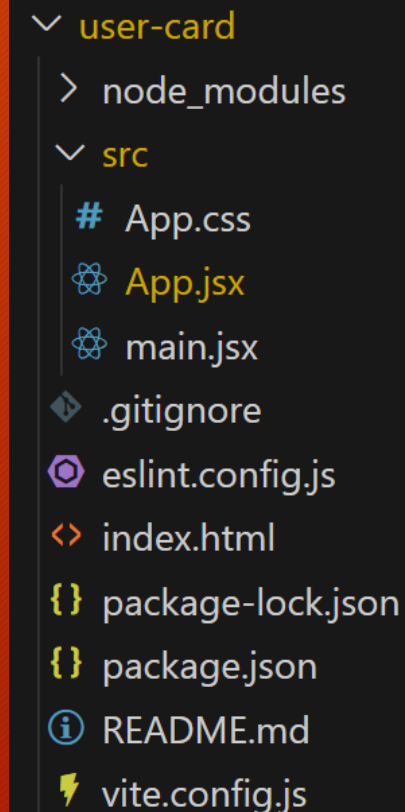
32

- Css pré-écrits
- Amène de l'homogénéité (y compris entre appareils)
- Gain de temps
- Exemple
 - Bootstrap (composants prêts à l'emploi)
 - Tailwind css (personnalisable)
 - Chakra UI (accent mis sur l'accessibilité)
 - ...

Architecture et Transpilation

33

- La base : index.html
 - Inclut l'appel au script main.jsx
- main.jsx
 - Fait l'insertion du résultat de App.jsx dans l'élément « root » de la page via l'API DOM
- App.jsx contient les composants React à rendre
- Transpilation
 - Agrège jsx, css pour produit html, css et js interprétable par un navigateur



```

user-card
├── node_modules
├── src
│   ├── App.css
│   ├── App.jsx
│   └── main.jsx
├── .gitignore
├── eslint.config.js
├── index.html
├── package-lock.json
├── package.json
├── README.md
└── vite.config.js

```

Syntaxe jsx

34

- Mélange de js et de pseudo-html
 - Syntaxe proche, mais peut différer sur les attributs
 - Toutes les balises doivent être fermées
- Le composant renvoie un nœud unique
 - qui peut être un `<Fragment/>` ou `</>`
 - si on ne veut pas d'un nœud typé
- Ajout d'une interprétation js dans le html
 - Entre `{}`

```
1  const title = 'Bonjour'
2
3  function App() {
4    return <div>
5      <h1>{title}</h1>
6      <p>Lorem ipsum dolor sit amet co
7    </div>
8  }
9
10 export default App
11
```

Les « Props »

- On peut fabriquer d'autres fonctions (ou composants)
 - Pour faciliter la lecture
 - Pour faciliter la réutilisabilité
- Un composant React peut avoir des paramètres (tableau props)
 - Pour améliorer la réutilisabilité
 - On utilise la déstructuration du tableau

35

```
App.jsx > ...
1  const title = 'Bonjour'
2
3  function App() {
4    return <div>
5      <Title/>
6      <p>Lorem ipsum dolor sit am
7    </div>
8  }
9
10 function Title() {
11   return <h1>{title}</h1>
12 }
13 export default App
14
```

```
App.jsx > ...
const title = 'Bonjour'

function App() {
  return <div>
    <Title color="red">{title}</Title>
    <p>Lorem ipsum dolor sit amet consectetur adipiscing
  </div>
}

function Title({ color, children }) {
  return <h1 style={{ color: color }}>{children}</h1>
}
export default App
```


La notion de Hook

36

- Fonctions de la bibliothèques React
- Hooks d'état local
 - Mémorise une valeur, permet la mise à jour, provoque le redessin
 - `useState`, `useReducer`
- Hooks de contexte
 - Permet la communication entre parents éloignés
 - Sans devoir créer une chaîne d'appels complexe
 - `useContext`
- Hooks de référencement
 - Permet de conserver des informations non utiles au rendu
 - Généralement un nœud du DOM
 - `useRef`, `useImperativeHandle`

La notion de Hook

37

- Hooks d'effet
 - Permet la synchronisation avec des éléments extérieurs
 - Réseau, DOM, animations, éléments d'interface écrits en utilisant une autre bibliothèque, et autres codes non React.
 - `useEffect`, `useLayoutEffect`, `useInsertionEffect`
- Hooks de performance
 - Permet de mettre en cache des calculs ou des définitions de fonctions pour éviter des recalculs inutiles du Virtual DOM
 - `useMemo`, `useCallback`, `useTransition`, `useDeferredValue`
- Hooks personnalisés
 - Création de nouveaux hooks

useState

38

Un hook d'état local
Le hook le plus utilisé

Premier Hook : useState + Render

39

- `Const [getter, setter] = useState(initialisation)`
 - Par convention, le getter prend le nom de la propriété
- Ex. `[count, setCount] = useState(0)`
- Render causé par la mise à jour d'un state
 - Rendu sur changement d'état, puis rendu des enfants
 - Modification du ReactDOM
 - Puis calcul par React des transformations nécessaires sur le DOM
 - Et Mise à jour des useState

Exercice 2 - Compteur

40

Base du projet sur moodle
Illustration du comportement asynchrone

Détails

41

- Tout dans le fichier App.jsx
- Un useState pour le compteur
- Un <bouton> pour déclencher
- Un <h1> pour l'affichage de count
- Quelle est la différence d'effet entre la ligne de code commentée et la dernière ligne ?

```
const increment = () => {  
  setCount(count + 1);  
  //setCount(count + 1);  
  setCount((c) => c + 1);  
}
```


Exercice 3 - Formulaire

42

Base du projet sur moodle
Champs contrôlés vs. Champs non contrôlés

IMPORTANT : Cas des formulaires

43

- Différence entre champs contrôlés et non contrôlés

```
const [firstName, setFirstname] = useState('')

const handleFirstname = (evt)=>{
  | setFirstname(evt.target.value)
  |
}

const handleSubmit = (evt)=> {
  | evt.preventDefault()
  | var formData = new FormData(evt.target)
  | console.log(formData.get('lastname'));
  |
}

console.log('render')

return <form onSubmit={handleSubmit}>
  | <input type='text' name='firstname' value={firstName} onChange={handleFirstname}/>
  | <input type='text' name='lastname' />
  | <button>Validate</button>
  | </form>
```

Exercice 4 - Gestion de produits

44

Base du projet sur moodle
Un exemple concret

Recherche et Filtres

Rechercher ...

☐ Afficher seulement produits en stock

Prix max : 4.55

Gestion de produits

Catégorie	Nom	Prix
Boisson		
	Eau	0.2
	Limonade	1.8
Fruit		
	Banane	2.4
	Cerise	3.4
	Kiwi	4.55
	Pomme	1.6
Légume		
	Carotte	1.2
	Chou	4.5
	Célerie	2.9

Structure de l'interface

46

▼ src

▼ components

▼ products

🔗 RowProduct.jsx

🔗 TableCategory.jsx

🔗 TableProducts.jsx

▼ searchBar

🔗 Checkbox.jsx

🔗 Input.jsx

🔗 Range.jsx

🔗 SearchBar.jsx

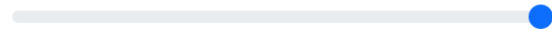
🔗 App.jsx

🔗 main.jsx

Recherche et Filtres

☐ Afficher seulement produits en stock

Prix max : 4.55



Gestion de produits

Catégorie	Nom	Prix
Boisson		
	Eau	0.2
	Limonade	1.8
Fruit		
	Banane	2.4
	Cerise	3.4
	Kiwi	4.55
	Pomme	1.6
Légume		
	Carotte	1.2
	Chou	4.5
	Célerie	2.9

Objectifs

47

- Dessiner le modèle de composant avec Corba CCM
 - L'idée est de concevoir des composants simples ET réutilisables
 - CheckBox, InputText, ...
- Implémenter UI-First
 - Implémenter chaque sous-composants de SearchBar
 - Implémenter SearchBar
 - Implémenter chaque sous-composants de TableProducts
 - Implémenter TableProducts
 - Utilisation de class CSS de Bootstrap
- Finir l'implémentation

Les autres hook

48

Hook - useEffect

49

- Création d'effets de bord
 - Surveillance d'un ensemble de variable
 - Callback appelée sur chaque changement de ces variables
- Rend les variables observables
- Limite les redessins
- Peut servir à initialiser des abonnements globaux
 - Penser au désabonnement

```
const exampleUseEffect = useEffect(() => {  
  console.log(`Rendu à chaque changement de : ${firstName}`);  
  return () => {  
    console.log(`Nettoyage avant le prochain rendu ou avant la fin du composant`);  
  }  
}, [firstName]);
```

- Utilisation à limiter
- Eviter les setters dans le useEffect

Compteur avec useEffect

50

- Démo
- Code disponible sur Moodle

Hook - useMemo

51

- Permet de créer un déclencheur sur changement effectif
 - Permet d'éviter des rendus inutiles sur changement des autres états
- Même syntaxe que useEffect
 - En plus, retourne une valeur
- « Memoisation »
 - Mise en cache d'une valeur
 - Réaction sur changement
- N'utiliser que si nécessaire
 - Pas d'optimisation préventive

```
const [firstname, setFirstname] = useState('John')
const [password, setPassword] = useState('MotDePasse')
const security = useMemo(() => {
  return passwordSecurity(password)
}, [password])
```

Hook - useId

52

- Générateur d'un id unique basé sur le ReactDOM

Hook - useRef

53

- Référence à un élément html
- Peut être utiliser comme un useMemo
- A utiliser avec un useEffect

Hook personnalisé

54

- C'est une fonction qui utilise des hooks
 - Peut renvoyer autre chose qu'un tableau
- Permet de faire des fonctions de plus haut niveau
 - Ex : `const [checked, toggleChecked] = useToggle(false)`
 - Ex : `const {count, increment, decrement} = useIncrement({base: 0, max: 10, min: 0})`
 - Ex : `useFetch`, fonction de chargement de fichier
- Permet d'isoler une partie de la logique de l'application
- Cf. <https://usehooks.com>
- Exemple d'implémentations dans la version typeScript
- Cf. react-use sur GitHub

Fonctionnement et optimisation

55

- Rappel : Principe de rendu de React
 - Exécution de fonctions
 - Restitue un virtual Dom
 - Compare avec la version précédente
 - Met à jour le DOM
- Utilisation du profiler
- Utilisation de useMemo et memo pour contrer les render inutiles

Exemple de « mémorisation » d'un composant

56

- Base du code sur Moodle
 - XXX
- Step 1 : on regarde en console l'effet de la modification de la zone de texte
 - Pourquoi ?
 - Utilisation du profileur du navigateur (lancer enregistrement d'une action)
 - Est-ce grave ?
- Step 2 : ajout d'un traitement long dans le composant Info
 - => impacte les performances de la zone de texte
 - Pourquoi ?
 - Utilisation du profileur du navigateur (lancer enregistrement d'une action)
 - Est-ce grave ?

Exemple de « mémoisation » d'un composant

57

- Pour améliorer la performance, deux solutions
- Solution 1 - Réorganisation du code (App2.jsx)
 - On isole les parties impactées par un changement d'état
 - Fonctions séparées
 - Limite les redessins des éléments indépendants
- Solution 2 - Mémoisation (App3.jsx)
 - Sauvegarde des paramètres et des résultats d'une fonction
 - Permet d'éviter la réexécution de la fonction pour les mêmes paramètres

Exemple de « mémoisation » d'un composant

58

- Maintenant, on ajoute une dépendance entre App et Info (App4.jsx)
 - Ex: Ajout d'un handleClick passé en param à Info
 - Retour du lag alors que la fonction ne « change » pas
- Solution 1 - Mémoisation avec useMemo (App5.jsx)
 - Plus adapté aux variables qu'aux fonctions
- Solution 2 - Mémoisation avec useCallback(App6.jsx)
 - Simplification du useMemo pour les fonctions
- Si trop de dépendances, on peut aussi utiliser des useRef
- Globalement, ne « Mémoiser » que quand il y a des pbs de performance