REACT

Thinking in React

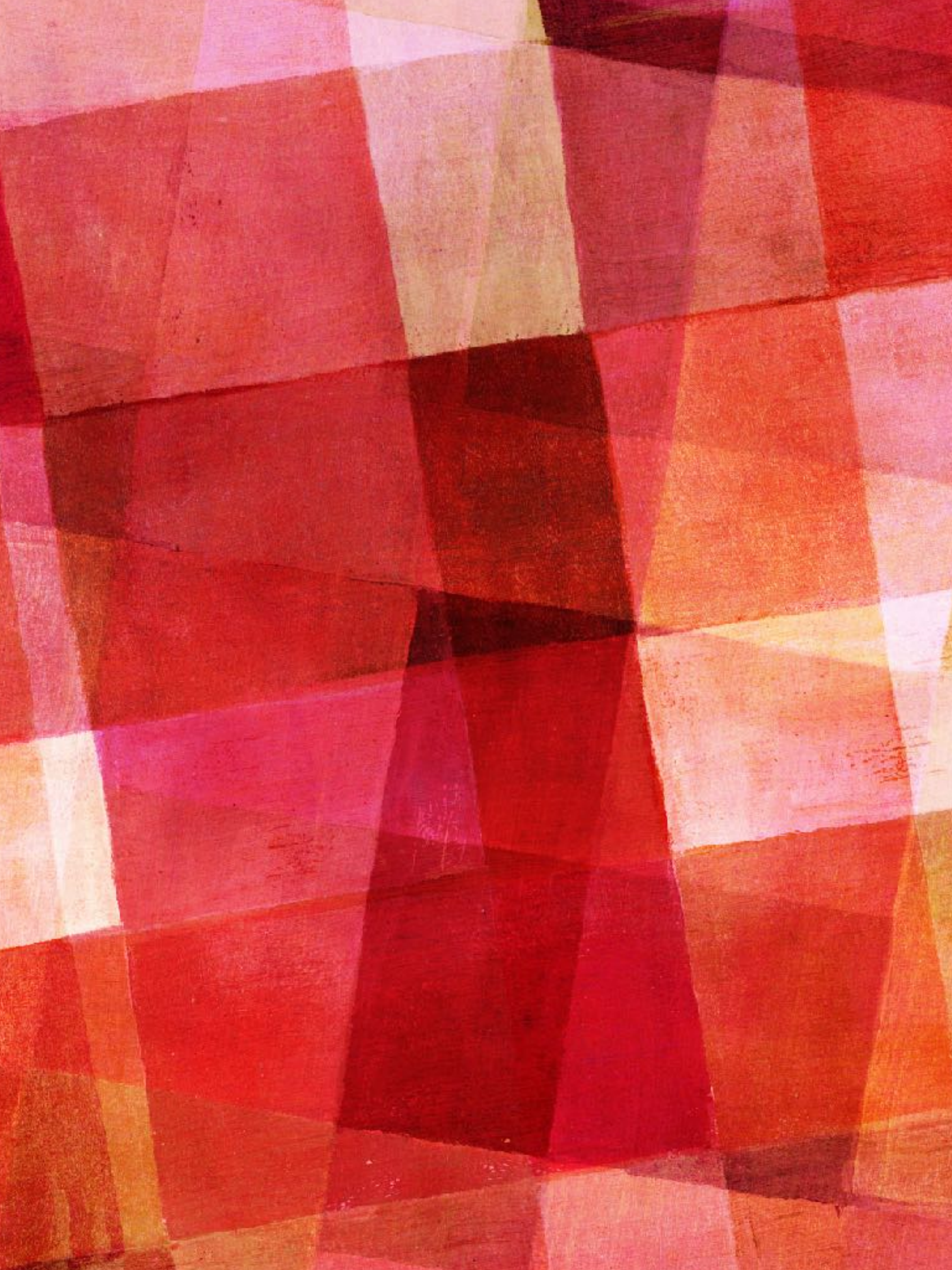
INTRO : REACT

- 1-way data binding qui rend l'interface plus réactive que dans les frameworks qui font du 2-way (2 watchers)

INTRO : SETUP

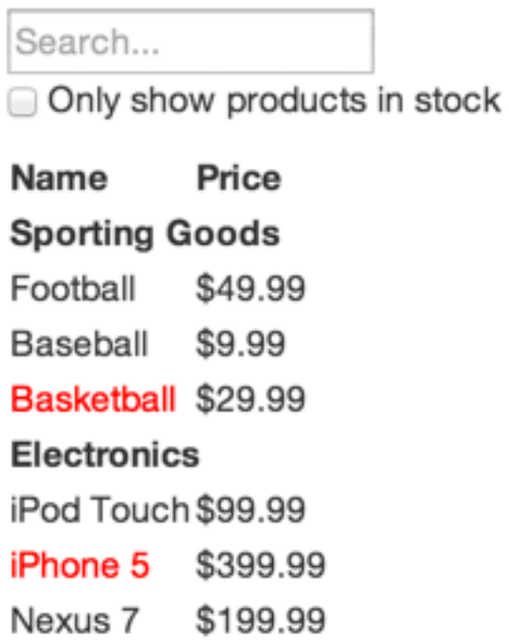
- Installer node et create-react-app

```
> npm install -g create-react-app
> create-react-app my-app
> cd my-app/
> npm start
```



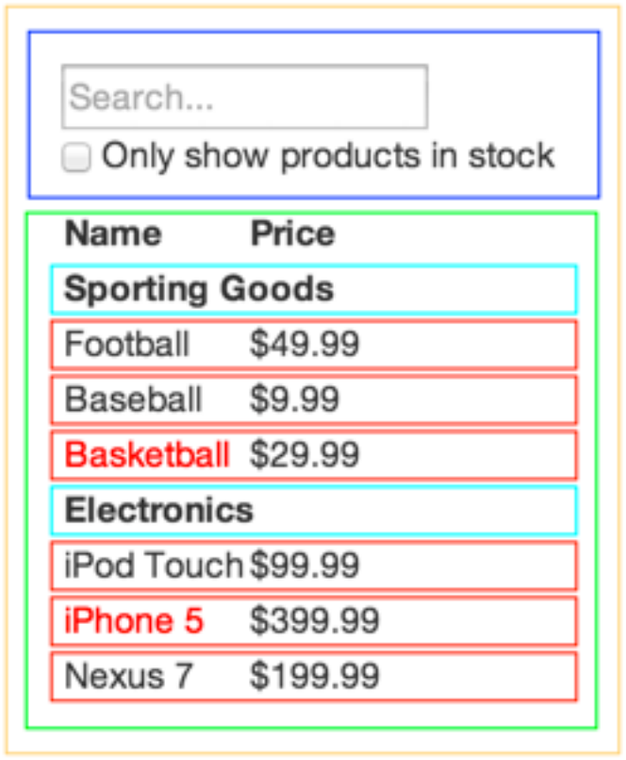
COMMENCER AVEC UN MOCK

Imaginer que nous avons un template de notre designer et une API JSON




```
[
  {category: "Sporting Goods", price: "$49.99", stocked: true, name:
"Football"},
  {category: "Sporting Goods", price: "$9.99", stocked: true, name:
"Baseball"},
  {category: "Sporting Goods", price: "$29.99", stocked: false, name:
"Basketball"},
  {category: "Electronics", price: "$99.99", stocked: true, name:
"iPod Touch"},
  {category: "Electronics", price: "$399.99", stocked: false, name:
"iPhone 5"},
  {category: "Electronics", price: "$199.99", stocked: true, name:
"Nexus 7"}
];
```

ETAPE 1 : DIVISER L'UI EN UNE HIÉRARCHIE DE COMPOSANTS



Nous avons donc 5 composants dans notre application

1. **FilterableProductTable** (orange) : contient la totalité de l'exemple
2. **SearchBar** (bleu) : reçoit les inputs
3. **ProductTable** (vert) : affiche et filtre les données en fonction des inputs de l'utilisateur
4. **ProductCategoryRow** (turquoise) : affiche le nom de chaque catégorie
5. **ProductRow** (rouge) : affiche une ligne pour chaque produit

ETAPE 2 : CONSTRUIRE UNE VERSION STATIQUE EN REACT

- Construire une version statique demande d'écrire beaucoup et de réfléchir peu, là où ajouter de l'interactivité demande de réfléchir beaucoup et d'écrire peu
- Construire des composants qui vont réutiliser d'autres composants et passer de la donnée en utilisant des props
- Construire top-down ou bottom-up : en partant des composants du haut de la hiérarchie ou du bas
- In fine, nous aurons donc une bibliothèque de composants réutilisables qui permettra de render de notre modèle de données

INTERLUDE : PROPS VS STATE

- En React 2 types de stockage pour la donnée
- Les props sont passés par les parents du composant
- Le state est réservé à l'interactivité et devrait ne contenir que la donnée qu'un eventHandler pourra modifier pour déclencher une mise à jour de l'UI
- Le state ne devrait pas contenir de donnée :
  - Calculable à partir des props
  - Dupliquant des props

ETAPE 3 : IDENTIFIER LA REPRÉSENTATION MINIMALE DU STATE

- Nous devons identifier le plus petit ensemble de données mutables dont notre application a besoin
- Par exemple si on a une TODO list, le state ne devrait contenir qu'une array des éléments de la TODO list. Pas le nombre d'items dans la TODO list par exemple qui est facilement calculable avec array.length
- Si on pense à toute la data que nous avons dans notre app :
  - La liste initiale des produits
  - Le texte dans la barre de recherche entré par l'utilisateur
  - La valeur de la checkbox
  - La liste filtrée des produits

ETAPE 3 : SUITE

- Si on pense à toute la data que nous avons dans notre app :
  - La liste initiale des produits
  - Le texte dans la barre de recherche entré par l'utilisateur
  - La valeur de la checkbox
  - La liste filtrée des produits
- Il suffit de se poser 3 questions pour chaque :
  1. Est-ce qu'elle est passée via des props ? Si oui, pas state
  2. Est-ce qu'elle ne change pas dans le temps ? Si oui, pas state
  3. Est-ce qu'on peut la calculer à partir d'un autre state ou props du composant ? Si oui, pas state

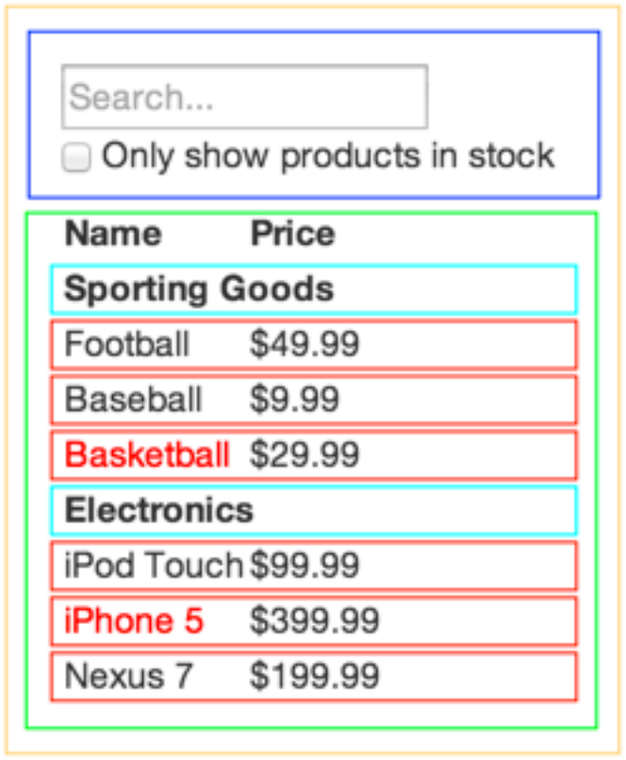
ETAPE 3 : ET FIN

- Si on applique ces questions à notre data :
  - La liste initiale des produits : props donc pas state
  - Le texte dans la barre de recherche entré par l'utilisateur : change dans le temps et ne peut pas être calculé donc state
  - La valeur de la checkbox : idem
  - La liste filtrée des produits : peut être calculé en combinant la liste initial des produits, le texte dans la barre de recherche et la valeur de la checkbox, donc pas state
- **Au final notre state c'est : le texte dans la barre de recherche et la valeur de la checkbox**

ETAPE 4 : IDENTIFIER OÙ LE STATE DEVRAIT VIVRE

- Identifier chaque composant qui render quelque chose basé sur le state
- Trouver un composant commun à tous ces composants
- Ce composant ou tout autre composant plus haut dans la hiérarchie devrait être propriétaire de ce state
- Si ça n’a pas sens d’avoir ces informations sur l’un des composants communs existants, insérer un nouveau composant dédié à héberger le state dans la hiérarchie

ETAPE 4 : SUITE

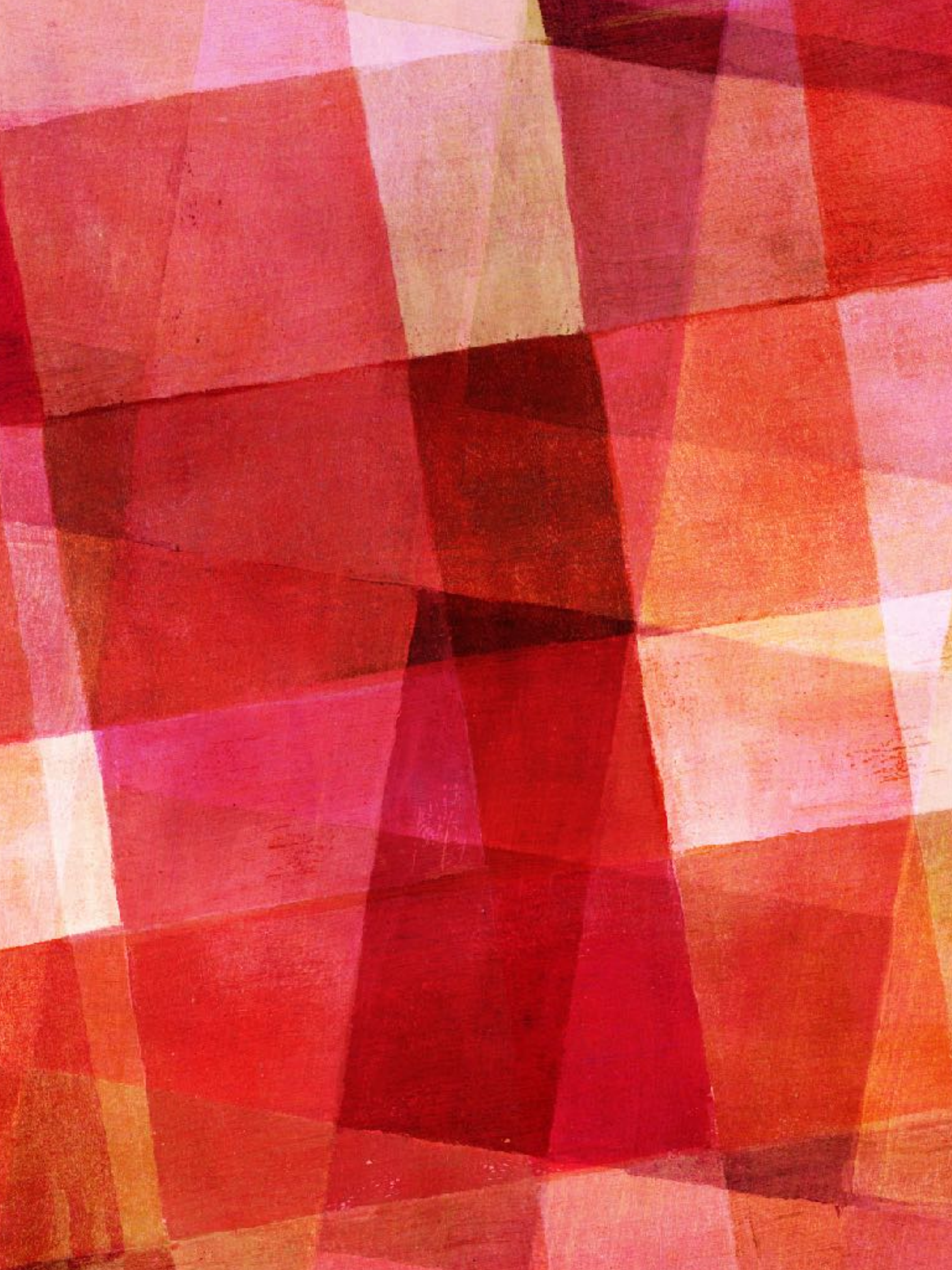


Dans notre application :

- ProductTable (vert) : doit filtrer la liste de produits en fonction du state et SearchBar (bleu) doit afficher le texte de la recherche et l'état de la checkbox du state
- Le composant commun à ces deux composants est FilterableProductTable (orange)

ETAPE 5 : AJOUTER LE FLUX DE DONNÉES INVERSE

- Les composants bas dans la hiérarchie doivent mettre à jour le state sur FilterableProductTable
- React rend ce flux de données explicite pour rendre plus facilement compréhensible le fonctionnement du programme
- Etant donné que les composants ne doivent mettre à jour que leur propre state FilterableProductTable doit passer un callback à SearchBar qui sera exécuté chaque fois que le state doit être mis à jour
- On peut utiliser l'event onChange pour en être notifié, le callback appellera setState() et l'application sera mise à jour



LIENS

- Codecademy : Learn ReactJS
- Create React apps with no build configuration
- Thinking in React
- React Stateless Functional Components: Nine Wins You Might Have Overlooked
- Removing User Interface Complexity, or Why React is Awesome
- Netflix JavaScript Talks: React plus X: Best Practices for Reusable UI Components
- Redux + Getting Started with Redux + You Might Not Need Redux