REACT PARA PRINCIPIANTES

Construyendo aplicaciones web dinámicas

Camila Valenzuela Fierro
Desarrolladora de Software
CEO & Co-founder Coding Chickens

coding chickens®

¿QUÉ ES REACT?

- React es una de las bibliotecas JavaScript más populares entre los desarrolladores Front End, programadores y testers de software.
- Te permite crear interfaces de usuario interactivas, ahorrando tiempo y reduciendo los costos de desarrollo.
- Desarrollado por Facebook en 2011 y lanzado al público en 2013.
- Es de código abierto y tiene una gran comunidad alrededor.

¿POR QUÉ FUE CREADO?

- A raíz de problemas significativos que Facebook estaba enfrentando con su interfaz de usuario.
- Por la necesidad de simplificar la construcción de dichas interfaces.
- Así como también, permitir la actualización más rápida de los componentes en la interfaz.

BENEFICIOS

- Su objetivo principal era simplificar el proceso de construcción de interfaces de usuario mediante la creación de una estructura que permitiera una actualización más rápida y sencilla de los componentes en la interfaz.
- Los componentes de React agilizan la creación de una interfaz sensible a cualquier cambio en un sitio web o una aplicación de cualquier complejidad.
- Gracias al DOM virtual, la biblioteca ahorra recursos y tráfico.
- El código de React tiene una lógica clara, es fácil de leer, entender y depurar, lo que ayuda a reducir errores.
- Las interfaces interactivas creadas con React garantizan una mejor experiencia de usuario.
- React es fácil de aprender, tiene una documentación accesible y muchos recursos gratuitos online.
- Dominar React es una de las habilidades más demandadas para conseguir el trabajo de desarrollo Front End.

¿CÓMO FUNCIONA?

1. Diseño en base a bloques reutilizables o COMPONENTES.
2. Los componentes representan una fusión entre la estructura de HTML y la funcionalidad de JavaScript.
3. Estos componentes se escriben en JSX, el cual es una extensión de la sintaxis de JavaScript.
4. A través de JSX, se crea una copia del DOM o modelo de objetos del documento o DOM Virtual.
5. React compara el VDOM con el DOM real y aplica el cambio sólo al elemento que ha sido actualizado, sin renderizar nuevamente toda la página.

CONCEPTOS CLAVE

COMPONENTE

Un componente es una pieza de UI (siglas en inglés de interfaz de usuario) que tiene su propia lógica y apariencia. Un componente puede ser tan pequeño como un botón, o tan grande como toda una página.

```
function MyButton() {
  return (
    <button>Soy un botón</button>
  );
}
```

```
export default function MyApp() {
  return (
    <div>
      <h1>Bienvenido a mi aplicación</h1>
      <MyButton />
    </div>
  );
}
```

CONCEPTOS CLAVE

PROPS Y ESTADO

- Las props son como argumentos que pasas a una función. Le permiten a un componente padre pasar datos a un componente hijo y personalizar su apariencia. Por ejemplo, un componente Form puede pasar una prop color a un componente Button.
- El estado es como la memoria de un componente. Le permite a un componente realizar un seguimiento de alguna información y cambiarla en respuesta a interacciones. Por ejemplo, un componente Button pudiera querer hacer un seguimiento del estado isHovered.

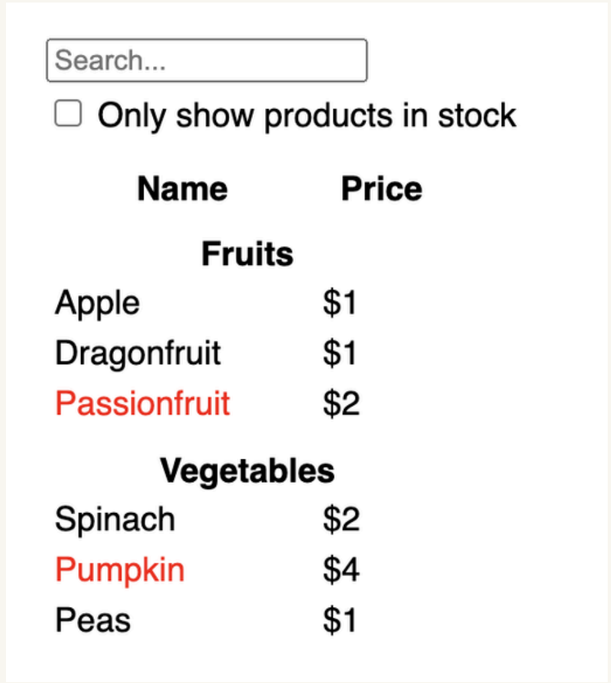
PENSANDO EN MODO REACT ⚛️💡

https://react.dev/learn/thinking-in-react

PASO 0: PARTAMOS DESDE EL DISEÑO

DOS ELEMENTOS: JSON Y BOCETO DE DISEÑO

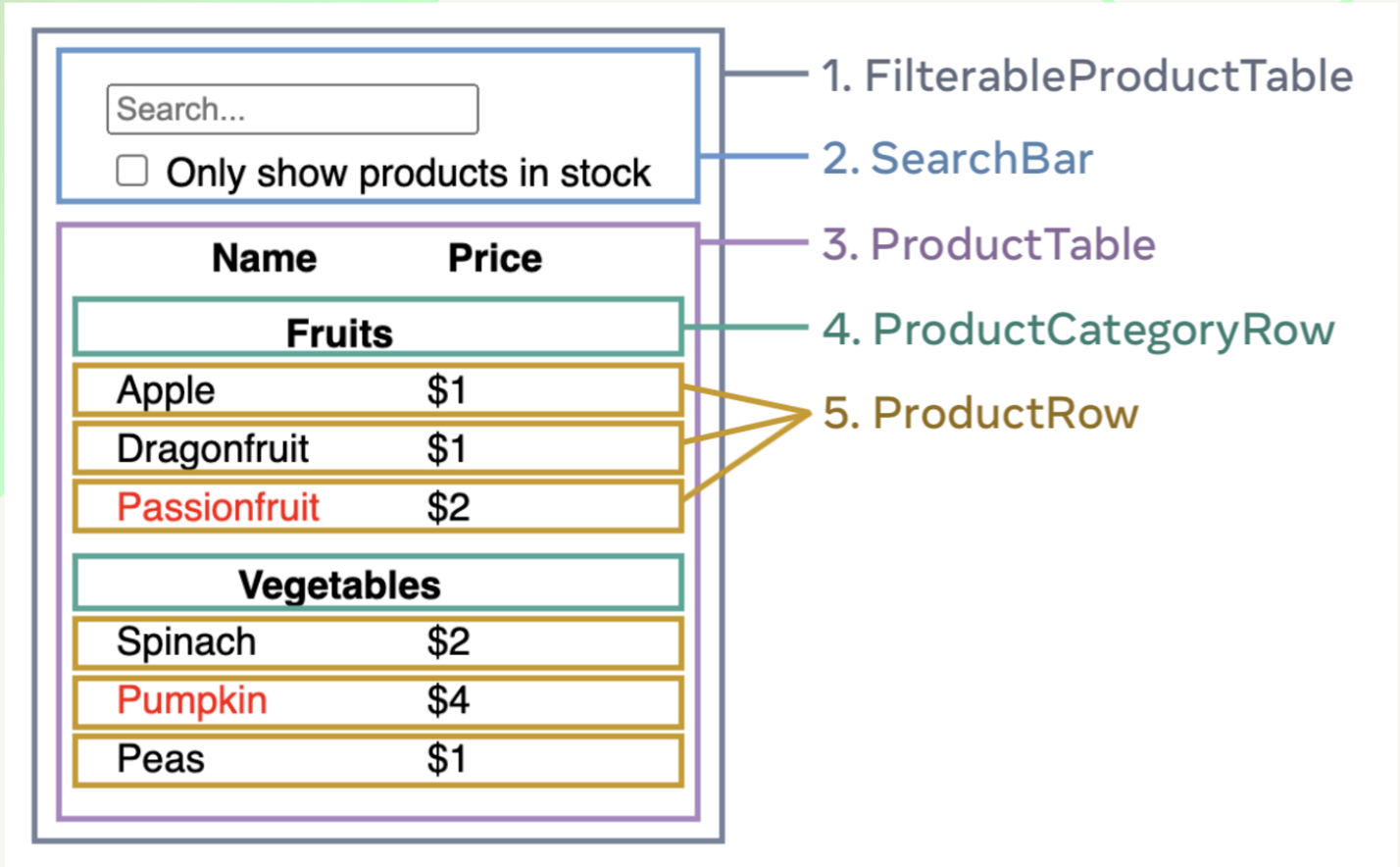
```
[
  { category: "Frutas", price: "$1", stocked: true, name: "Manzana" },
  { category: "Frutas", price: "$1", stocked: true, name: "Fruta del dragón" },
  { category: "Frutas", price: "$2", stocked: false, name: "Maracuyá" },
  { category: "Verduras", price: "$2", stocked: true, name: "Espinaca" },
  { category: "Verduras", price: "$4", stocked: false, name: "Calabaza" },
  { category: "Verduras", price: "$1", stocked: true, name: "Guisantes" }
]
```



PASO 1: SEPARA LA UI EN UNA JERARQUÍA DE COMPONENTES

Basarse en el diseño, JSON y/o dividir en base a los componentes que identifiquemos, en conjuntos más pequeños, que sean jerarquizables.

1. **FilterableProductTable** (gris) contiene toda la aplicación.
2. **SearchBar** (azul) recibe la entrada del usuario.
3. **ProductTable** (lavanda) muestra y filtra la lista de acuerdo a la entrada del usuario.
4. **ProductCategoryRow** (verde) muestra un encabezado para cada categoría.
5. **ProductRow** (amarillo) muestra una fila para cada producto.



JERARQUÍA

- FilterableProductTable
  - SearchBar
  - ProductTable
    - ProductCategoryRow
    - ProductRow

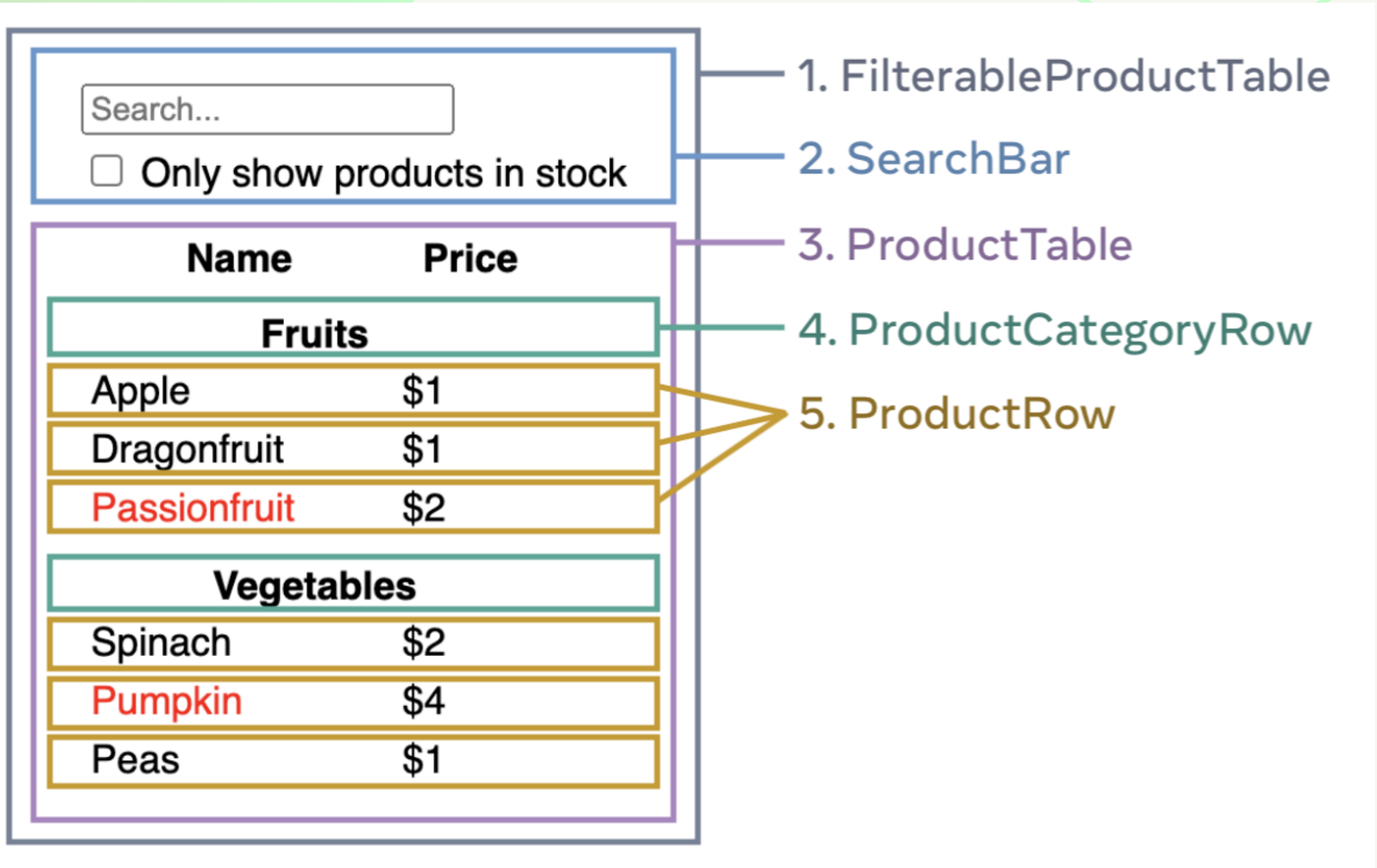
PASO 2: CONSTRUYE UNA VERSIÓN ESTÁTICA

Este es el momento de implementación, donde escribimos sin pensar aún en la interactividad de la aplicación.

Construir pensando en que los componentes que están más arriba en la jerarquía reutilicen otros componentes y pasen datos usando props.

Resultado:

- Biblioteca de componentes reutilizables que renderizan tu modelo de datos
- Flujo de datos en un sentido



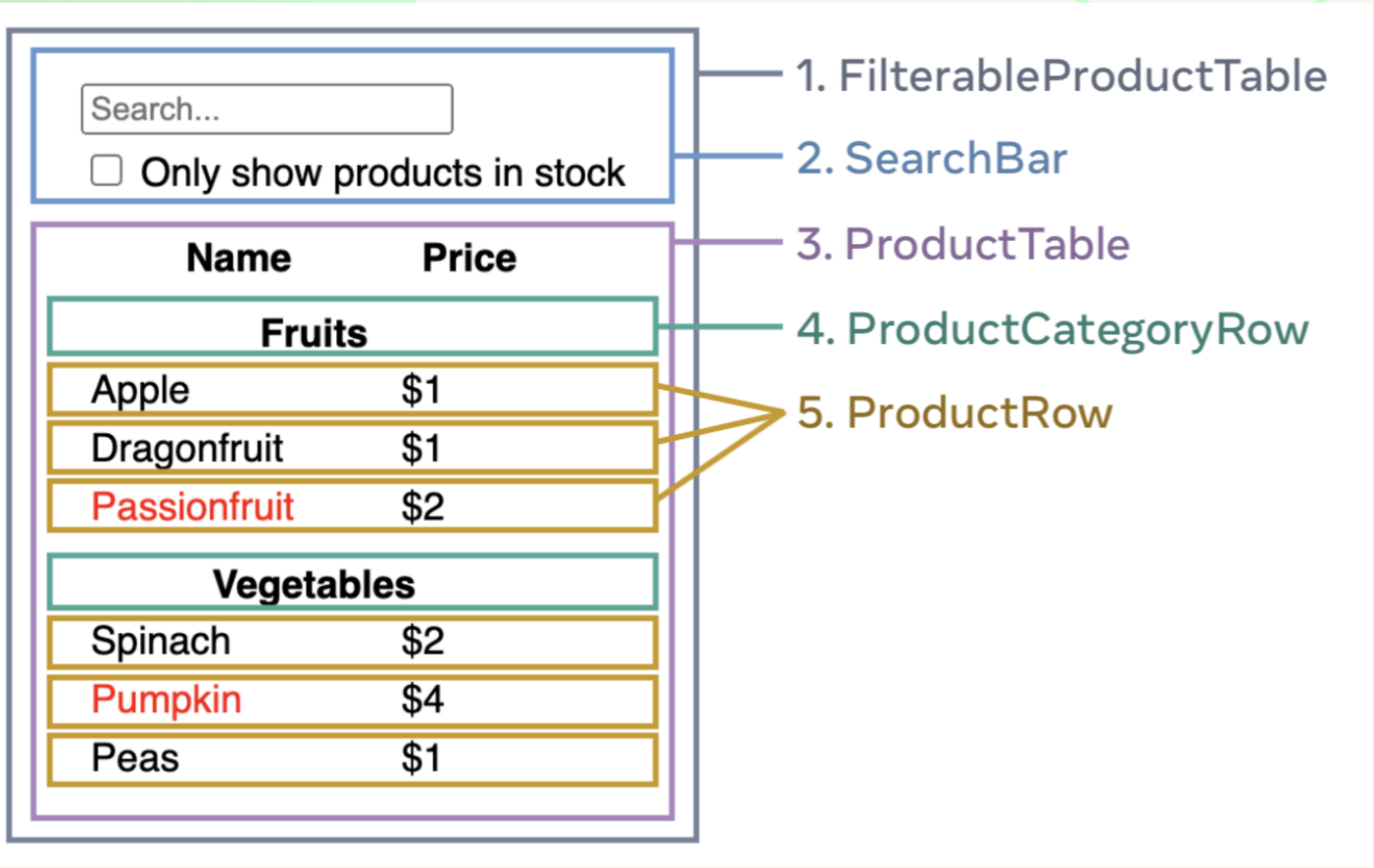
PASO 3: ENCUENTRA LA REPRESENTACIÓN MÍNIMA PERO COMPLETA DEL ESTADO DE LA UI

Recién acá pensamos en la interactividad, a través del ESTADO.

Este es el conjunto mínimo de datos cambiantes que la aplicación necesita recordar. Ejemplo: Si necesitas saber la cantidad de elementos en tu lista de compras, no la almacenes en la aplicación, calcúlala en base a tu arreglo de datos.

1. La lista original de productos
2. El texto de búsqueda que el usuario ha escrito
3. El valor del checkbox
4. La lista de productos filtrada

Preguntas clave:
¿Se mantiene sin cambios con el tiempo?
¿Se pasa desde un padre por props?
¿Puedes calcularlo basado en estado existente en props?



PASO 4: IDENTIFICAR DÓNDE DEBE VIVIR TU ESTADO

identifica qué componente es responsable de cambiar qué estado.

Pasos a seguir:

1. Identifica cada componente que renderiza algo basado en ese estado.
2. Encuentra su componente padre común más cercano, un componente que esté encima de todos en la jerarquía
3. Decide dónde debe residir el estado (padre, encima del padre, componente nuevo).

Resultado:

1. Identifica componentes que usen estado:
   - ProductTable necesita filtrar la lista de productos con base en ese estado (texto de búsqueda y valor del checkbox).
   - SearchBar necesita mostrar ese estado (texto de búsqueda y valor del checkbox).
2. Encuentra su padre común: El primer componente padre que ambos componentes comparten es FilterableProductTable.
3. Decide donde reside el estado: Mantendremos el texto de filtrado y el estado de valor seleccionado en FilterableProductTable.

```
function FilterableProductTable({ products }) {
  const [filterText, setFilterText] = useState('');
  const [inStockOnly, setInStockOnly] = useState(false);
```

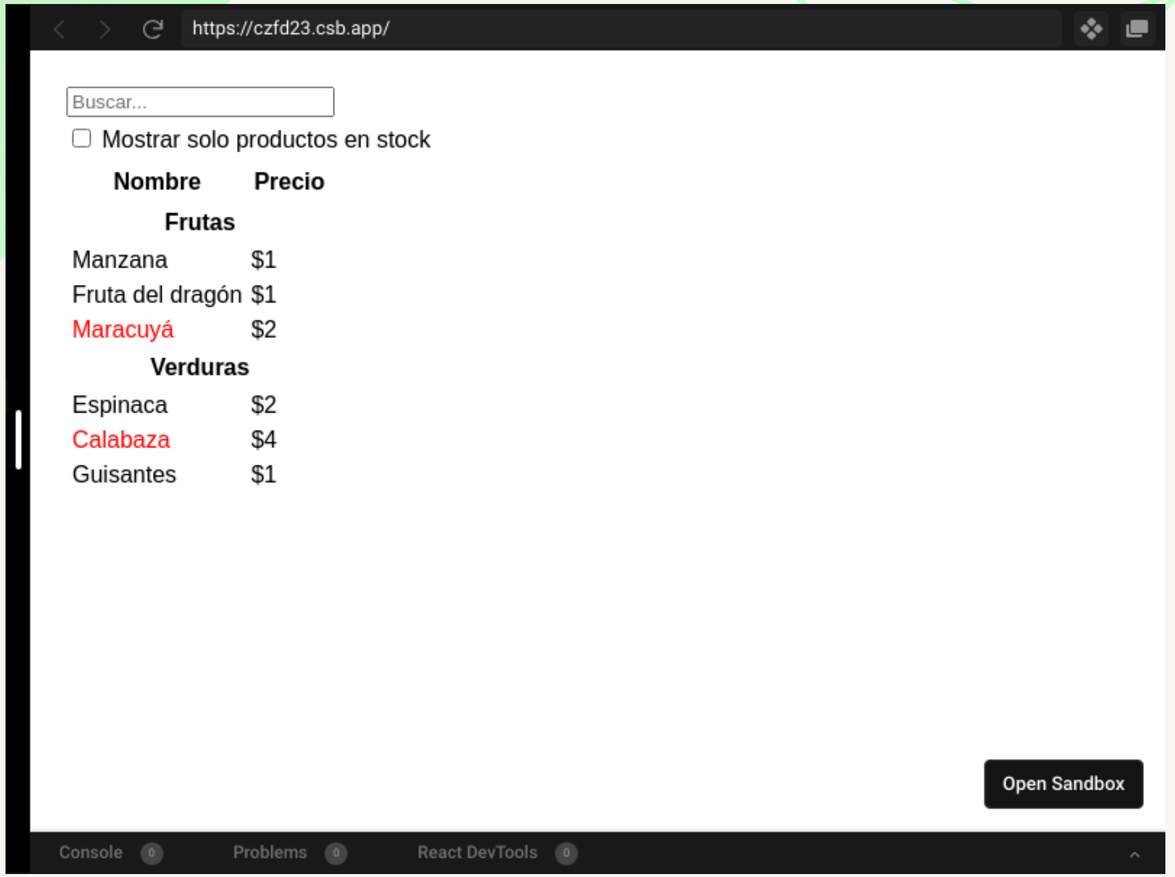
```
<div>
  <SearchBar
    filterText={filterText}
    inStockOnly={inStockOnly} />
  <ProductTable
    products={products}
    filterText={filterText}
    inStockOnly={inStockOnly} />
</div>
```

PASO 5: AÑADE FLUJO DE DATOS INVERSO

Debes lograr que cuando el usuario cambie las entradas del formulario, el estado se actualice para reflejar esos cambios. El estado lo posee FilterableProductTable, por lo que solo él puede llamar a setFilterText y setInStockOnly. Para permitir que SearchBar actualice el estado de FilterableProductTable necesitas pasar estas funciones para abajo hacia SearchBar:

```
function FilterableProductTable({ products }) {
  const [filterText, setFilterText] = useState('');
  const [inStockOnly, setInStockOnly] = useState(false);

  return (
    <div>
      <SearchBar
        filterText={filterText}
        inStockOnly={inStockOnly}
        onFilterTextChange={setFilterText}
        onInStockOnlyChange={setInStockOnly} />
```




```
function FilterableProductTable({ products }) {
  const [filterText, setFilterText] = useState('');
  const [inStockOnly, setInStockOnly] = useState(false);

  return (
    <div>
      <SearchBar
        filterText={filterText}
        inStockOnly={inStockOnly}
        onFilterTextChange={setFilterText}
        onInStockOnlyChange={setInStockOnly} />
```

JUNTOS

TRANSFORMAMOS EL FUTURO

¡Gracias por su atención!

coding chickens®

codingchickens.com

@codingchickens