Crypto List Filtering Logic

```
import React, { useState, useEffect } from 'react';
import millify from 'millify';
import { Link } from 'react-router-dom';
import { Card, Row, Col, Input } from 'antd';
import { useGetCryptosQuery } from '../services/cryptoApi';

const Cryptocurrencies = ({ simplified }) => {
    const count = simplified ? 10 : 100;//when we click on show more count becomes 100 since
simplified is false
    const { data: cryptosList, isFetching } = useGetCryptosQuery(count);
    const [cryptos, setCryptos] = useState([]);
    const [searchTerm, setSearchTerm] = useState('');

    // Set initial cryptos when API data is available
//     Because cryptosList is fetched asynchronously.
//  When the API call completes, cryptosList updates.
//  We need to update cryptos whenever cryptosList changes.
    useEffect(() => {
        if (cryptosList?.data?.coins) {
            setCryptos(cryptosList.data.coins);
        }
    }, [cryptosList]); // at first render cryptos is updated with cryptolist and useeffect is executed

    // Filter cryptos based on search term
    useEffect(() => {
        if (cryptosList?.data?.coins) {
            const filteredData = cryptosList.data.coins.filter(coin =>
                coin.name.toLowerCase().includes(searchTerm.toLowerCase())
            );
            setCryptos(filteredData);// again we have to update crptos only
        }
    }, [searchTerm, cryptosList]);// runs when searchTerm changes

    if (isFetching) return 'Loading...';

    return (
        <>
         {!simplified && (
                <div className='search-crypto'>
                    <Input
                        placeholder='Search Cryptocurrency'
                        onChange={(e) => setSearchTerm(e.target.value)}
                    />
                </div>
```

```
        )}


        <Row gutter={[32, 32]} className='crypto-card-container'>
          {cryptos?.map((currency) => (
            <Col xs={24} sm={12} lg={6} className='crypto-card' key={currency.uuid}>
              <Link to={`/crypto/${currency.uuid}`}>
                <Card
                  title={`${currency.rank}. ${currency.name}`}
                  extra={<img className="crypto-image" src={currency.iconUrl} alt=
{currency.name} />}
                  hoverable
                >
                  <p>Price: {millify(currency.price)}</p>
                  <p>Market Cap: {millify(currency.marketCap)}</p>
                  <p>Daily Change: {millify(currency.change)}</p>
                </Card>
              </Link>
            </Col>
          ))}
        </Row>
      </>
    );
};

export default Cryptocurrencies;

explain this code in details
that how re-rendering is happening for every time
how many times useEffect hooks are executed , explain the complete flow with an example
```

This `Cryptocurrencies` component is a React functional component that fetches and displays a list of cryptocurrencies. Let's go step by step to understand how it works and how re-rendering happens.

Component Breakdown

1. Props and State Initialization

```
js

const Cryptocurrencies = ({ simplified }) => {
    const count = simplified ? 10 : 100;
    const { data: cryptosList, isFetching } = useGetCryptosQuery(count);
    const [cryptos, setCryptos] = useState([]);
    const [searchTerm, setSearchTerm] = useState('');
```

- `simplified`: Determines whether to show only 10 cryptocurrencies (`true`) or 100 (`false`).
- `count`: Decides how many cryptocurrencies to fetch.
- `useGetCryptosQuery(count)`: Fetches cryptocurrency data asynchronously based on the `count` value.

- `cryptosList`: Holds the API response containing the cryptocurrency list.
- `isFetching`: A boolean flag indicating whether the data is still being fetched.
- `cryptos`: Stores the list of cryptocurrencies after filtering.
- `searchTerm`: Stores the user's search input.

---

2. First `useEffect`: Updating Cryptos When Data is Fetched

```
js

useEffect(() => {
    if (cryptosList?.data?.coins) {
        setCryptos(cryptosList.data.coins);
    }
}, [cryptosList]);
```

- This runs when `cryptosList` changes (i.e., when new data is fetched from the API).
- When the API call completes, `cryptosList` updates, triggering a re-render.
- This ensures that the `cryptos` state is updated with the latest coin data.

---

3. Second `useEffect`: Filtering Cryptos on Search

```
js

useEffect(() => {
    if (cryptosList?.data?.coins) {
        const filteredData = cryptosList.data.coins.filter(coin =>
            coin.name.toLowerCase().includes(searchTerm.toLowerCase())
        );
        setCryptos(filteredData);
    }
}, [searchTerm, cryptosList]);
```

- Runs whenever `searchTerm` or `cryptosList` changes.
- Filters the list of cryptocurrencies based on the user's search query.
- Updates the `cryptos` state, triggering a re-render.

---

4. Conditional Loading State

```
js

if (isFetching) return 'Loading...';
```

- If `isFetching` is `true`, the component returns `'Loading...'` without rendering further.

---

5. Rendering the Search Input (Only When `simplified` is `false`)

```
js

{!simplified && (
    <div className='search-crypto'>
        <Input
            placeholder='Search Cryptocurrency'
```

```
            onChange={(e) => setSearchTerm(e.target.value)}
        />
    </div>
)}
```

- The search input appears **only when `simplified` is `false`**.
- `onChange` updates `searchTerm`, triggering the second `useEffect`.

---

6. Rendering Cryptocurrency Cards

```
js

<Row gutter={[32, 32]} className='crypto-card-container'>
    {cryptos?.map((currency) => (
        <Col xs={24} sm={12} lg={6} className='crypto-card' key={currency.uuid}>
            <Link to={`/crypto/${currency.uuid}`}>
                <Card
                    title={`${currency.rank}. ${currency.name}`}
                    extra={<img className="crypto-image" src={currency.iconUrl} alt={currency.name}
                    hoverable
                >
                    <p>Price: {millify(currency.price)}</p>
                    <p>Market Cap: {millify(currency.marketCap)}</p>
                    <p>Daily Change: {millify(currency.change)}</p>
                </Card>
            </Link>
        </Col>
    ))}
</Row>
```

- `cryptos` (filtered list) is mapped to render cryptocurrency cards.
- Each card includes name, rank, image, price, market cap, and daily change.
- Clicking a card navigates to the individual cryptocurrency's page.

How Re-Rendering Happens

Re-rendering occurs when **state or props change**. Let's analyze when and why.

Initial Render

1. The component mounts.
2. `useGetCryptosQuery(count)` starts fetching data asynchronously.
3. `cryptosList` is initially `undefined`, so `isFetching` is `true`.
4. `"Loading..."` is displayed.

First Re-Render (After API Data is Fetched)

1. API call completes, updating `cryptosList`.
2. `useEffect` (first one) runs because `cryptosList` has changed.
3. `setCryptos(cryptosList.data.coins)` updates the `cryptos` state.
4. Since `cryptos` changes, the component **re-renders**.

Re-Render on Search Input Change

1. The user types something in the search bar.
2. `setSearchTerm(e.target.value)` updates `searchTerm`, triggering the second `useEffect`.
3. The second `useEffect` filters cryptocurrencies based on `searchTerm`.
4. `setCryptos(filteredData)` updates `cryptos`, triggering another **re-render**.

Re-Render on Clicking "Show More"

1. Clicking "Show More" makes `simplified` `false`, increasing `count` to `100`.
2. `useGetCryptosQuery(count)` fetches more data.
3. API response updates `cryptosList`, triggering both `useEffect` hooks.
4. `cryptos` updates, triggering a **re-render**.

How Many Times Does `useEffect` Run?

Scenario	First `useEffect` (on `cryptosList` change)	Second `useEffect` (on `searchTerm` or `cryptosList` change)
Initial render	✅ (Once, when data is fetched)	❌ (searchTerm is empty)
API fetch completes	✅ (Updates `cryptos`)	❌ (searchTerm is still empty)
User types in search	❌	✅ (Each keystroke triggers a filter)
Clicking "Show More"	✅ (New data fetched)	✅ (Filtering based on search)

- The **first `useEffect` runs once** per API fetch.
- The **second `useEffect` runs every time the user types in the search bar** or when new data is fetched.

Final Summary

1. **Mounting**: `"Loading..."` appears while fetching data.
2. **Data Fetch Complete**: API response updates `cryptosList`, which updates `cryptos`, causing a **re-render**.
3. **User Searches**: `searchTerm` changes, triggering filtering and another **re-render**.
4. **Show More Clicked**: New data fetch triggers both `useEffect` hooks, causing another **re-render**.

This ensures efficient rendering, keeping the UI responsive while minimizing unnecessary renders. 🚀

so what is the use of useEffect when we use ?
does we have to use useEffect when ever we need to re-render
what does side effects means

What is the use of `useEffect`?

`useEffect` is a React Hook that lets you perform **side effects** in function components. It is commonly used for:

- Fetching data from an API.
- Updating the DOM directly.
- Setting up subscriptions or event listeners.
- Managing timers (e.g., `setTimeout`, `setInterval`).
- Cleaning up resources when the component unmounts.

Does `useEffect` trigger a re-render?

No, `useEffect` itself **does not cause a re-render**. However:

- If `useEffect` updates state using `setState`, it **triggers a re-render**.
- If dependencies in the dependency array (`[]`) change, the effect **runs again**.

When should we use `useEffect`?

You should use `useEffect` when:

1. **You need to run code after a render.**
   - Example: Fetching data after the component mounts.
2. **You need to update state based on external data changes.**
   - Example: Filtering data when search input changes.
3. **You need to perform cleanup when a component unmounts.**
   - Example: Removing event listeners or stopping intervals.

What does "side effects" mean?

A **side effect** is any operation that **affects something outside of the function's scope**.
React renders components **purely based on props and state**, but side effects involve **external interactions**.

Examples of Side Effects:

- Fetching API data.
- Changing the document title.
- Logging to the console.
- Subscribing to a WebSocket.

How `useEffect` Works in Your Example

```
js

useEffect(() => {
    if (cryptosList?.data?.coins) {
        setCryptos(cryptosList.data.coins);
    }
}, [cryptosList]);
```

- Runs when `cryptosList` **changes** (after the API call finishes).
- Updates `cryptos`, which **triggers a re-render**.

```
js


useEffect(() => {
    if (cryptosList?.data?.coins) {
        const filteredData = cryptosList.data.coins.filter(coin =>
            coin.name.toLowerCase().includes(searchTerm.toLowerCase())
        );
        setCryptos(filteredData);
    }
}, [searchTerm, cryptosList]);
```

- Runs when `searchTerm` or `cryptosList` **changes**.
- Updates `cryptos`, triggering another **re-render**.

Key Takeaways

- `useEffect` **does not cause re-renders directly**—only state updates do.
- It runs **after** a render and only when dependencies **change**.
- Side effects are **external interactions**, like API calls or DOM updates.

Would you like a real-world analogy to understand it better? 🚀