React Concepts Summary - Crypto Dashboard Basics

1. JSX (JavaScript XML)

What it is: HTML-like syntax inside JavaScript functions

```jsx
// JSX - looks like HTML
return (
  <div className="App">
    <h1>Crypto Briefing Dashboard</h1>
    <p>Bitcoin Price: ${bitcoinPrice}</p>
  </div>
)

// Gets converted to regular JavaScript:
return React.createElement("div", {className: "App"},
  React.createElement("h1", null, "Crypto Briefing Dashboard"),
  React.createElement("p", null, `Bitcoin Price: $${bitcoinPrice}`)
)
```

Why we use it: Much easier to read and write than pure JavaScript!

2. useState Hook

What it does: Creates state variables that React watches for changes

```jsx
const [bitcoinPrice, setBitcoinPrice] = useState(null)
const [loading, setLoading] = useState(true)
```

Breakdown:

- `bitcoinPrice` = current value (starts as `null`)
- `setBitcoinPrice` = function to update the value (React creates this automatically)
- `useState(null)` = initial value is `null`
- When `setBitcoinPrice(50000)` is called, React re-renders the component

Array Destructuring: `useState()` returns `[value, updaterFunction]`, we extract both at once

3. useEffect Hook

What it does: Runs code at specific times in component lifecycle

```jsx
// Run ONCE when component first loads
useEffect(() => {
  fetchBitcoinPrice()
}, []) // Empty array = run once

// Run when 'coinId' changes
useEffect(() => {
  fetchCoinPrice(coinId)
}, [coinId]) // Run when coinId changes

// Run on EVERY re-render
useEffect(() => {
  console.log("Component updated")
}) // No array = run always
```

Why the dependency array matters:

- `[]` = "Run once when page loads" ✅
- `[variable]` = "Run when this variable changes" ✅
- No array = "Run constantly" ❌ (usually causes infinite loops)

4. API Call Pattern

Standard pattern for fetching crypto data:

```jsx
```

```
useEffect(() => {
 fetch('https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=usd')
  .then(response => response.json())          // Convert to JSON
  .then(data => {
   setBitcoinPrice(data.bitcoin.usd)        // Update state
   setLoading(false)                        // Stop loading
  })
  .catch(error => {
   console.error('Error:', error)           // Handle errors
   setLoading(false)
  })
}, []) // Run once when component loads
```

5. Conditional Rendering

Show different content based on state:

```jsx
{loading ? (
  <p>Loading Bitcoin price...</p>
) : (
  <p>Bitcoin Price: ${bitcoinPrice?.toLocaleString()}</p>
)}
```

Breakdown:

- `condition ? valueIfTrue : valueIfFalse` (ternary operator)
- `bitcoinPrice?.toLocaleString()` = optional chaining (safe even if bitcoinPrice is null)
- `.toLocaleString()` = formats numbers with commas (50000 → "50,000")

6. Component Structure

Our basic crypto component pattern:

```jsx
```

```
import { useState, useEffect } from 'react'

function App() {
  // 1. Declare state variables
  const [data, setData] = useState(null)
  const [loading, setLoading] = useState(true)

  // 2. Fetch data when component loads
  useEffect(() => {
    // API call here
  }, [])

  // 3. Return JSX with conditional rendering
  return (
    <div>
      {loading ? <p>Loading...</p> : <p>Data: {data}</p>}
    </div>
  )
}
```

Key Takeaways for Crypto Development

1. **State Management**: Use `useState` for any data that changes (prices, loading states, user selections)
2. **Data Fetching**: Use `useEffect` with empty `[]` to fetch initial data
3. **User Experience**: Always show loading states while fetching data
4. **Number Formatting**: Use `.toLocaleString()` to make crypto prices readable
5. **Error Handling**: Always include `.catch()` for API calls

What's Next?

- Add multiple coins to our dashboard
- Create reusable components for crypto cards
- Add real-time price updates
- Build more complex layouts for market data