11. Custom Hooks in React | Currency Project

📘 Index of Key Topics

❓ Questions & In-Depth Answers

1. What are Custom Hooks in React?

Q: What are custom hooks, and why are they useful?

A: Custom hooks are JavaScript functions that start with “use” and may call other hooks. They allow you to extract component logic into reusable functions, promoting code reuse and separation of concerns.

Analogy: Think of custom hooks as recipes. Instead of writing the same instructions repeatedly, you create a recipe (custom hook) that you can follow whenever needed.

---

2. How do you set up the Currency Converter project?

Q: What steps are involved in setting up the Currency Converter project?

A: To set up the Currency Converter project:

1. **Create a New React App:** Use `create-react-app` to initialize a new React project.

```
npx create-react-app currency-converter
cd currency-converter
```

2. **Install Dependencies:** Install necessary packages like `axios` for making HTTP requests.

```
npm install axios
```

3. **Create Components and Hooks:** Develop the necessary components and custom hooks to handle the currency conversion logic.

---

3. How do you create the `useCurrency` custom hook?

Q: Can you provide an example of the `useCurrency` custom hook?

A: Certainly! Here's an example of the `useCurrency` custom hook:

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

const useCurrency = () => {
  const [rates, setRates] = useState({});
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchRates = async () => {
      try {
        const response = await axios.get('https://api.exchangerate-api.com/v
4/latest/USD');
        setRates(response.data.rates);
        setLoading(false);
      } catch (error) {
        console.error('Error fetching exchange rates:', error);
        setLoading(false);
      }
    };

    fetchRates();
  }, []);
```

```
  return { rates, loading };
};

export default useCurrency;
```

Explanation:

- `useState`: Manages the state for exchange rates and loading status.
- `useEffect`: Fetches the exchange rates from an API when the component mounts.
- `axios`: Makes HTTP requests to retrieve the exchange rates.

---

4. How do you implement the Currency Converter component?

Q: Can you provide an example of the Currency Converter component?

A: Certainly! Here's an example of the Currency Converter component:

```
import React, { useState } from 'react';
import useCurrency from './useCurrency';

const CurrencyConverter = () => {
  const { rates, loading } = useCurrency();
  const [amount, setAmount] = useState(1);
  const [fromCurrency, setFromCurrency] = useState('USD');
  const [toCurrency, setToCurrency] = useState('INR');
  const [convertedAmount, setConvertedAmount] = useState(null);

  const handleConversion = () => {
    if (rates[fromCurrency] && rates[toCurrency]) {
      const conversionRate = rates[toCurrency] / rates[fromCurrency];
      setConvertedAmount(amount * conversionRate);
    }
  };

  return (
    <div>
      <h1>Currency Converter</h1>
```

```
      {loading ? (
        <p>Loading...</p>
      ) : (
        <>
          <inputtype="number"
            value={amount}
            onChange={(e) => setAmount(e.target.value)}
          />
          <select value={fromCurrency} onChange={(e) => setFromCurrency(e.
target.value)}>
            {Object.keys(rates).map((currency) => (
              <option key={currency} value={currency}>
                {currency}
              </option>
            ))}
          </select>
          <span> to </span>
          <select value={toCurrency} onChange={(e) => setToCurrency(e.targe
t.value)}>
            {Object.keys(rates).map((currency) => (
              <option key={currency} value={currency}>
                {currency}
              </option>
            ))}
          </select>
          <button onClick={handleConversion}>Convert</button>
          {convertedAmount !== null && (
            <p>
              {amount} {fromCurrency} = {convertedAmount.toFixed(2)} {toCurr
ency}
            </p>
          )}
        </>
      )}
    </div>
  );
};
```

```
export default CurrencyConverter;
```

Explanation:

- `useCurrency`: Custom hook that provides exchange rates and loading status.
- **State Management**: Manages states for amount, selected currencies, and converted amount.
- **Conversion Logic**: Calculates the converted amount based on selected currencies and amount.

---

5. How can you enhance the user interface?

Q: What features can be added to improve the Currency Converter UI?

A: To enhance the user interface:

- **Add Error Handling:** Display error messages if the API request fails.
- **Implement Loading Indicators:** Show a loading spinner while fetching data.
- **Style the Components:** Use CSS or a CSS framework like Tailwind CSS to style the components.

Example:

```
{loading && <div className="spinner">Loading...</div>}
{error && <div className="error">{error}</div>}
```

Explanation:

- `loading`: Displays a loading spinner while data is being fetched.
- `error`: Displays an error message if the API request fails.

---

6. What are best practices for creating custom hooks?

Q: How can you create effective custom hooks?

A: Best practices for creating custom hooks include:

- **Encapsulation:** Keep the hook focused on a single responsibility.
- **Reusability:** Design the hook to be reusable across different components.

- **Separation of Concerns:** Separate logic related to side effects, state management, and DOM interactions.

Analogy: Creating custom hooks is like building modular tools in a toolbox. Each tool (hook) serves a specific purpose and can be used independently or together to accomplish tasks efficiently.

---

7. Where can I find more resources on React and custom hooks?

Q: Where can I learn more about React and custom hooks?

A: For a comprehensive understanding, consider exploring the following resources:

- Chai Aur React Series on GitHub: Offers source code and additional materials.
- Chai Aur React YouTube Playlist: Features video tutorials covering various React topics.

🎯 Learning Path Summary

1. **Understand Custom Hooks:** Learn the purpose and benefits of custom hooks in React.
2. **Build Projects:** Apply your knowledge by building projects that utilize custom hooks.
3. **Enhance User Experience:** Add features and optimizations to improve the application's usability and performance.
4. **Continue Learning:** Explore advanced topics and patterns in React to deepen your understanding.