34. Handle APIs like a pro in Reactjs | Custom react query | Axios | Race condition

📚 Index

❓ Questions & Answers

1. What is the importance of handling APIs in React applications?

Answer:

Handling APIs in React is crucial for fetching and managing data from external sources. It allows applications to display dynamic content, interact with databases, and provide real-time updates to users.

Analogy:

Think of an API as a waiter in a restaurant. You (the client) place an order (request), and the waiter (API) brings back the food (response) from the kitchen (server).

---

2. How do you set up Axios for making API requests in React?

Answer:

To set up Axios:

1. **Install Axios:**

   ```
   npm install axios
   ```

2. **Create an Axios instance:**

   ```
   import axios from 'axios';

   const axiosInstance = axios.create({
     baseURL: 'https://api.example.com',
     headers: {
       'Content-Type': 'application/json',
     },
   });

   export default axiosInstance;
   ```

3. **Use Axios in components:**

   ```
   import axiosInstance from './axiosInstance';

   axiosInstance.get('/data')
     .then(response => console.log(response.data))
     .catch(error => console.error('Error fetching data:', error));
   ```

Example:

This setup allows for centralized configuration and easier management of API requests.

---

3. How do you create a custom React query hook for data fetching?

Answer:

To create a custom hook:

1. **Define the hook:**

```
import { useState, useEffect } from 'react';
import axiosInstance from './axiosInstance';

const useFetch = (url) => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axiosInstance.get(url)
      .then(response => {
        setData(response.data);
        setLoading(false);
      })
      .catch(err => {
        setError(err);
        setLoading(false);
      });
  }, [url]);

  return { data, loading, error };
};

export default useFetch;
```

2. **Use the hook in a component:**

```
import useFetch from './useFetch';

const MyComponent = () => {
  const { data, loading, error } = useFetch('/data');

  if (loading) return <div>Loading...</div>;
  if (error) return <div>Error: {error.message}</div>;

  return <div>{JSON.stringify(data)}</div>;
};
```

Analogy:

Creating a custom hook is like setting up a reusable recipe. Instead of cooking the same dish repeatedly, you follow the recipe (hook) each time to get consistent results.

---

4. How do you manage race conditions in API calls?

Answer:

Race conditions occur when multiple asynchronous operations complete in an unpredictable order, leading to inconsistent states. To manage them:

1. **Cancel previous requests:**

   Use `AbortController` to cancel ongoing requests when a new one is initiated.

   ```
   const controller = new AbortController();
   const signal = controller.signal;

   axiosInstance.get('/data', { signal })
    .then(response => console.log(response.data))
    .catch(error => {
     if (error.name !== 'AbortError') {
      console.error('Error fetching data:', error);
     }
    });

   // To cancel the request
   controller.abort();
   ```

2. **Track request status:**

   Maintain a flag to track whether a request is in progress and prevent unnecessary state updates.

   ```
   let isRequestInProgress = false;

   const fetchData = () => {
    if (isRequestInProgress) return;

    isRequestInProgress = true;
   ```

```
  axiosInstance.get('/data')
    .then(response => {
      // Handle response
    })
    .catch(error => {
      // Handle error
    })
    .finally(() => {
      isRequestInProgress = false;
    });
};
```

Analogy:

Managing race conditions is like ensuring that multiple chefs in a kitchen don't interfere with each other's tasks. Clear communication and coordination prevent overlapping efforts.

---

5. What are some best practices for handling APIs in React?

Answer:

Best practices include:

- **Centralize API configuration:** Use a single Axios instance to manage base URLs and headers.
- **Handle loading and error states:** Provide feedback to users during data fetching.
- **Use custom hooks:** Encapsulate data fetching logic for reusability.
- **Implement caching:** Store fetched data to avoid redundant requests.
- **Secure API calls:** Use authentication tokens and HTTPS to protect data.

Example:

By following these practices, applications become more maintainable, efficient, and secure.

🧠 Additional Insights

- **Handling Pagination:** For APIs that return large datasets, implement pagination to load data in chunks, improving performance and user experience.
- **Error Boundaries:** Use React's error boundaries to catch and handle errors in components, preventing crashes.
- **Optimistic UI Updates:** Implement optimistic updates to provide immediate feedback to users while awaiting server responses.

🔗 Useful Resources

- Axios Documentation
- React Custom Hooks Guide
- Managing Side Effects with useEffect