Day 4 - Advanced API Integration and Testing for Food Truck CMS

Objective:

Optimize API integrations, implement advanced testing techniques, and ensure the Food Truck CMS performs seamlessly. Focus on enhancing the dynamic capabilities of the Next.js frontend and refining Sanity CMS integrations.

Key Learning Outcomes:

1. **Master advanced API integration techniques, including caching and pagination.**
2. **Implement comprehensive testing for APIs and frontend components.**
3. **Optimize data fetching and rendering for better performance.**
4. **Refine and validate data consistency between the CMS and the frontend.**

Day 4 Scope:

1. Optimize API Calls:

- **Caching:**
  - **Use caching mechanisms like SWR or React Query to optimize API calls.**
  - **Example:**

```
import useSWR from 'swr';

const fetcher = (url) => fetch(url).then((res) => res.json());

export default function MenuList() {
  const { data, error } = useSWR('https://foodtruck-qcommerce.vercel.app/api/menu', fetcher);

  if (error) return <div>Failed to load menu</div>;
  if (!data) return <div>Loading...</div>;

  return (
    <div>
      {data.map((item) => (
        <div key={item.id}>
          <h2>{item.name}</h2>
          <p>{item.price}</p>
        </div>
      ))}
    </div>
  );
```

```
    }
    ```
- **Pagination:**
    - **Add pagination to handle large datasets.**
    - **Example:**

    ```
    const [page, setPage] = useState(1);

    useEffect(() => {
      fetch(`https://foodtruck-qcommerce.vercel.app/api/menu?page=${page}`)
        .then((res) => res.json())
        .then((data) => setMenuItems(data));
    }, [page]);

    return (
      <div>
        {menuItems.map((item) => (
          <div key={item.id}>
            <h2>{item.name}</h2>
          </div>
        ))}
        <button onClick={() => setPage(page - 1)} disabled={page === 1}>Previous</button>
        <button onClick={() => setPage(page + 1)}>Next</button>
      </div>
    );
    ```

2. Advanced Testing Techniques:

- **API Testing:**
    - **Use tools like Postman or Jest to validate API responses.**
    - **Example with Jest:**

    ```
    test('fetches menu items', async () => {
      const data = await fetchMenuItems();
      expect(data).toBeDefined();
      expect(data).toBeInstanceOf(Array);
    });
    ```
- **Component Testing:**
    - **Use tools like React Testing Library to test frontend components.**
    - **Example:**

    ```
    import { render, screen } from '@testing-library/react';
    import MenuList from './MenuList';

    test('renders menu items', async () => {
      render(<MenuList />);
      expect(await screen.findByText(/Menu Item Name/i)).toBeInTheDocument();
    });
    ```

3. Refine Data Consistency:

- Ensure API responses align with Sanity CMS schema.
- Implement validation functions to check for mismatches.
- function validateData(data) {
-   return data.every((item) => item.name && item.price);
- }
-
- const isValid = validateData(menuItems);
- if (!isValid) {
-   console.error('Data validation failed');
- }

4. Improve Frontend Performance:

- Lazy Loading:
  - Load images and components only when needed using libraries like react-lazyload.
- Code Splitting:
  - Use dynamic imports in Next.js:
  - const DynamicMenu = dynamic(() => import('./MenuList'));

Expected Output:

1. Enhanced API integrations with caching and pagination.
2. Fully tested APIs and components with error-free responses.
3. Improved frontend performance with optimized data rendering.
4. Comprehensive validation ensuring consistency across CMS and frontend.

Submission Requirements:

1. A report titled: Day 4 - Advanced API Integration Report - [Your Food Truck Name]
2. Include:
   - Advanced API integration details.
   - Testing strategies and results.
   - Optimization techniques used.
3. Screenshots of:
   - Caching and pagination in action.
   - Test results for APIs and components.
   - Enhanced frontend performance.
4. Code snippets for caching, pagination, and testing.

Best Practices:

1. Use .env for API keys:
   - Store sensitive information securely.
2. Write reusable functions:
   - Modularize API calls and validations.
3. Monitor performance:
   - Use tools like Lighthouse or React Profiler.

4. **Thoroughly document:**
    - **Provide comments and explanations for complex code.**
5. **Test extensively:**
    - **Include edge cases and unexpected inputs.**

Day 4 Checklist:

Task	Completed?
API Optimization	[ ]
Advanced Testing	[ ]
Data Consistency Validation	[ ]
Frontend Performance	[ ]
Submission Preparation	[ ]

FAQs:

1. **How can we implement caching easily? Use libraries like SWR or React Query for automatic caching and revalidation.**
2. **What tools are best for testing? Use Postman for API testing and Jest with React Testing Library for component tests.**
3. **What if the data is inconsistent? Validate data during fetching and log discrepancies for review.**
4. **Can we skip pagination? Pagination is optional but highly recommended for large datasets.**

5. **What should we submit? A detailed report with integration, testing, and optimization details, along with code snippets and screenshots.**