Document Title: Day 3 - Data Integration and API Setup - [Comforty Marketplace]

1. Overview:

This document covers the work completed on **Day 3**, where we focused on API integration, data migration, and setting up the backend for our marketplace project. The key goal was to ensure that dynamic data is handled properly, enabling smooth data flow and integration with Sanity CMS (or an alternative API). We also explored error handling and API responses to ensure the frontend can work seamlessly with the backend data.

2. Key Tasks Completed:

- **API Integration:**
  - Integrated the provided API endpoints to fetch product and category data.
  - Configured API requests to ensure that the frontend can consume and display dynamic content.
  - Set up functions to handle data fetching from external sources (Sanity CMS or APIs).
  - Ensured the data was successfully rendered in the product listing, product details, and category components.
- **Data Migration:**
  - Migrated product data and categories from external sources to **Sanity CMS** using appropriate scripts and tools.
  - Verified that data migration scripts are functional and that data is correctly stored in Sanity's backend.
- **Testing API Responses:**
  - Validated that the API returns the correct data.
  - Implemented error handling mechanisms for failed API calls or empty responses.
  - Ensured that the API responses are properly mapped to frontend components, allowing data like product names, prices, images, stock status, and more to appear correctly.
- **Data Handling:**
  - Tested and ensured that data fetched from the API or Sanity CMS matches the expected structure for product listings and categories.
  - Implemented proper data handling in Next.js components to display dynamic content.

3. Challenges Faced:

- **API Response Errors:**
  - Some API endpoints were initially not returning the correct data due to misconfigured request headers or API keys. This was resolved by double-checking the API configuration and ensuring the correct access permissions were applied.
  - Another issue was handling large product datasets. Pagination and efficient data fetching techniques were employed to prevent performance degradation.

- **Sanity CMS Integration:**
  - Configuring the Sanity CMS schema to align with the marketplace's data structure required adjustments to the dataset model. Ensuring proper migration of content was another hurdle that required thorough verification.

4. Solutions Implemented:

- **Error Handling:**
  - Introduced try-catch blocks and custom error messages for failed API requests to provide more transparency.
  - Added fallback content for situations where data may not be available.
- **API Key and Data Configuration:**
  - Ensured the API keys were configured correctly in environment variables.
  - Applied robust mapping functions to ensure data fetched from Sanity CMS is aligned with frontend components.
- **Performance Optimization:**
  - Implemented pagination for large datasets to enhance performance on pages with many products.
  - Applied lazy loading for images to ensure smooth rendering of product data without impacting page speed.

5. Best Practices Followed:

- **Modular and Reusable Code:**
  - Created modular functions to handle API requests and data fetching. This ensures that the code can be reused across different components like product listings, product details, and category filters.
- **State Management:**
  - Used React's `useState` and `useEffect` hooks to manage dynamic data across components. This ensures data consistency and enables components to re-render when new data is fetched.
- **Responsiveness:**
  - Ensured that the data display components (e.g., product listings and product detail pages) were styled and structured to be responsive, following mobile-first design principles.

6. Key Components Implemented:

- **Product Listing Page:**
  - Displayed a grid of products fetched from the API with product names, prices, and stock statuses. Used `map()` to render dynamic product cards.
- **Product Detail Page:**
  - Implemented dynamic routing using Next.js' `getServerSideProps` for individual product details pages. Each product's details (e.g., description, price, available sizes) are fetched based on its unique ID.
- **Category Filtering:**

    - Built a category filter component to filter products based on categories dynamically fetched from the Sanity CMS API.
- **Search Bar:**
    - Created a search bar that dynamically filters products based on product names or tags.

7. Next Steps:

- **Data Validation:**
    - Continue testing data fetched from the API for consistency, particularly for edge cases like missing or incomplete data.
- **Integrate Additional Data Points:**
    - Add additional data fields for product variants (size, color) and user-generated content (e.g., reviews).
- **Enhancements for Day 4:**
    - Move forward with building dynamic frontend components, focusing on modular, reusable designs for the marketplace. This will include components like product listing, user profile, and cart functionality.

8. Screenshots/Visual Proof:

- Attach screenshots of the following:
    1. Product listing page showing products rendered dynamically.
    2. Product detail page with correct dynamic data for each product.
    3. Category filter component working as expected.
    4. Search functionality working to filter products.

9. Code Snippets:

- Attach code snippets for the following key sections:
    1. API integration and data fetching functions.
    2. Dynamic routing for product detail pages.
    3. Category filter implementation.
    4. Code to handle API responses and errors.

10. Conclusion:

Day 3 focused on backend integration and data management to ensure that the frontend could display dynamic content sourced from APIs or Sanity CMS. Challenges around data handling and performance were addressed with solutions like pagination and lazy loading. The groundwork for building dynamic, data-driven frontend components is now set, enabling the project to progress into frontend component development in the coming days.