Day 3 - API Integration Report - Ras Healthcare Marketplace

API Integration Process

Overview

The API integration for Ras Healthcare leverages **Appwrite**, a backend-as-a-service platform, to manage data operations and streamline interactions between the frontend (built using Next.js) and backend services. The integration ensures efficient data retrieval, secure transactions, and seamless synchronization across the application.

Steps:

1. **Setting Up Appwrite:**
    - Configured Appwrite on the server to manage collections for products, orders, and customers.
    - Enabled authentication and access control for secure API usage.
2. **Defining Endpoints:**
    - Custom endpoints were defined to fetch and manage data, aligning with the data schema drafted earlier:
        - `/products` (GET): Fetch product listings.
        - `/orders` (POST): Submit customer orders.
        - `/shipment` (GET): Retrieve shipment tracking information (future integration with Trax).
3. **Frontend Integration:**
    - Connected the Next.js frontend with Appwrite using the `node-appwrite` SDK.
    - Implemented state management for seamless data flow between API calls and UI components using **@tanstack/react-query**.
4. **Testing and Validation:**
    - Verified endpoint responses using Postman and integration tests.
    - Cross-checked data integrity by populating the frontend with retrieved data and ensuring alignment with the backend schema.

Adjustments Made to Schemas

Products Schema:

- **Original:** Included fields for name, price, stock, description, and image.
- **Adjusted:**
    - Added `category` field for product classification.
    - Enhanced `image` field to support multiple images for product galleries.

Orders Schema:

- **Original:** Captured customer details, product IDs, and total amount.
- **Adjusted:**
    - Added `orderStatus` field for tracking order progress.
    - Introduced `paymentMethod` to specify payment options.

Customers Schema:

- **Original:** Contained customer name, email, and address.
- **Adjusted:**
    - Added `phoneNumber` for contact purposes.
    - Included `orderHistory` as a reference to previous orders.

Migration Steps and Tools Used

Tools Used:

- **Appwrite Console:** For managing collections and real-time database updates.
- **Custom Migration Script:** Written in Node.js to migrate legacy data into Appwrite collections.

Migration Steps:

1. **Data Extraction:**
    - Exported existing data from spreadsheets and JSON files.
    - Normalized data formats to align with the Appwrite schema.
2. **Script Development:**
    - Developed a Node.js script utilizing `node-appwrite` to push data into the Appwrite database.
    - Implemented error handling to ensure data consistency during migration.
3. **Execution:**
    - Ran the migration script in batches to avoid performance bottlenecks.
    - Verified successful migration by querying the database and comparing results with the source data.

4. **Post-Migration Validation:**

    - Conducted API calls to confirm data availability and accuracy.
    - Ensured that the frontend displayed migrated data correctly.

Screenshots (To Be Added)

- **API Calls:** Captures of successful requests and responses from Postman.
- **Frontend Display:** Screenshots of populated product pages, order summaries, and user dashboards.
- **Appwrite Collections:** Visuals of populated collections within the Appwrite Console.

Code Snippets

API Integration Example

```
import { Client, Databases } from "appwrite";

const client = new Client();
client.setEndpoint("https://your-appwrite-server.com/v1")
    .setProject("project_id");

const databases = new Databases(client);

export async function fetchProducts() {
  try {
    const response = await databases.listDocuments("database_id", "products_collection_id");
    return response.documents;
  } catch (error) {
    console.error("Error fetching products:", error);
  }
}
```

Data Migration Script Example

```
import { Client, Databases } from "appwrite";

const client = new Client();
client.setEndpoint("https://your-appwrite-server.com/v1")
    .setProject("project_id");

const databases = new Databases(client);

async function migrateData(data) {
  for (const item of data) {
```

```
    try {
      await databases.createDocument("database_id", "collection_id", item.id, item);
      console.log(`Migrated: ${item.id}`);
    } catch (error) {
      console.error(`Error migrating ${item.id}:`, error);
    }
  }
}

const data = [
  { id: "1", name: "Multivitamin", price: 1500, stock: 50 },
  { id: "2", name: "Vitamin D", price: 1200, stock: 100 }
];

migrateData(data);
```

Conclusion

The API integration and data migration processes were successfully implemented using Appwrite. The adjustments to schemas ensured alignment with evolving business requirements, while the custom scripts facilitated a smooth data migration. The integration lays a strong foundation for the scalable and efficient operation of Ras Healthcare Marketplace.