DAY 3 - API INTEGRATION AND DATA MIGRATION - [General E-Commerce]

Name: Mahnoor Ansari

API Integration:

It connects different software systems, allowing them to share data and work together (e.g., your app fetching product info from a server).

Data Migration:

It means moving data from one system or storage to another (e.g., transferring user data to a new database).

1. The Process of API Integration

1. **Understand the API Documentation**
   Read the API documentation to understand:
   - Available endpoints (e.g., `/products`, `/cart`).
   - HTTP methods (GET, POST, PUT, DELETE).
   - Authentication requirements (e.g., API key or tokens).

   The purpose of API integration:

   GET: To **fetch** data (e.g., a list of products or users).

   POST: To **send** new data (e.g., adding a product to the cart).

   PUT: To **update** existing data (e.g., changing the quantity in the cart).

   DELET: To **delete** data (e.g., removing a product from the cart).

Set Up the API Key or Authentication:

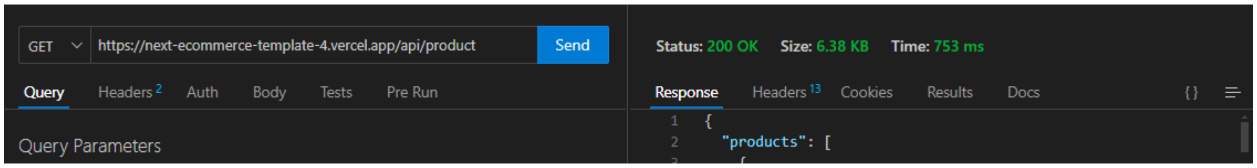
- Obtain the required API key, token, or credentials.
- Securely store them (e.g., in environment variables).

Install Necessary Tools or Libraries

- Use libraries like `axios` or `fetch` in JavaScript/TypeScript for making API calls.
- Install them if needed (e.g., `npm install axios`).

Test API Endpoints

- Use tools like **Postman** or **cURL** to test endpoints.
- Verify the responses and understand the data structure.



```
https://next-ecommerce-template-4.vercel.app/api/product
```

Status: 200 OK Size: 6.38 KB Time: 753 ms

```
{
  "products": [
```

Handle Responses and Errors

- Process the data received from the API.
- Handle errors gracefully (e.g., invalid API key or server issues).

Test the Integration

- Ensure all API calls are working correctly.
- Check for edge cases and handle failures.

---

2. Adjustments made to schemas:

Schemas define the structure and organization of your database or content in a project.

For Example:

```
export default{

 name: "product", title: "Product", type: "document", fields: [

 { name: "name", type: "string", title: "Product Name" },

 { name: "price", type: "number", title: "Price" },

{ name: "rating", type: "number", title: "Product Rating", validation: (Rule) =>
Rule.min(0).max(5) }, ], };
```

- You add new information to the system, like a `description` for each product.
- You change the type of information, for example, making a price field a number instead of text.
- You delete unnecessary information, like removing an old `discount` field that's no longer needed.
- You connect different parts of your system, like linking `users` to their `orders` to keep everything organized.

3. Migration steps and tools used:

- The script was designed to connect to the source database, extract data, and then push it to the target system
- The script started by querying the old system to fetch all necessary data, like user details, product information, etc.
- Data types (e.g., converting text to numbers) were changed, missing values were cleaned up, or the data was restructured to match the new system's schema.
- Inserting the data into the new database or storage, ensuring that it matched the expected structure.

- Tested the migration by running queries to make sure everything was transferred correctly.
- ensure that the data was successfully transferred and was in the correct format.