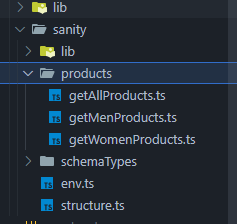
**Day 3 -**  **API Integration Report - VaultSneaks**

**Objective**

For integrating existing APIs and migrating data to Sanity CMS to ensure the favorable running of the new dynamic shoe marketplace. This consists of using, collaborating with Typegen for obtaining data, making a helper function in **sanity/products**, and integrating products on the frontend.



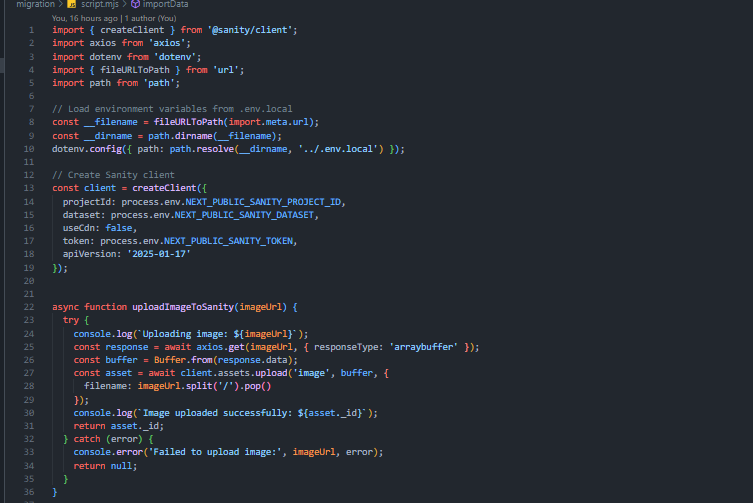
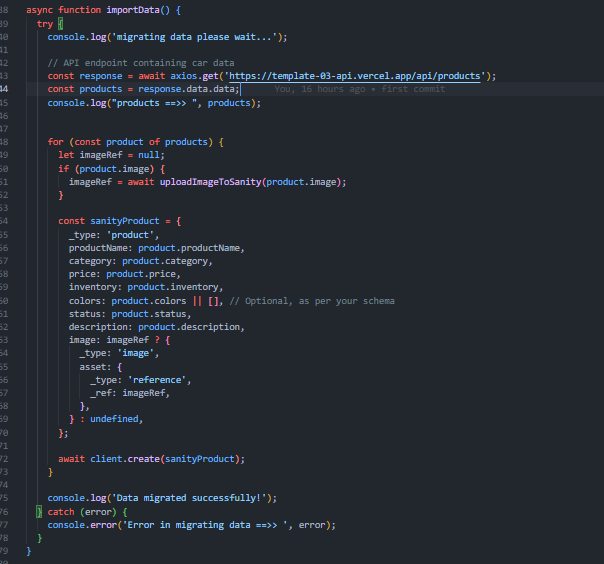
**Key Activities**

1. **API Integration**

* Integrated API to fetch product data
* Utilized Typegen to streamline the process of fetching data from Sanity CMS.

1. **Data Migration**

* Migrated product data into Sanity CMS using a migration script.
* Validated data integrity and alignment with the defined schema.

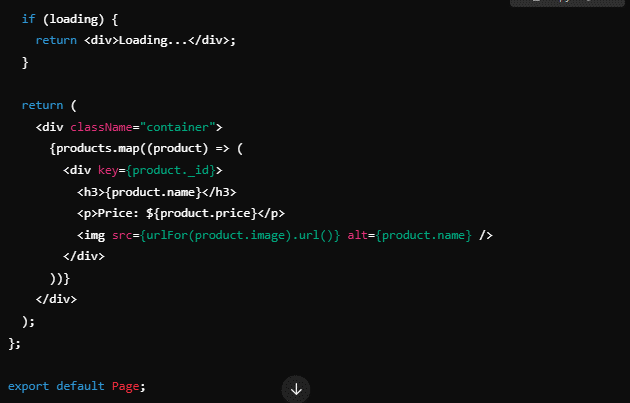
1. **Helper Function Creation**

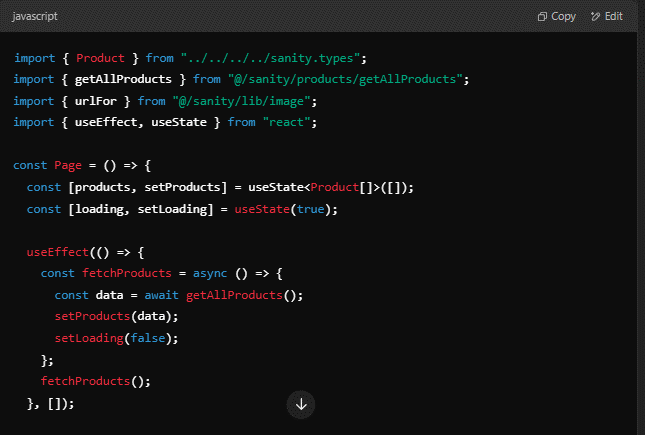
* Developed a helper function in **sanity/products** to efficiently fetch products from Sanity CMS.
* Ensured the function could be reused across different parts of the application to maintain consistency.



1. **Frontend Implementation**

* Displayed migrated products on the frontend using Next.js.
* Ensured that data fetched from Sanity CMS was correctly rendered in the user interface.





**Steps Taken**

1. **Understanding the Provided API**

* Reviewed API documentation to comprehend available endpoints and data structures.
* Key API Endpoint Used:
  + **/products** (GET): Fetched product listings for migration into Sanity CMS.

1. **Data Migration Process**

* Used the migration script to transfer data from the API to Sanity CMS.
* **Steps**:
  + Fetched data from the API.
  + Transformed data to match Sanity schema requirements.
  + Imported data into Sanity CMS.

1. **Helper Function in sanity/products**

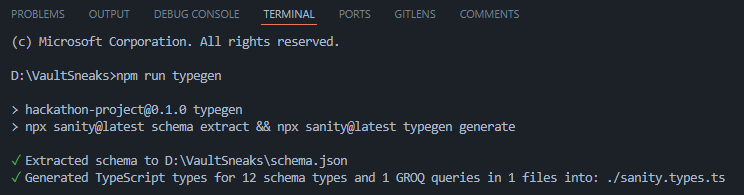
* Created a helper function for fetching product data from Sanity CMS, ensuring that all requests for product data follow a standardized approach.
* **Function**: getAllProducts():
  + **Purpose**: To fetch and return product data from Sanity CMS.
  + **Usage**: Utilized across various components to ensure consistent data fetching and minimize redundant code.

1. **API Integration in Next.js**

* Created utility functions to fetch and handle API data.
* Rendered fetched data in Next.js components to display product listings on the frontend.
* Tested integration using tools like Postman, thunderclient and browser developer tools.

1. **Understanding Typegen**

* **Typegen**: A tool that automatically generates TypeScript types based on your GraphQL schema or API data structure. It ensures type safety by providing autocompletion and type-checking in your IDE.



* **Benefits**:
  + Reduces the risk of runtime errors by catching type-related issues during development
  + Enhances developer productivity through autocompletion and better documentation.

**Best Practices Followed**

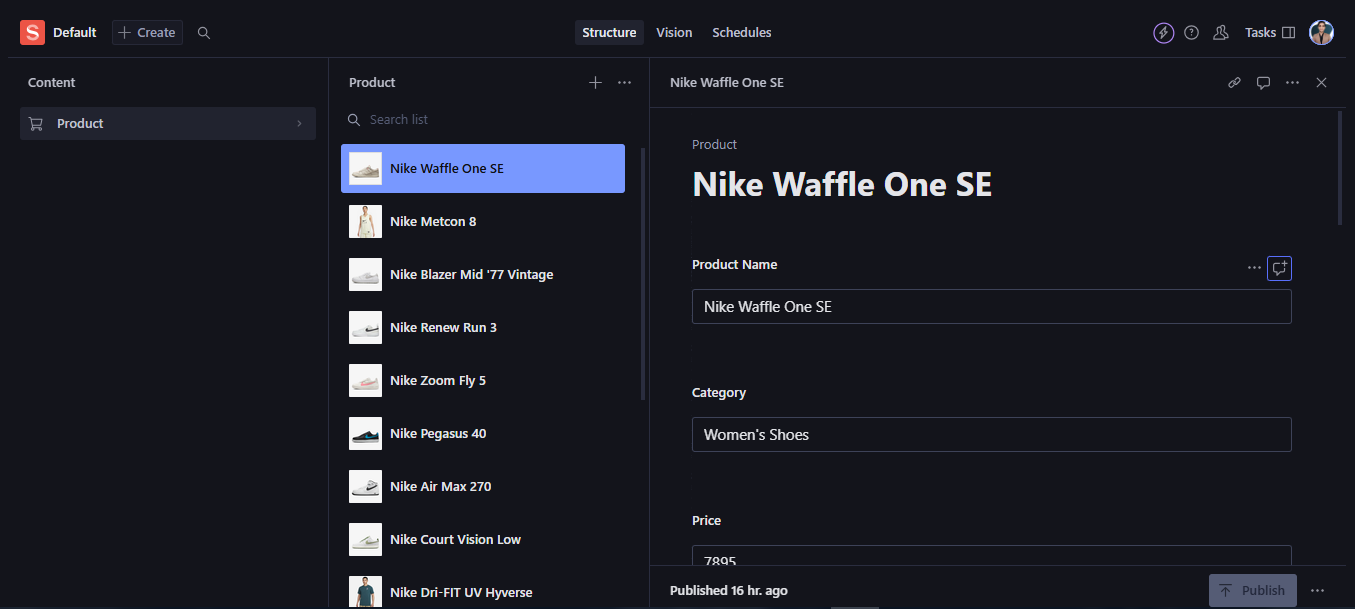
* Used .env files for secure API key management.
* Modularized code for better reusability and maintainability.
* Validated data during migration to ensure schema alignment.

**Error Handling**

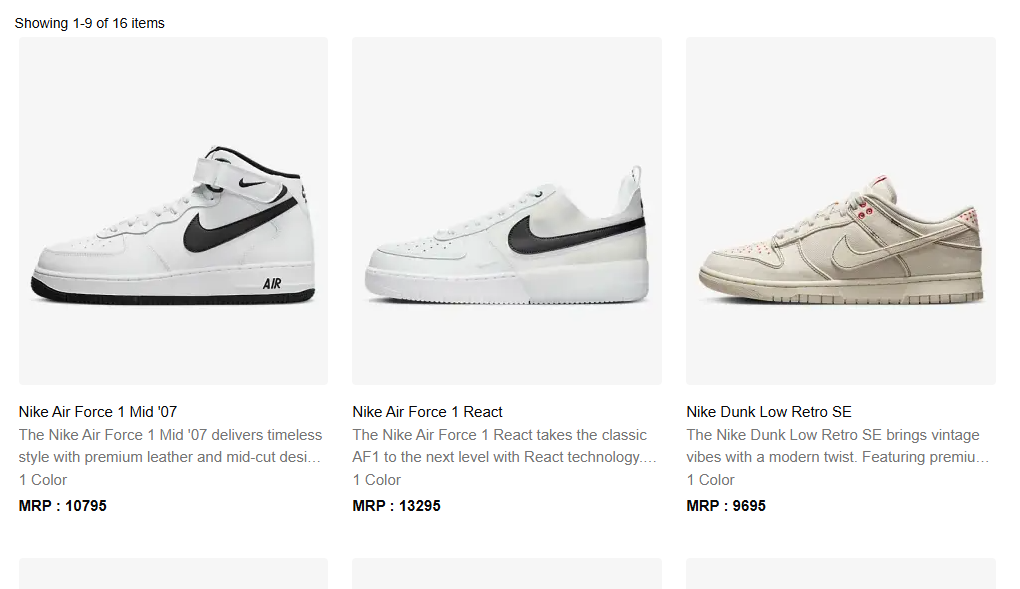
* Implemented error logging for debugging.
* Displayed user-friendly error messages on the frontend.
* Used fallback data to enhance the user experience in case of API failure.

**Expected Output**

* **Sanity CMS**: Populated with imported product data from API.



* **Frontend**: Functional product listings displayed using data fetched from Sanity CMS.



**Conclusion**

This documentation encapsulates the efforts on Day 3, highlighting the integration of API, successful data migration into Sanity CMS, the creation of a reusable helper function, and frontend display of products. The use of Typegen ensures type safety and enhances the development process, contributing to a robust and scalable shoe marketplace.