

## Day 3 - API Integration Report - [Avion]

### 1. Data displayed on the frontend.

### All products

#### Product type

☐ Furniture

☐ Homeware

☐ Sofas

☐ Light fittings

☐ Accessories

#### Price

☐ 0 - 100

☐ 101 - 250

☐ 250+


#### Designer

☐ Robert Smith


☐ Liam Gallagher

☐ Biggie Smalls


☐ Thom Yorke



**Modern Rubber Fish**  
37.00




**Bespoke Rubber Hat**  
132.00



**Generic Granite Fish**  
630.00

[All Products](#) [Plant pots](#) [Ceramics](#) [Tables](#) [Chairs](#) [Crocery](#) [Tableware](#) [Cutlery](#)



**Luxurious Wooden Mouse**  
**\$ 223.00**

Description:

Ergonomic executive chair upholstered in bonded black leather and PVC padded seat and back for all-day comfort and support

- Premium material
- Handmade upholstery
- Quality timeless classic

Colors

Red

Blue

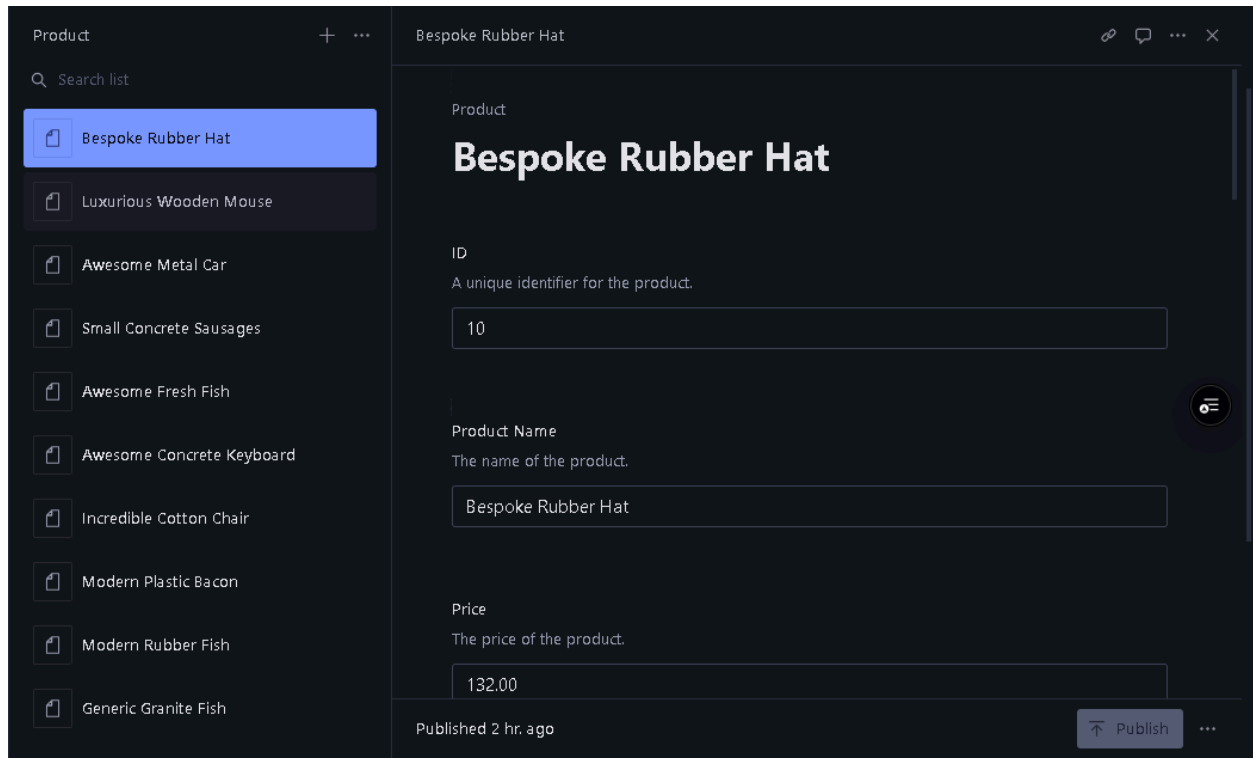
Black

Quantity: 

- 1 +

ADD TO CART

### 2. Populated Sanity CMS fields.



## Successful API calls.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

> next dev --turbo

▲ Next.js 15.0.4 (Turbopack)
- Local:      http://localhost:3000
- Environments: .env.local

✓ Starting...
✓ Ready in 2.7s
○ Compiling / ...
✓ Compiled / in 9s
GET / 200 in 10296ms
○ Compiling /favicon.ico ...
✓ Compiled /favicon.ico in 627ms
GET /favicon.ico?favicon.45db1c09.ico 200 in 898ms
✓ Compiled /products in 346ms
GET /products 200 in 453ms
○ Compiling /api/products ...
✓ Compiled /api/products in 1193ms
GET /api/products 200 in 1891ms
GET /api/products 200 in 357ms
□
```

## Code Scripts Screen Shots:

```
app > api > products > TS route.ts > ➔ Product
1  import { client } from "@sanity/lib/client";
2  import axios from "axios";
3  import { NextResponse } from "next/server";
4  import { nanoid } from 'nanoid';
5
6  interface Variation {
7    color: string; // Color of the product (e.g., "Red", "Blue")
8    size: string; // Size of the product (e.g., "S", "M", "L")
9    quantity: number; // Available quantity for the specific color and size
10 }
11
12 interface Product {
13   id: string;
14   name: string;
15   price: number;
16   discountPercentage: number;
17   image: string | string[];
18   rating: string;
19   tags: string[];
20   description: string;
21   variations: Variation[]; // Variations of the product with different colors, sizes, and quantities
22 }
23
24 const MOCK_API_URL = `${process.env.NEXT MOCK_API}`;
25
```

app > api > products > TS route.ts > uploadImagesToSanity > assets > urls.map() callback

```
25
26
27 async function uploadImagesToSanity(image: string | string[]) {
28   if (!image) {
29     console.warn("No image URLs provided.");
30     return [];
31   }
32
33   const urls = Array.isArray(image) ? image : [image];
34   console.log("Processing the following URLs:", urls);
35
36   const assets = await Promise.all(
37     urls.map(async (url) => {
38       try {
39         console.log("Fetching image URL:", url);
40         const response = await axios.get(url, { responseType: "arraybuffer" });
41         console.log("Fetched image response status:", response.status);
42
43         const buffer = Buffer.from(response.data, "binary");
44         const asset = await client.assets.upload("image", buffer, {
45           filename: `product_image_${Date.now()}.jpg`,
46         });
47         console.log("Successfully uploaded asset:", asset);
48
49         return {
50           _type: "image",
51           _key: nanoid(),
52           asset: { _type: "reference", _ref: asset._id },
53         };
54       } catch (error) {
55         console.error(`Error uploading image from ${url}:`, error);
56         return null;
57       }
58     })
59   );
60
61   const filteredAssets = assets.filter(Boolean);
62   console.log("Final filtered assets:", filteredAssets);
63   return filteredAssets;
64 }
65
```

```
app > api > products > TS route.ts > uploadImagesToSanity > assets > urls.map() callback
65
66 export async function POST() {
67   try {
68     const { data: products } = await axios.get<Product[]>(MOCK_API_URL);
69     console.log("Fetched products:", products);
70
71     if (!Array.isArray(products) || products.length === 0) {
72       return NextResponse.json(
73         { success: false, error: "Invalid or empty product data" },
74         { status: 400 }
75       );
76     }
77
78     async function delay(ms: number) {
79       return new Promise((resolve) => setTimeout(resolve, ms));
80     }
81
82     const sanityOperations = [];
83
84     // First, delete all existing products
85     // await client.delete({query: '*[_type == "product"]'});
86
87     for (const product of products) {
88       await delay(1000);
89       console.log("Processing product:", product);
90       console.log("Image URL:", product.image);
91
92       const operation = (async () => {
93         const images = await uploadImagesToSanity(product.image);
94
95         const variations = product.variations.map((variation) => ({
96           color: variation.color,
97           size: variation.size,
98           quantity: variation.quantity,
99         }));
100
```

```

app > api > products > TS route.ts > uploadImagesToSanity > assets > urls.map() callback
66   export async function POST() {
92     const operation = (async () => {
100
101       const sanityProduct = {
102         _type: "product",
103         id: `${product.id}`,
104         name: product.name,
105         price: product.price,
106         priceWithoutDiscount: product.price,
107         discountPercentage: product.discountPercentage,
108         description: product.description,
109         images,
110         ratings: product.rating,
111         tags: product.tags,
112         variations,
113       };
114
115       return client.createOrReplace({
116         _id: `product-${product.id}`,
117         ...sanityProduct,
118       });
119     })();
120     sanityOperations.push(operation);
121   }
122
123   const results = await Promise.all(sanityOperations);
124

```

```

124
125   console.log("Products synced successfully!");
126   return NextResponse.json(
127     {
128       success: true,
129       message: "Products synced successfully!",
130       data: results,
131     },
132     { status: 200 }
133   );
134 } catch (error) {
135   console.error("Error syncing products:", error);
136   return NextResponse.json(
137     {
138       success: false,
139       message: "Error syncing products",
140       error: error instanceof Error ? error.message : String(error),
141     },
142     { status: 500 }
143   );
144 }
145 }
146

```

```

147
148 export async function GET() {
149   try {
150     const query = `*[_type == "product"]`; // Fetch all products from Sanity
151     const products = await client.fetch(query);
152
153     return NextResponse.json(
154       { success: true, data: products },
155       { status: 200 }
156     );
157   } catch (error) {
158     return NextResponse.json(
159       { success: false, message: "Error fetching products", error: error instanceof Error ? error.message : String(error) },
160       { status: 500 }
161     );
162   }
163 }
164

```

### Fetching Products on Frontend:

```

42   const [products, setProducts] = useState<Product[]>([]);
43
44   useEffect(() => {
45     fetch('/api/products')
46       .then((res) => res.json())
47       .then((data) => setProducts(data.data))
48       .catch((error) => {
49         console.error("Error fetching featured products:", error);
50       });
51   }, []);
52

```

## 2. Schema Adjustments:

- **Product Schema:** The Sanity schema for the **product** document includes fields like **id**, **name**, **price**, **discountPercentage**, **images**, **ratings**, **tags**, and **variations**.
- **Adjusting for Variations:** The **variations** field is a nested object that contains **color**, **size**, and **quantity**. This schema was adjusted to allow multiple variations for each product.

## 3. Data Migration Steps:

The following steps were followed for migrating data into Sanity CMS:

- **API Call:** Fetch product data using the **axios** library from the provided API.

- **Image Upload:** Images for each product were fetched using their URL and uploaded to Sanity via the `uploadImagesToSanity` function.
- **Sanity Insertion:** Each product was either created or replaced in Sanity using `client.createOrReplace()`.

### **Day 3 Checklist:**

#### **Self-Validation Checklist:**

**API Understanding:** ✓

**Schema Validation:** ✓

**Data Migration:** ✓

**API Integration in Next.js:** ✓

**Submission Preparation:** ✓



