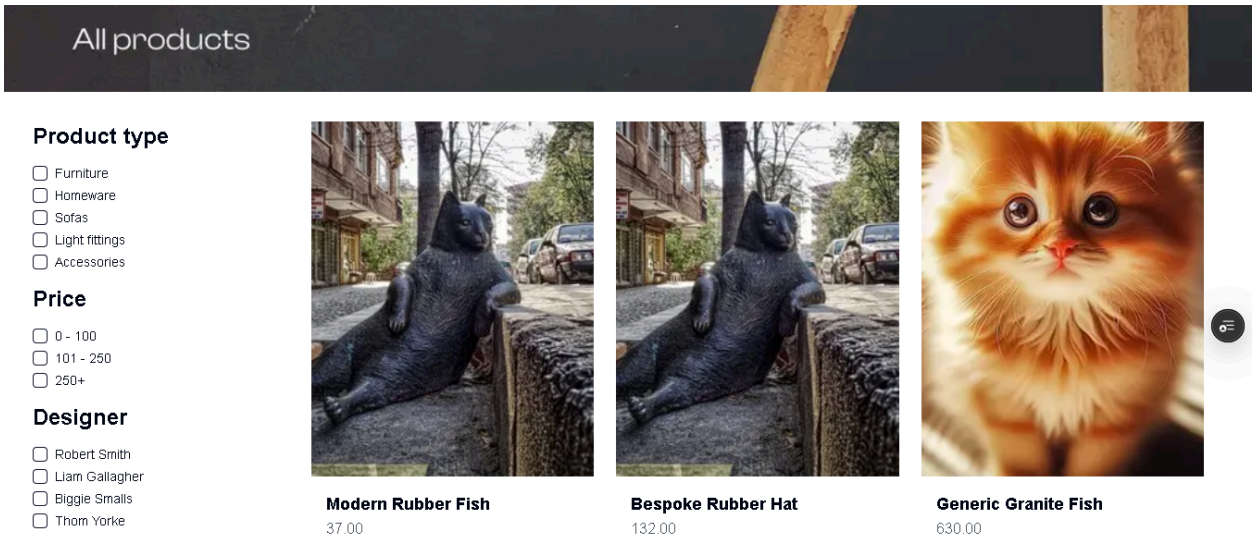
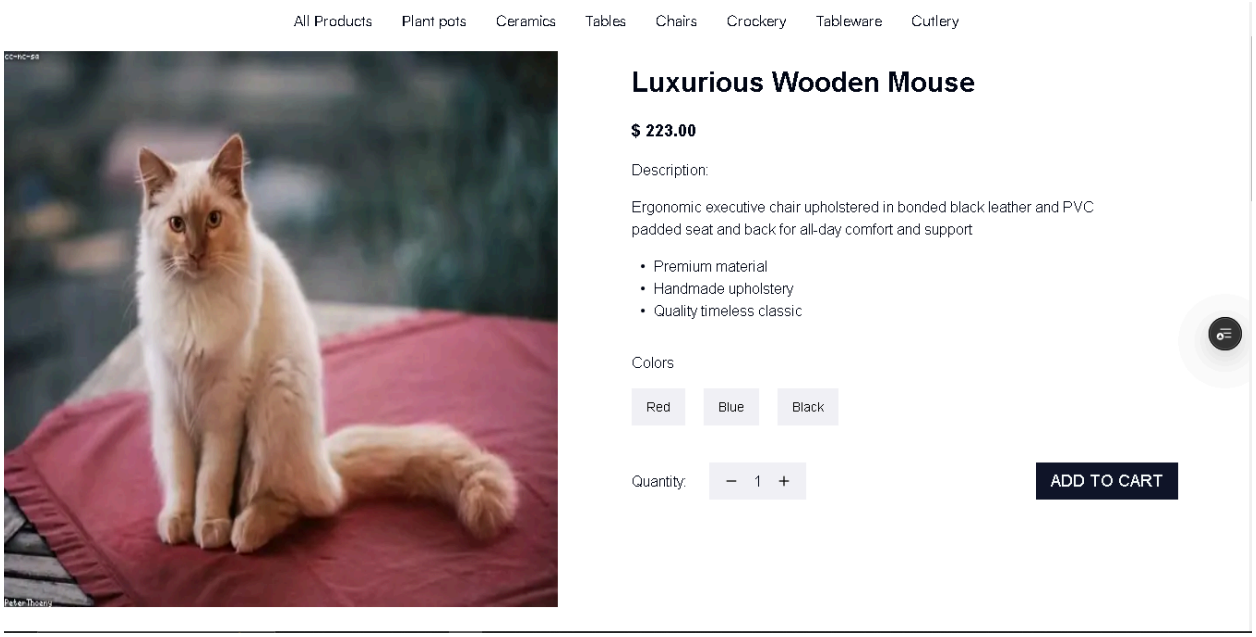
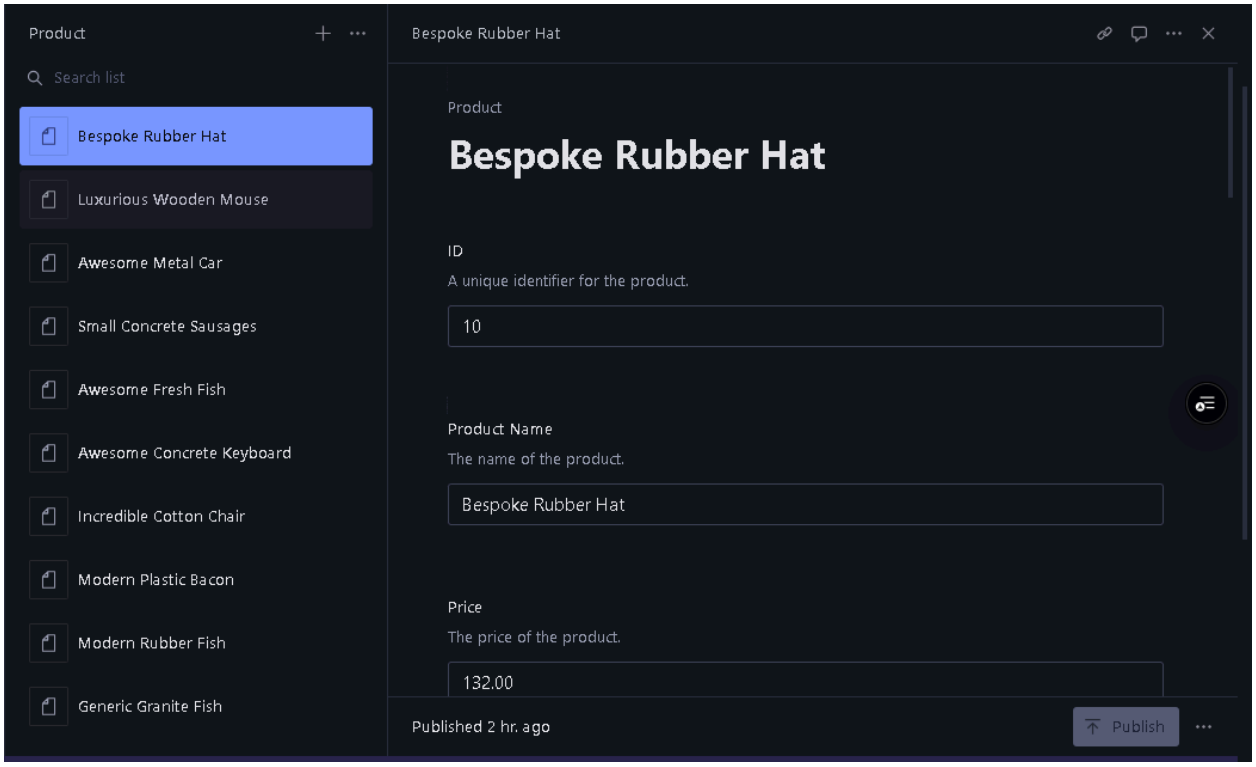
Day 3 - API Integration Report - [Avion]

1. Data displayed on the frontend.

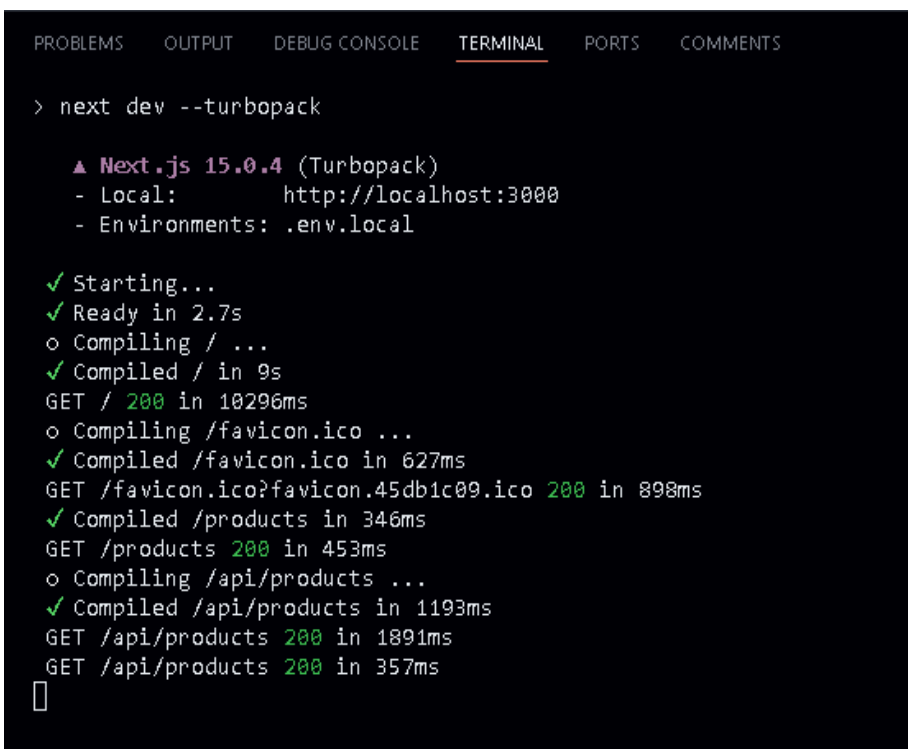




2. Populated Sanity CMS fields.



Successful API calls.



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 > 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 > 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          }));
```

app > api > products > 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
```

```
export async function GET() {
  try {
    const query = `*[_type == "product"]`; // Fetch all products from Sanity
    const products = await client.fetch(query);

    return NextResponse.json(
      { success: true, data: products },
      { status: 200 }
    );
  } catch (error) {
    return NextResponse.json(
      { success: false, message: "Error fetching products", error: error instanceof Error ? error.message : String(error) },
      { status: 500 }
    );
  }
}
```

Fetching Products on Frontend:

```
const [products, setProducts] = useState<Product[]>([]);

useEffect(() => {
  fetch('/api/products')
  .then((res) => res.json())
  .then((data) => setProducts(data.data))
  .catch((error) => {
    console.error("Error fetching featured products:", error);
  });
}, []);
```

Sanity Schema:

sanity > schemaTypes > TS Products.ts > default > fields

```
import { defineType, defineField } from "sanity";

export default defineType({
  name: "product",
  title: "Product",
  type: "document",
  fields: [
    defineField({
      name: "id",
      title: "ID",
      type: "string",
      description: "A unique identifier for the product.",
    }),
    defineField({
      name: "name",
      title: "Product Name",
      type: "string",
      description: "The name of the product.",
    }),
    defineField({
      name: "price",
      title: "Price",
      type: "string",
      description: "The price of the product.",
    }),
    defineField({
      name: "images",
      title: "Product Images",
      type: "array",
      of: [{ type: "image" }],
      description: "Images of the product.",
    }),
```

```
    defineField({
      name: "ratings",
      title: "Ratings",
      type: "number",
      description: "The average ratings of the product (e.g., '5.0').",
    }),
    defineField({
      name: "tags",
      title: "Tags",
      type: "array",
      of: [{ type: "string" }],
      description: "Tags to categorize the product (e.g., 'Best Selling').",
    }),
    defineField({
      name: "discountPercentage",
      type: "number",
      title: "Discount Percentage",
    }),
    defineField({
      name: "priceWithoutDiscount",
      type: "string",
      title: "Price Without Discount",
      description: "Original price before discount",
    }),
    defineField({
      name: "ratingCount",
      type: "number",
      title: "Rating Count",
      description: "Number of ratings",
    }),
```

```
63        defineField({
64          name: "description",
65          title: "Description",
66          type: "text",
67          description: "A detailed description of the product.",
68        }),
69        defineField({
70          name: "variations",
71          title: "Product Variations",
72          type: "array",
73          of: [
74            defineField({
75              name: "variation",
76              type: "object",
77              fields: [
78                {
79                  name: "color",
80                  title: "Color",
81                  type: "string",
82                  description: "Color of the product variation.",
83                },
84                {
85                  name: "size",
86                  title: "Size",
87                  type: "string",
88                  description: "Size of the product variation.",
89                },
```

```
89                },
90                {
91                  name: "quantity",
92                  title: "Quantity",
93                  type: "number",
94                  description: "Available quantity for this variation.",
95                },
96              ],
97            }),
98          ],
99          description: "List of variations for the product, including size, color, and quantity.",
100       }),
101     ],
102   });
103
```

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: ✔