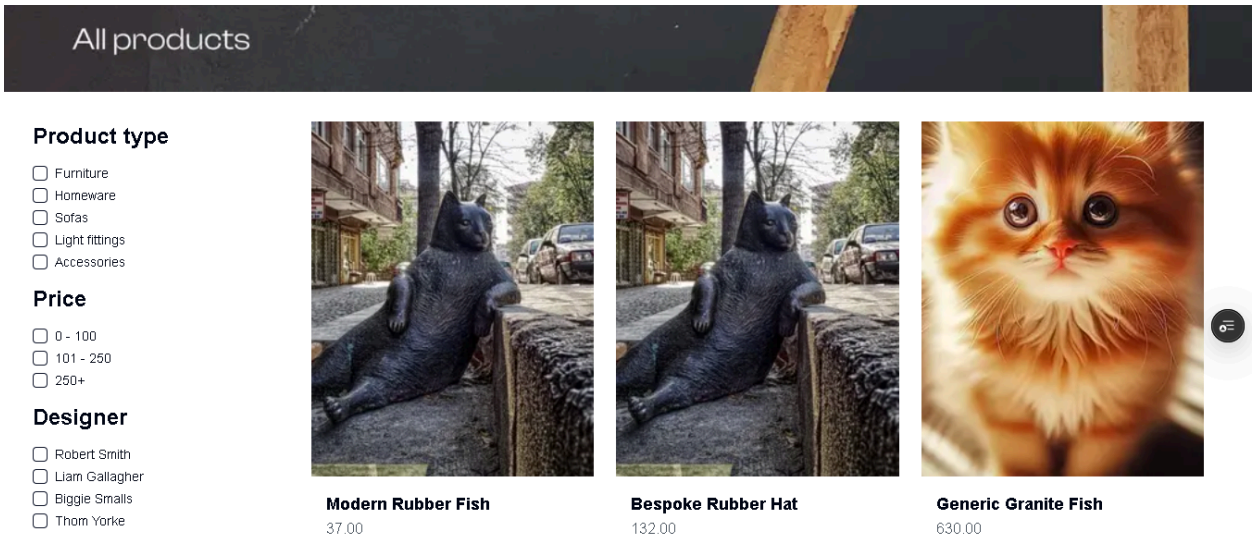
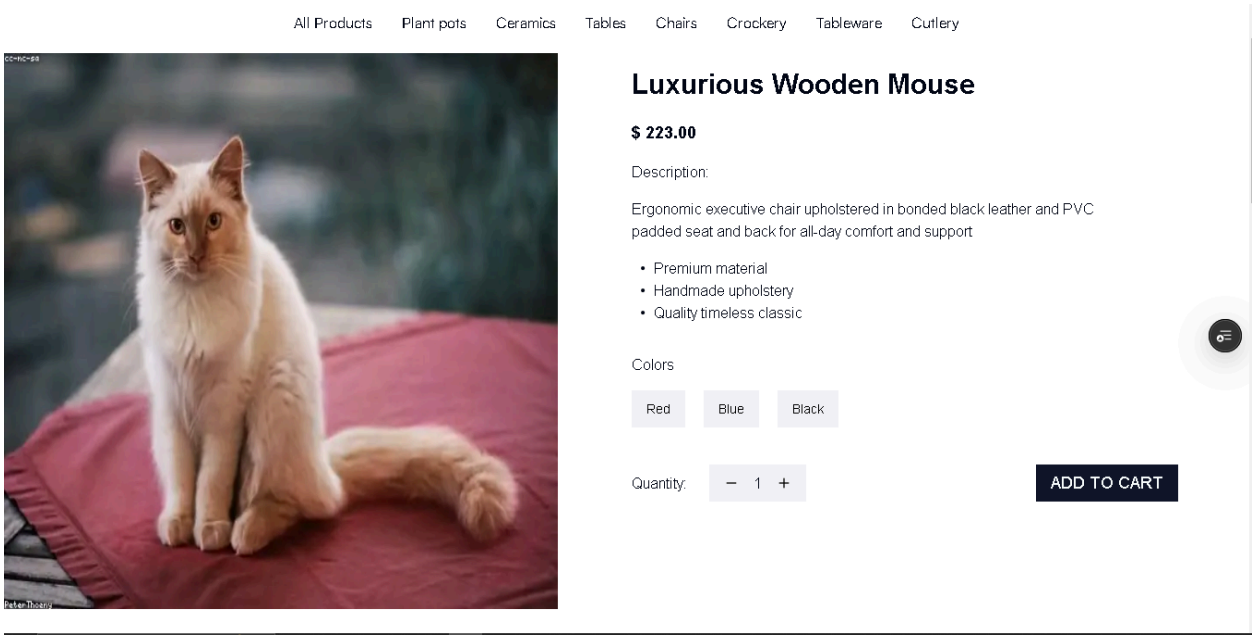
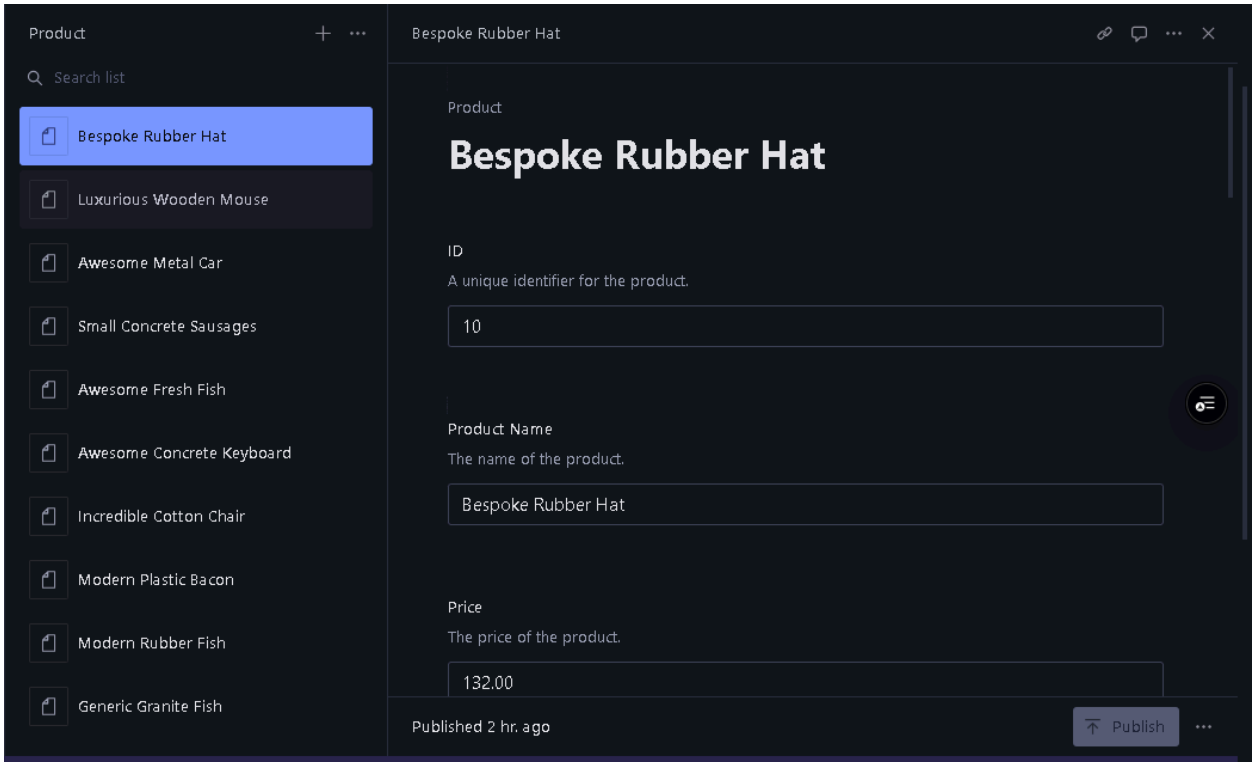
Day 3 - API Integration Report - [Avion]

1. Data displayed on the frontend.

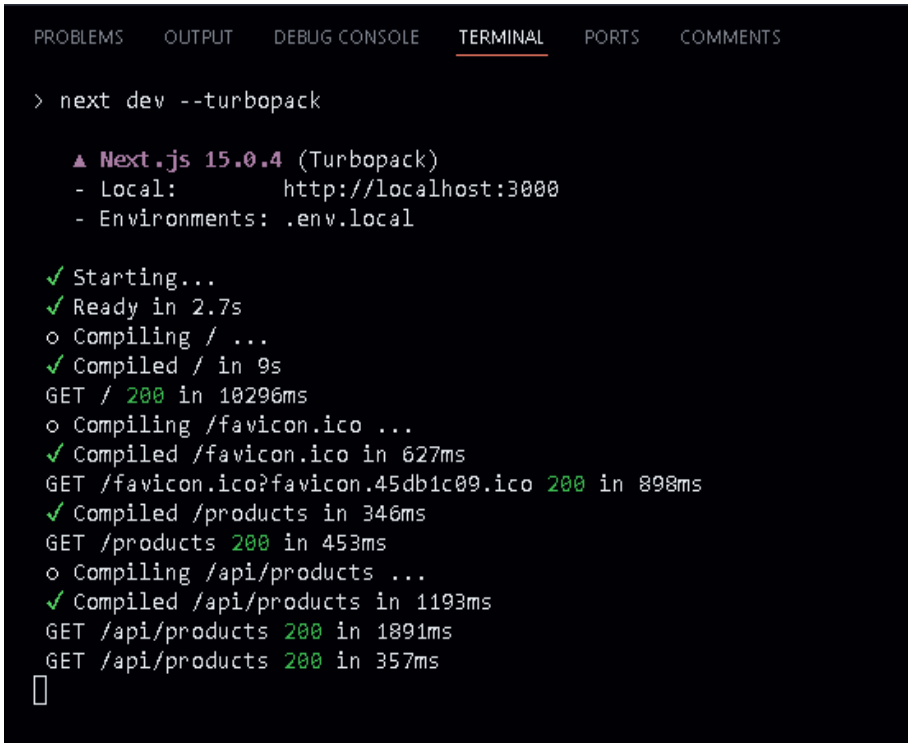




2. Populated Sanity CMS fields.



Successful API calls.



Code Scripts Screen Shots:

app > api > products > TS route.ts > Product

```ts
import { client } from "@/sanity/lib/client";
import axios from "axios";
import { NextResponse } from "next/server";
import { nanoid } from 'nanoid';

interface Variation {
  color: string; // Color of the product (e.g., "Red", "Blue")
  size: string; // Size of the product (e.g., "S", "M", "L")
  quantity: number; // Available quantity for the specific color and size
}

interface Product {
  id: string;
  name: string;
  price: number;
  discountPercentage: number;
  image: string | string[];
  rating: string;
  tags: string[];
  description: string;
  variations: Variation[]; // Variations of the product with different colors, sizes, and quantities
}

const MOCK_API_URL = `${process.env.NEXT_MOCK_API}`;

```

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 > route.ts > uploadImagesToSanity > assets > urls.map() callback
export async function POST() {
try {
const { data: products } = await axios.get<Product[]>(MOCK_API_URL);
console.log("Fetched products:", products);
if (!Array.isArray(products) || products.length === 0) {
return NextResponse.json(
{ success: false, error: "Invalid or empty product data" },
{ status: 400 }
);
}
async function delay(ms: number) {
return new Promise((resolve) => setTimeout(resolve, ms));
}
const sanityOperations = [];
// First, delete all existing products
// await client.delete({query: '*[_type == "product"]'});
for (const product of products) {
await delay(1000);
console.log("Processing product:", product);
console.log("Image URL:", product.image);
const operation = (async () => {
const images = await uploadImagesToSanity(product.image);
const variations = product.variations.map((variation) => ({
color: variation.color,
size: variation.size,
quantity: variation.quantity,
}));

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
```

```
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
```

Day 3 Checklist:

Self-Validation Checklist:

API Understanding: ✔
Schema Validation: ✔
Data Migration: ✔
API Integration in Next.js: ✔
Submission Preparation: ✔