

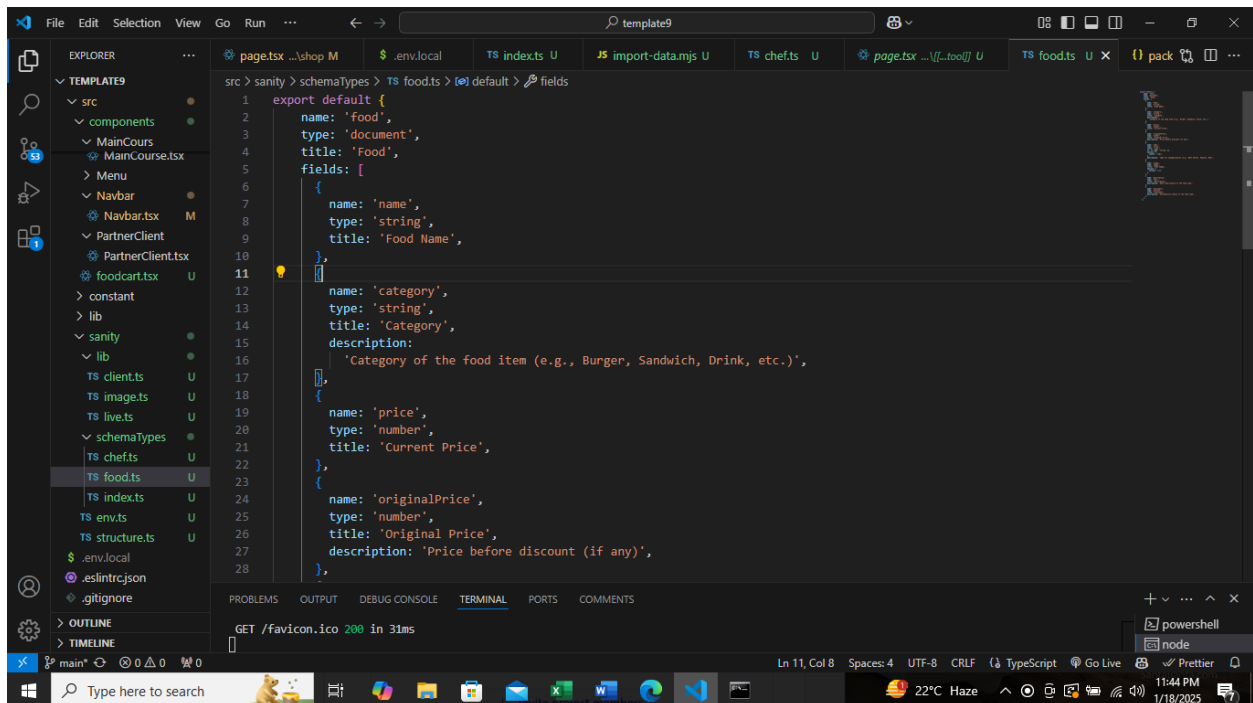
# Hackathon Day3

## API Integration and Migration Report:

### Sanity CMS Schema:

*Sanity Schema defines the structure and field for content in a sanity CMS. It allows to create custom document types like food and chef to organize and manage data efficiently....*

### Food.ts:

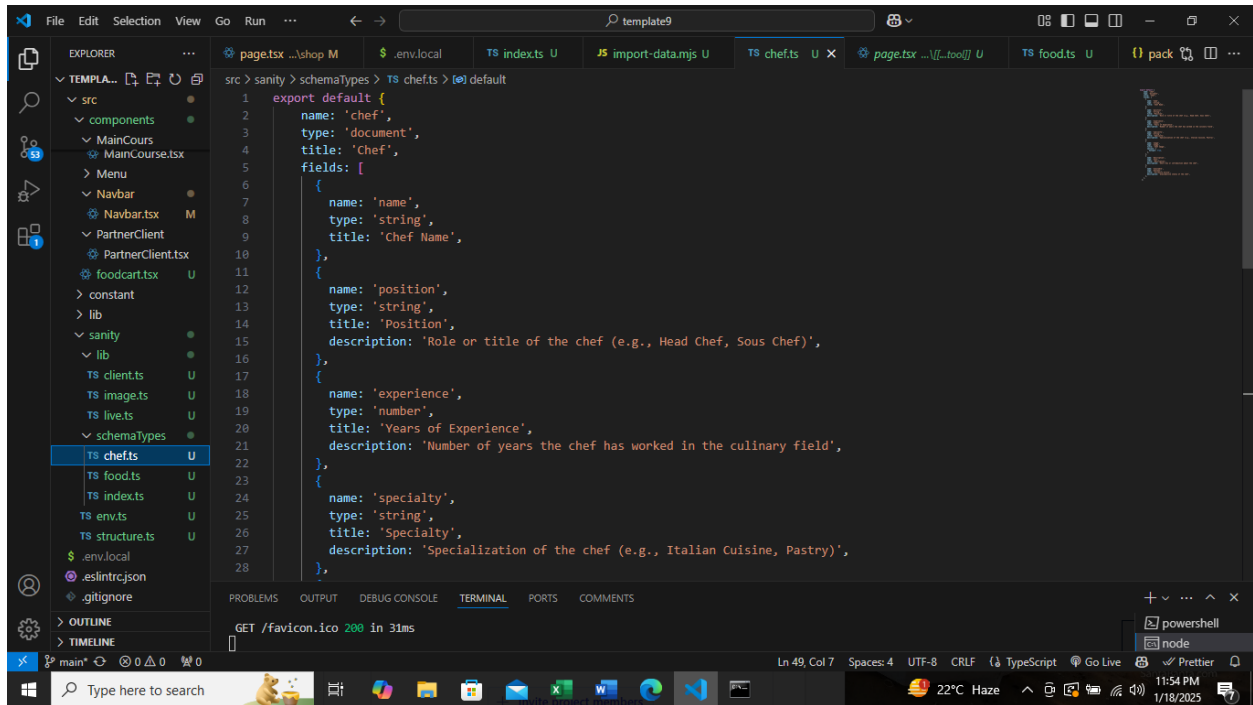


The screenshot shows a Visual Studio Code editor window with the file explorer on the left and the editor on the right. The file explorer shows a project structure with folders like 'src', 'components', 'MainCourse', 'Menu', 'Navbar', 'PartnerClient', 'PartnerClient.tsx', 'foodcart.tsx', 'constant', 'lib', 'sanity', 'lib', 'TS client.ts', 'TS image.ts', 'TS live.ts', 'schemaTypes', 'TS chefs.ts', 'TS food.ts', 'TS index.ts', 'TS env.ts', 'TS structure.ts', '.env.local', '.eslintrc.json', and '.gitignore'. The editor window shows the 'TS food.ts' file with the following TypeScript code:

```
1 export default {
2   name: 'food',
3   type: 'document',
4   title: 'Food',
5   fields: [
6     {
7       name: 'name',
8       type: 'string',
9       title: 'Food Name',
10    },
11  ],
12  {
13    name: 'category',
14    type: 'string',
15    title: 'Category',
16    description:
17      'Category of the food item (e.g., Burger, Sandwich, Drink, etc.)',
18  },
19  {
20    name: 'price',
21    type: 'number',
22    title: 'Current Price',
23  },
24  {
25    name: 'originalPrice',
26    type: 'number',
27    title: 'Original Price',
28    description: 'Price before discount (if any)',
29  },
30  ],
31 }
```

The bottom of the editor shows the 'TERMINAL' tab with the command 'GET /favicon.ico 200 in 31ms' and the 'node' process running in the background.

# Chef.ts:

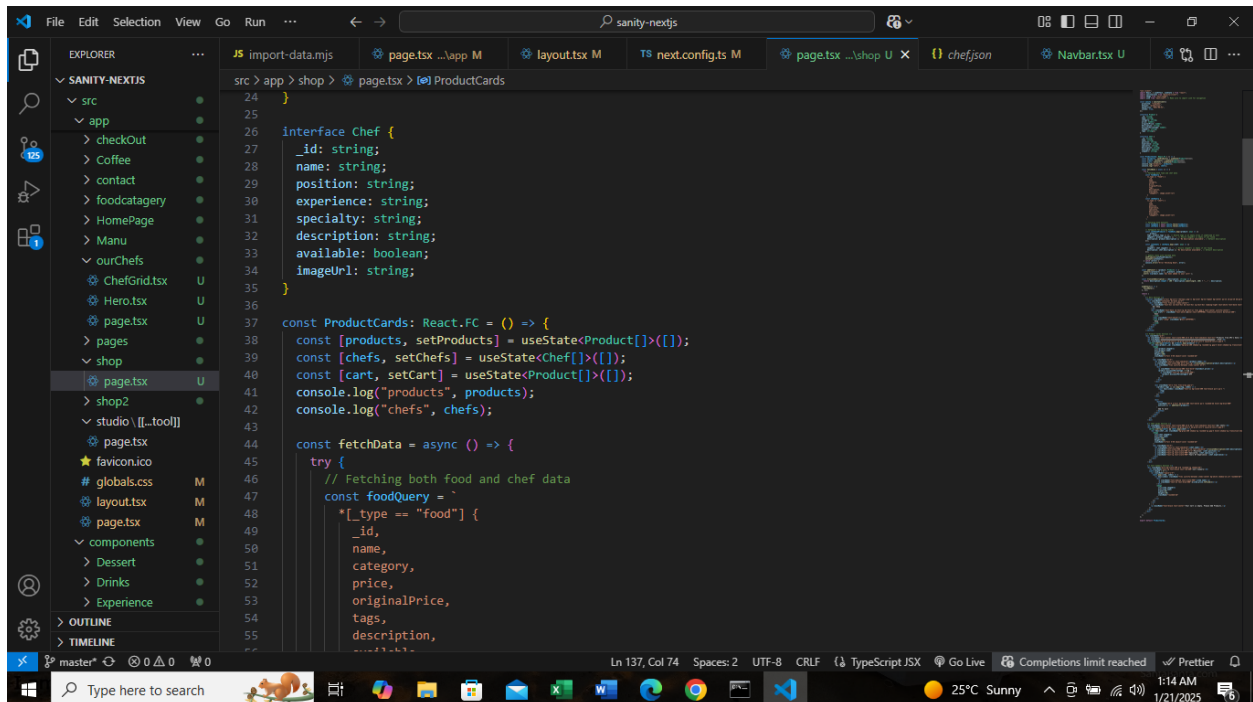


The screenshot shows a Visual Studio Code editor window with a project named 'sanity'. The Explorer panel on the left shows the file structure, with 'src > schemaTypes > TS chefs > default' selected. The main editor displays the content of 'default.ts', which defines a TypeScript interface for a chef. The interface has a 'name' field (string), a 'position' field (string), an 'experience' field (number), and a 'specialty' field (string). Each field has a corresponding title and description. The terminal at the bottom shows a successful GET request for a favicon.

```
1 export default {
2   name: 'chef',
3   type: 'document',
4   title: 'Chef',
5   fields: [
6     {
7       name: 'name',
8       type: 'string',
9       title: 'Chef Name',
10    },
11    {
12      name: 'position',
13      type: 'string',
14      title: 'Position',
15      description: 'Role or title of the chef (e.g., Head Chef, Sous Chef)',
16    },
17    {
18      name: 'experience',
19      type: 'number',
20      title: 'Years of Experience',
21      description: 'Number of years the chef has worked in the culinary field',
22    },
23    {
24      name: 'specialty',
25      type: 'string',
26      title: 'Specialty',
27      description: 'Specialization of the chef (e.g., Italian Cuisine, Pastry)',
28    }
29  ]
30 }
```

GET /favicon.ico 200 in 31ms

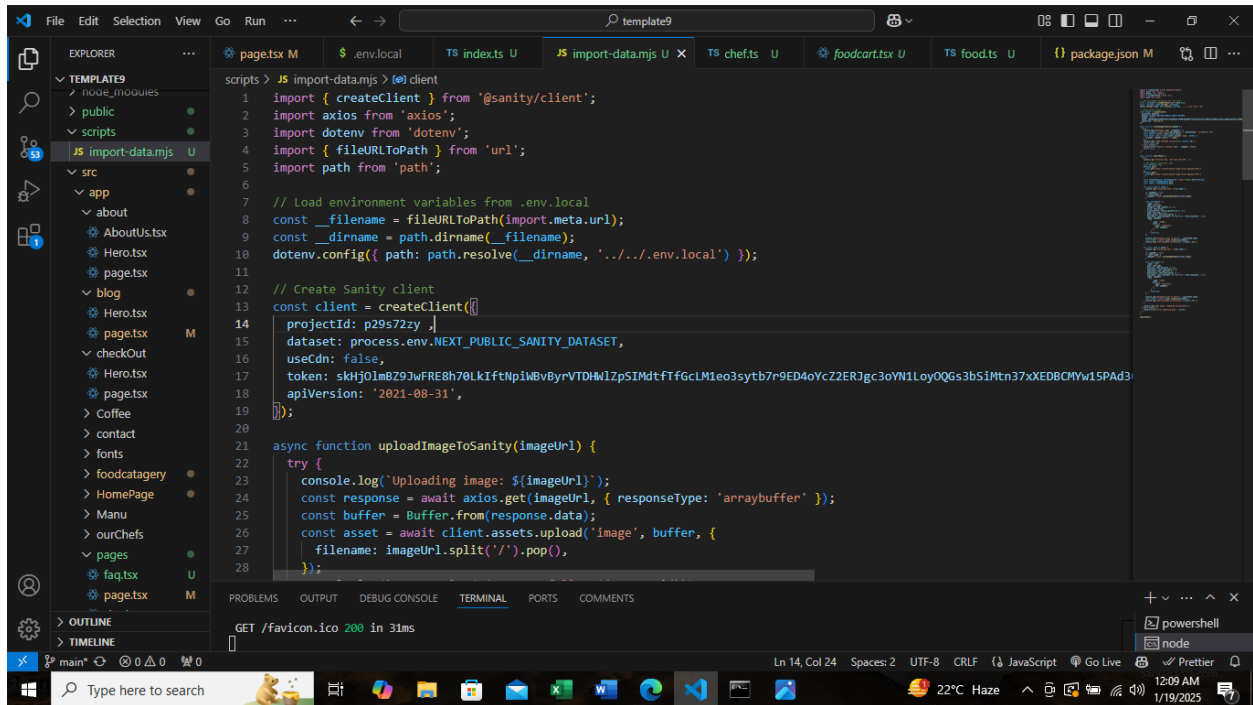
# API Data Fetching:



The screenshot shows a Visual Studio Code editor window with a project named 'sanity-nextjs'. The Explorer panel on the left shows the file structure, with 'src > app > shop > page.tsx' selected. The main editor displays the content of 'page.tsx', which defines a TypeScript interface for a chef and a React component for displaying product cards. The component uses useState to manage the state of products, chefs, and a cart. It also includes a fetchData function that fetches data from an API.

```
24 }
25
26 interface Chef {
27   _id: string;
28   name: string;
29   position: string;
30   experience: string;
31   specialty: string;
32   description: string;
33   available: boolean;
34   imageUrl: string;
35 }
36
37 const ProductCards: React.FC = () => {
38   const [products, setProducts] = useState<Product[]>([]);
39   const [chefs, setChefs] = useState<Chef[]>([]);
40   const [cart, setCart] = useState<Product[]>([]);
41   console.log("products", products);
42   console.log("chefs", chefs);
43
44   const fetchData = async () => {
45     try {
46       // Fetching both food and chef data
47       const foodQuery = `
48         *[_type == "food"] {
49           _id,
50           name,
51           category,
52           price,
53           originalPrice,
54           tags,
55           description,
56         }
57       `;
58     } catch (error) {
59       console.error(error);
60     }
61   };
62 }
```

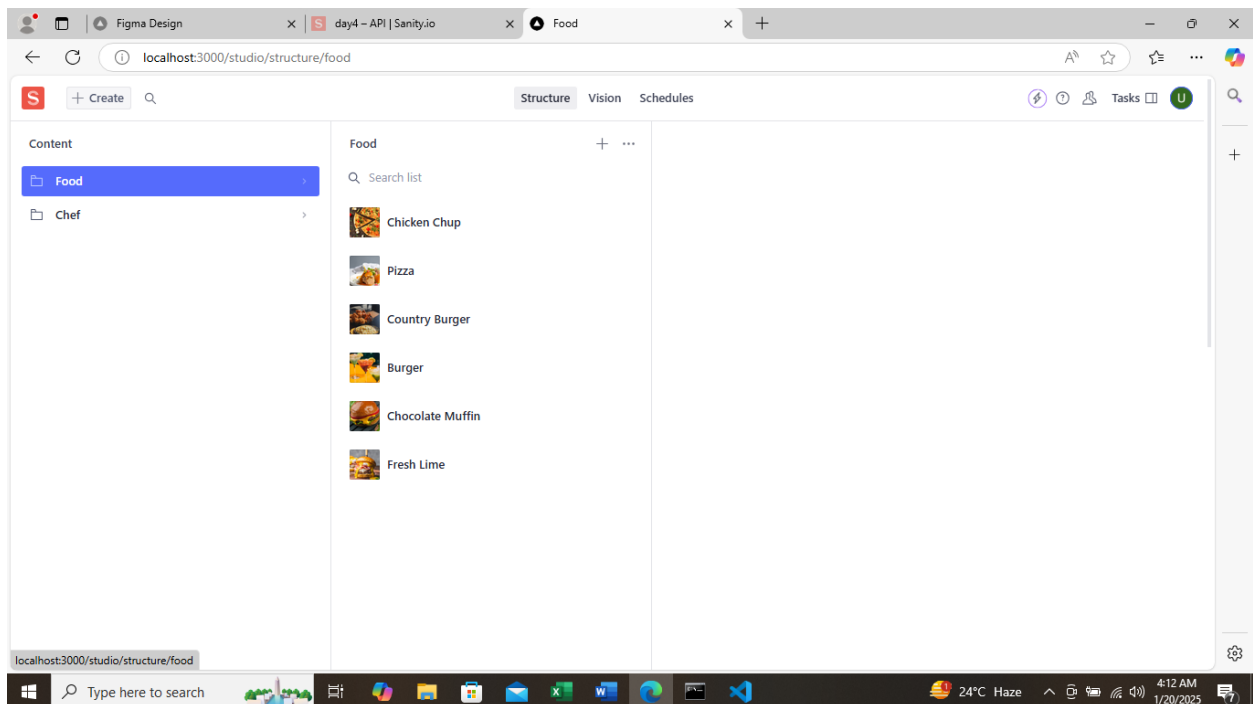
# Data Migration:

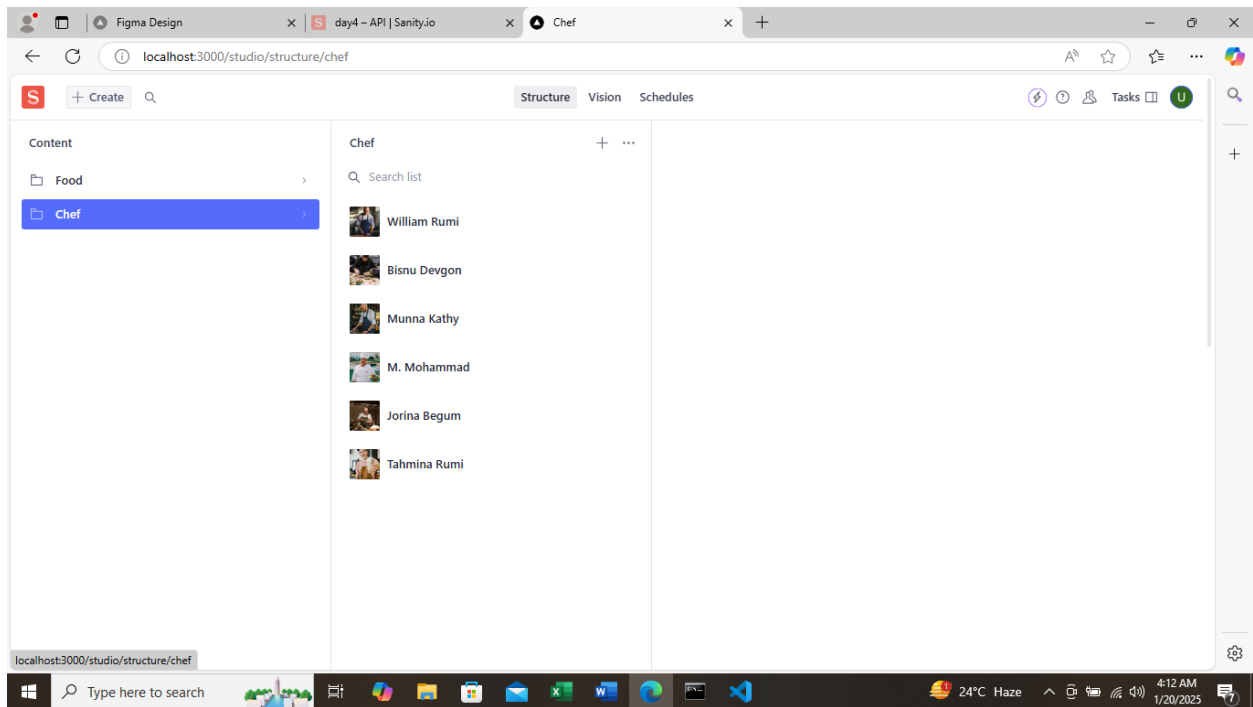


```
1 import { createClient } from '@sanity/client';
2 import axios from 'axios';
3 import dotenv from 'dotenv';
4 import { fileURLToPath } from 'url';
5 import path from 'path';
6
7 // Load environment variables from .env.local
8 const __filename = fileURLToPath(import.meta.url);
9 const __dirname = path.dirname(__filename);
10 dotenv.config({ path: path.resolve(__dirname, '../.env.local') });
11
12 // Create Sanity client
13 const client = createClient({
14   projectId: 'p29s72zy',
15   dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
16   useCdn: false,
17   token: 'skHj0lmbZ9JwFRE8h70LkIfNpimBvByrVTDHnL2pSIMdtFTfGcLM1eo3syb7r9ED4oYcZ2ERJgc3oYn1LoyOQ6s3bSiMtn37xxEDBCMYw15Pad3',
18   apiVersion: '2021-08-31',
19 });
20
21 async function uploadImageToSanity(imageUrl) {
22   try {
23     console.log('Uploading image: ${imageUrl}');
24     const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
25     const buffer = Buffer.from(response.data);
26     const asset = await client.assets.upload('image', buffer, {
27       filename: imageUrl.split('/').pop(),
28     });
29   } catch (error) {
30     console.error('Error uploading image: ', error);
31   }
32 }
```

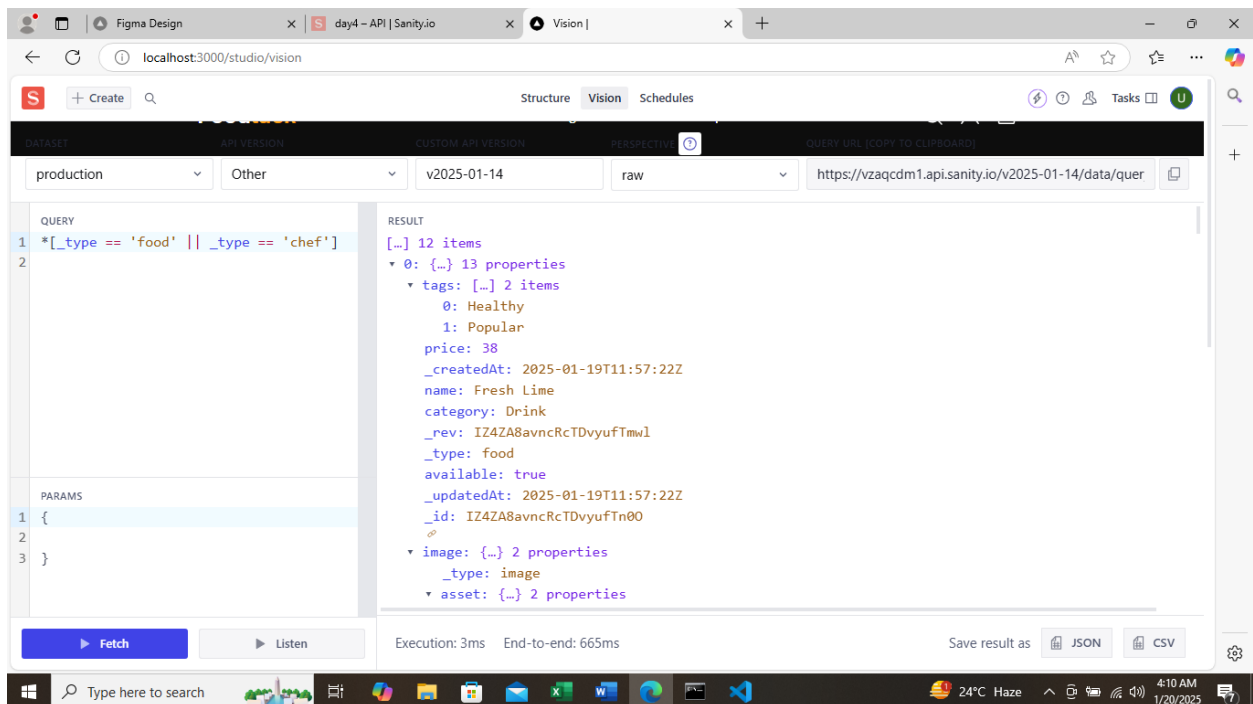
GET /favicon.ico 200 in 31ms

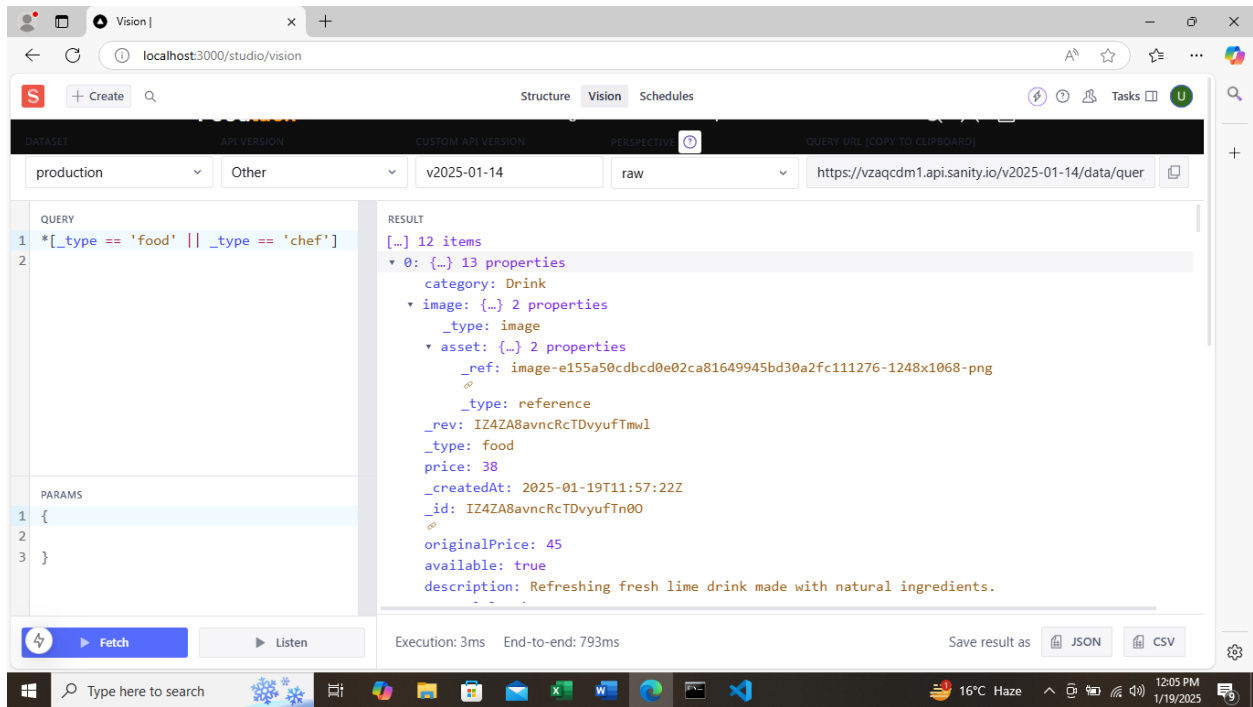
# Sanity Studio:



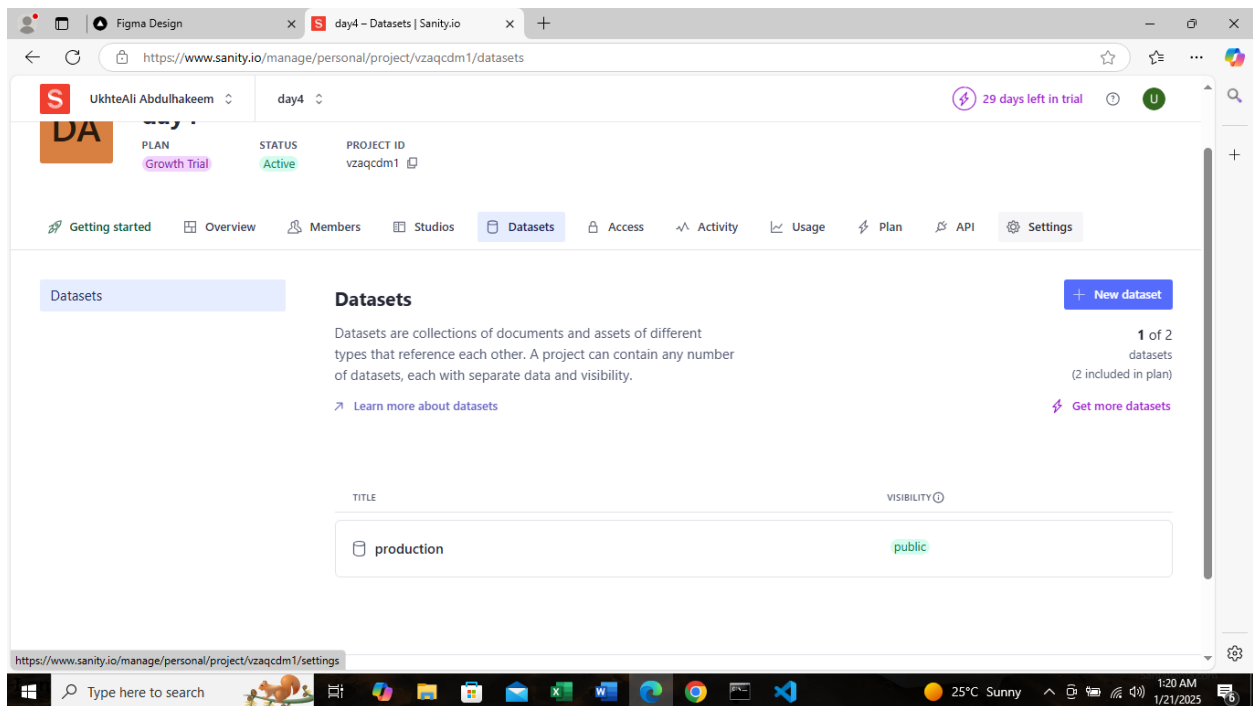


## Sanity Vision:

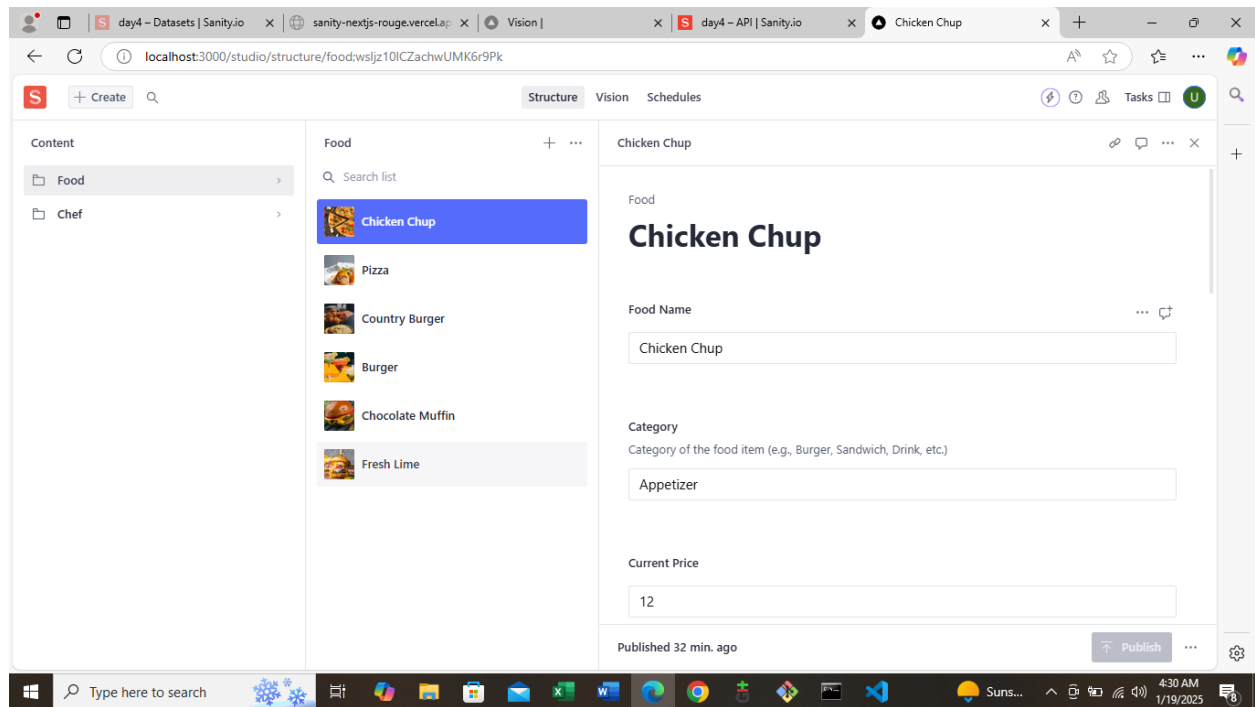




## Sanity DataBase:



## Product Detail page:



## Conclusion:

The API Integration and migration were successfully completed, improving the efficiency and scalability of the food tuck website. This integration simplified the process of updating food items, while migration ensured data accuracy across the system.