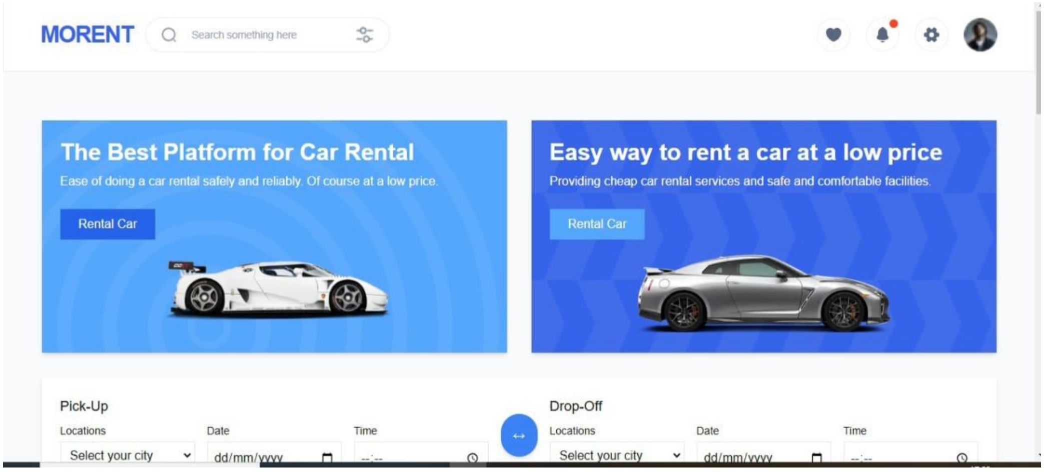
Hackathon Day 3

API Integration Report [Rental E-Commerce]

Project Name: Rental E-Commerce Marketplace.

Prepared by: [Tahira Ibrahim (103987].

Date: [18-01-25]



The Best Platform for Car Rental

Ease of doing a car rental safely and reliably. Of course at a low price.

Easy way to rent a car at a low price

Providing cheap car rental services and safe and comfortable facilities.

Objective

The purpose of this report is to document the process of API integration and data migration for the Rental E-Commerce Marketplace. The goal is to utilize provided APIs, customize schemas, migrate data into Sanity CMS, and ensure seamless API integration in a Next.js application.

Table of Contents

- Understanding API integration in Next.js.
- Migrating data from external sources into Sanity CMS.
- Validating and customizing schemas for data compatibility.
- Implementing best practices for marketplace development.

Sanity Installation:

For detailed instructions on installing Sanity, refer to the following guide: Sanity Setup Guide. If you already have a Next.js application, you can skip the first step of creating a new project.

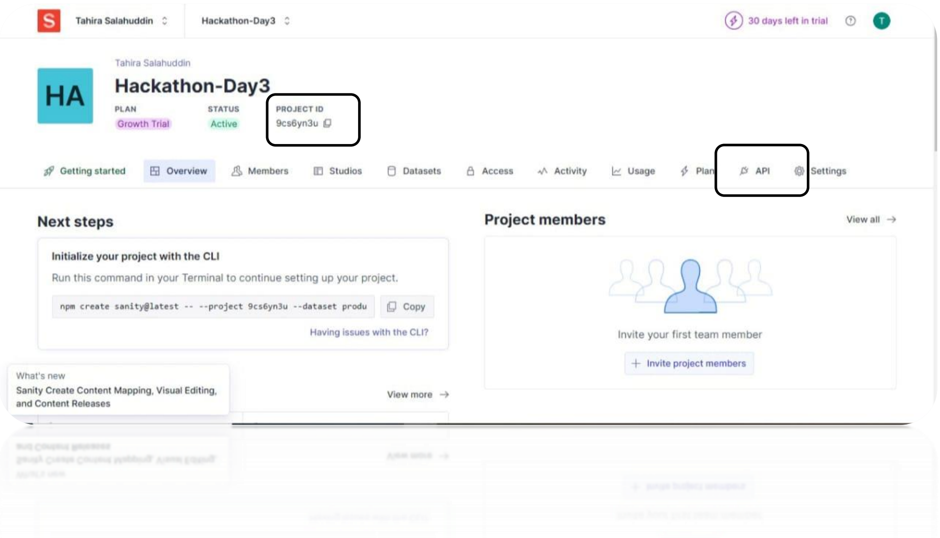
Api Integration and Data Migration:

create a. env. local file at the root of your project.

The file with the following entries:

```
NEXT_PUBLIC_SANITY_PROJECT_ID=your_project_id

NEXT_PUBLIC_SANITY_DATASET=production

SANITY_API_TOKEN=your_sanity_token
```

- **Fetching Sanity Project ID and API Token**
  Project ID.
- Log in to Sanity account at: https://www.sanity.io/manage/personal/project/9cs6yn3u
- Select project then in the project dashboard, you'll see the project ID listed.



Tahira Salahuddin · Hackathon-Day3 · 30 days left in trial

Tahira Salahuddin

Hackathon-Day3

PLAN: Growth Trial · STATUS: Active · PROJECT ID: 9cs6yn3u

Next steps

Initialize your project with the CLI

Run this command in your Terminal to continue setting up your project.

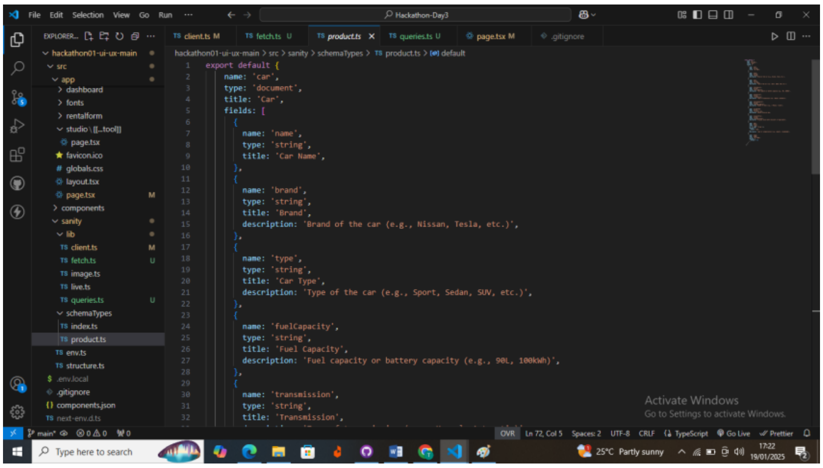
```
npm create sanity@latest -- --project 9cs6yn3u --dataset produ
```

What's new
Sanity Create Content Mapping, Visual Editing, and Content Releases

- Use this token for the `SANITY_API_TOKEN` in `. env. local` file.

Sanity Schema

In the sanity/schema Types folder, create a new file product.ts:

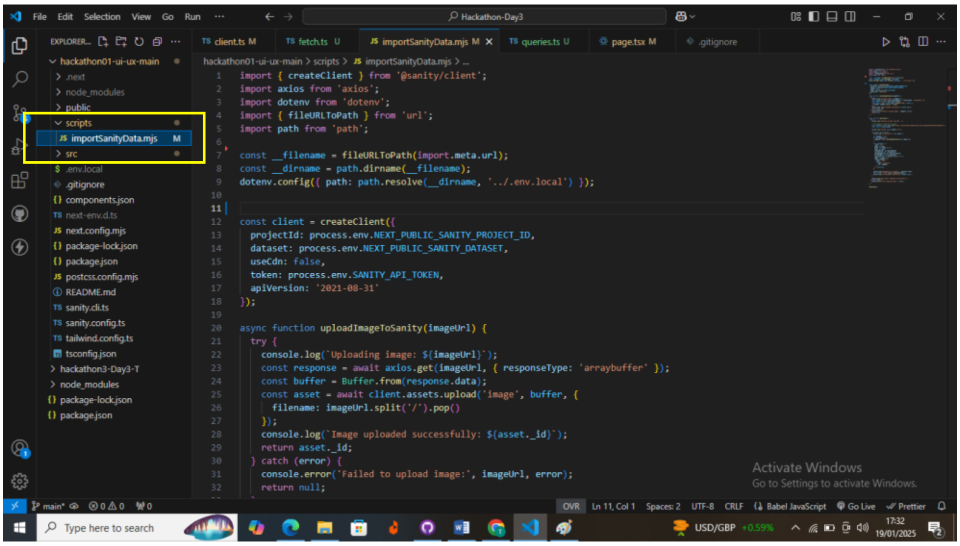


```
export default {
  name: 'car',
  type: 'document',
  title: 'Car',
  fields: [
    {
      name: 'name',
      type: 'string',
      title: 'Car Name',
    },
    {
      name: 'brand',
      type: 'string',
      title: 'Brand',
      description: 'Brand of the car (e.g., Nissan, Tesla, etc.)',
    },
    {
      name: 'type',
      type: 'string',
      title: 'Car Type',
      description: 'Type of the car (e.g., Sport, Sedan, SUV, etc.)',
    },
    {
      name: 'fuelCapacity',
      type: 'string',
      title: 'Fuel Capacity',
      description: 'Fuel capacity or battery capacity (e.g., 90L, 100kWh)',
    },
    {
      name: 'transmission',
      type: 'string',
      title: 'Transmission',
```

- Then, update your `sanity/schema Types/index.ts` file to include the new product schema.

```
import { type SchemaTypeDefinition } from 'sanity'
import product from './product'

export const schema: { types: SchemaTypeDefinition[] } = {
types: [cars],
}
```

- Create folder of script and file of import data .js.
- create a script to import data from an external API into Sanity. Create a new file `scripts/importSanityData.mjs` in your project root:

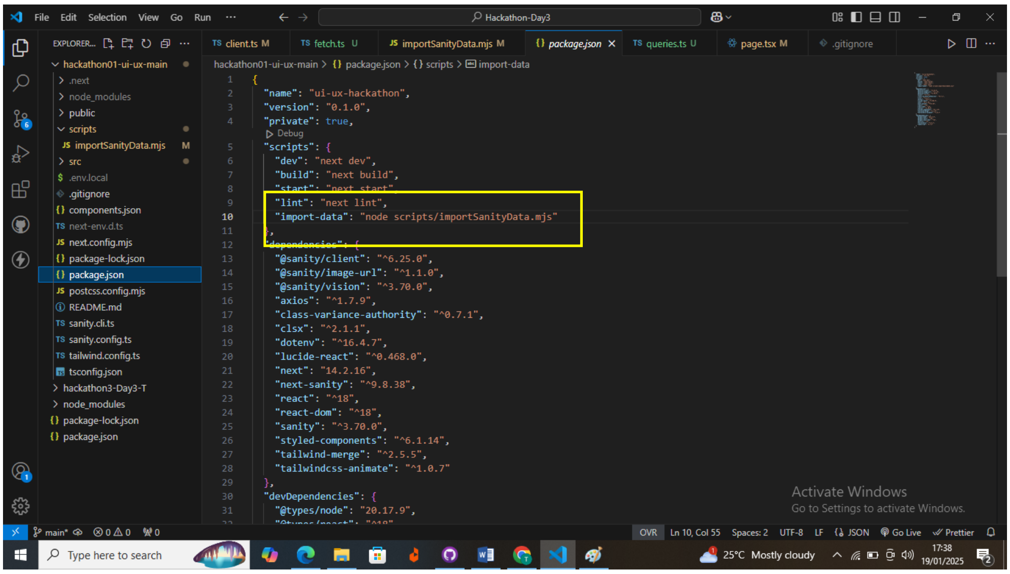


```
import { createClient } from '@sanity/client';
import axios from 'axios';
import dotenv from 'dotenv';
import { fileURLToPath } from 'url';
import path from 'path';

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
dotenv.config({ path: path.resolve(__dirname, '../.env.local') });


const client = createClient({
  projectId: process.env.NEXT_PUBLIC_SANITY_PROJECT_ID,
  dataset: process.env.NEXT_PUBLIC_SANITY_DATASET,
  useCdn: false,
  token: process.env.SANITY_API_TOKEN,
  apiVersion: '2021-08-31'
});

async function uploadImageToSanity(imageUrl) {
  try {
    console.log(`Uploading image: ${imageUrl}`);
    const response = await axios.get(imageUrl, { responseType: 'arraybuffer' });
    const buffer = Buffer.from(response.data);
    const asset = await client.assets.upload('image', buffer, {
      filename: imageUrl.split('/').pop()
    });
    console.log(`Image uploaded successfully: ${asset._id}`);
    return asset._id;
  } catch (error) {
    console.error('Failed to upload image:', imageUrl, error);
    return null;
```

- Run the following command in your terminal:

```
npm install @sanity/client axios dotenv
```

- Now run the import script



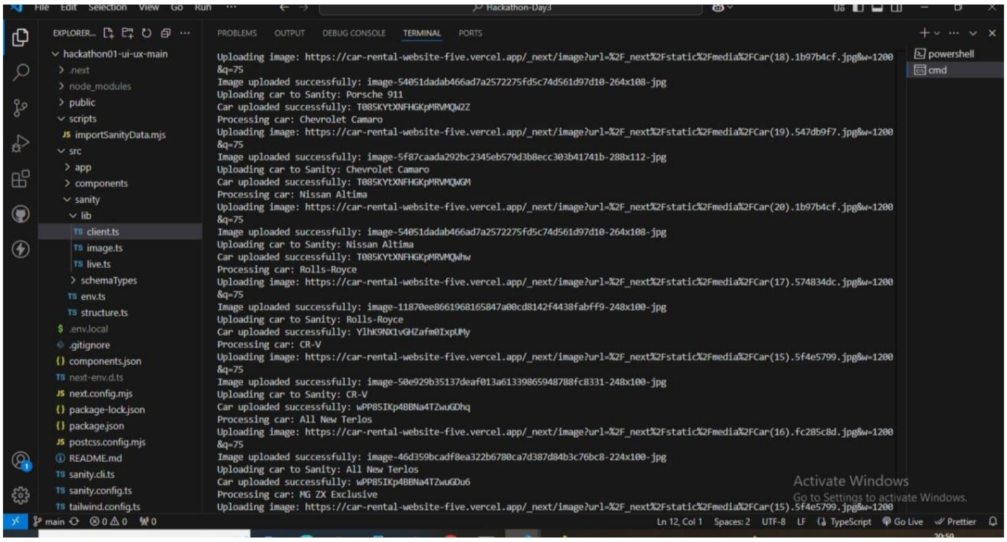
```
{
  "name": "ui-ux-hackathon",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "import-data": "node scripts/importSanityData.mjs"
  },
  "dependencies": {
    "@sanity/client": "^6.25.0",
    "@sanity/image-url": "^1.1.0",
    "@sanity/vision": "^3.70.0",
    "axios": "^1.7.9",
    "class-variance-authority": "^0.7.1",
    "clsx": "^2.1.1",
    "dotenv": "^16.4.7",
    "lucide-react": "^0.468.0",
    "next": "14.2.16",
    "next-sanity": "^9.8.38",
    "react": "^18",
    "react-dom": "^18",
    "sanity": "^3.70.0",
    "styled-components": "^6.1.14",
    "tailwind-merge": "^2.5.5",
    "tailwindcss-animate": "^1.0.7"
  },
  "devDependencies": {
    "@types/node": "20.17.9",
```

- To run the import script, we need to add a new script to our `package.json` file. Open your `package. json` and add the following to the `"scripts"` section.

```
"scripts": {
  "dev": "next dev --turbopack",
  "build": "next build",
  "start": "next start",
  "lint": "next lint",
  "import-data": "node scripts/importSanityData.mjs"
}
```

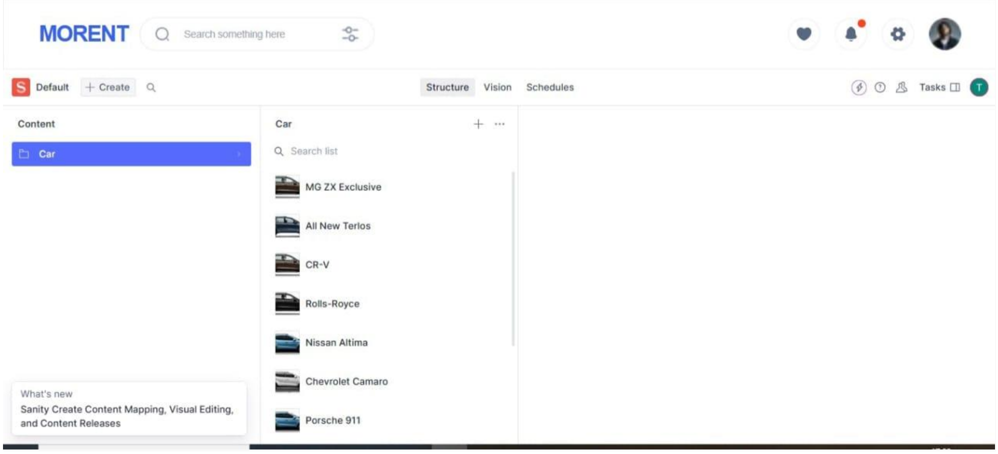
- Now run the import script using.

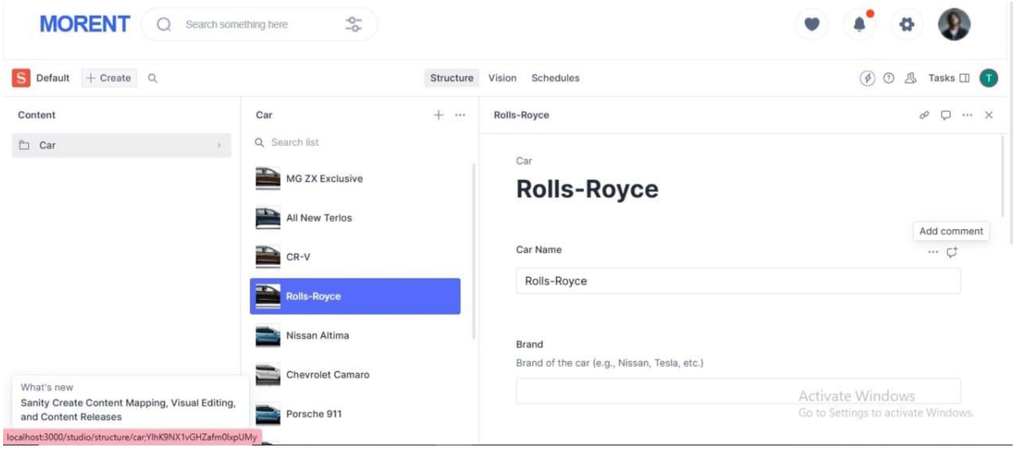
```
npm run import-data
```



```
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2FCar(18).1b97b4cf.jpg&w=1200
&q=75
Image uploaded successfully: image-54051dadab466ad7a2572275fd5c74d561d97d10-264x108-jpg
Uploading car to Sanity: Porsche 911
Car uploaded successfully: T085KYtXNFHGKpMRVMQW2Z
Processing car: Chevrolet Camaro
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2FCar(19).547db9f7.jpg&w=1200
&q=75
Image uploaded successfully: image-5f87caada292bc2345eb579d3b8ecc303b41741b-288x112-jpg
Uploading car to Sanity: Chevrolet Camaro
Car uploaded successfully: T085KYtXNFHGKpMRVMQWGM
Processing car: Nissan Altima
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2FCar(20).1b97b4cf.jpg&w=1200
&q=75
Image uploaded successfully: image-54051dadab466ad7a2572275fd5c74d561d97d10-264x108-jpg
Uploading car to Sanity: Nissan Altima
Car uploaded successfully: T085KYtXNFHGKpMRVMQWhw
Processing car: Rolls-Royce
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2FCar(17).574834dc.jpg&w=1200
&q=75
Image uploaded successfully: image-11870ee8661968165847a00cd8142f4438fabff9-248x100-jpg
Uploading car to Sanity: Rolls-Royce
Car uploaded successfully: YlhK9NX1vGHZafm0IxpUMy
Processing car: CR-V
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2FCar(15).5f4e5799.jpg&w=1200
&q=75
Image uploaded successfully: image-50e929b35137deaf013a61339865948788fc8331-248x100-jpg
Uploading car to Sanity: CR-V
Car uploaded successfully: wPP85IKp4BBNa4TZwuGDhq
Processing car: All New Terlos
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2FCar(16).fc285c8d.jpg&w=1200
&q=75
Image uploaded successfully: image-46d359bcadf8ea322b6780ca7d387d84b3c76bc8-224x100-jpg
Uploading car to Sanity: All New Terlos
Car uploaded successfully: wPP85IKp4BBNa4TZwuGDu6
Processing car: MG ZX Exclusive
Uploading image: https://car-rental-website-five.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2FCar(15).5f4e5799.jpg&w=1200
```

- `npm run dev.`





Car

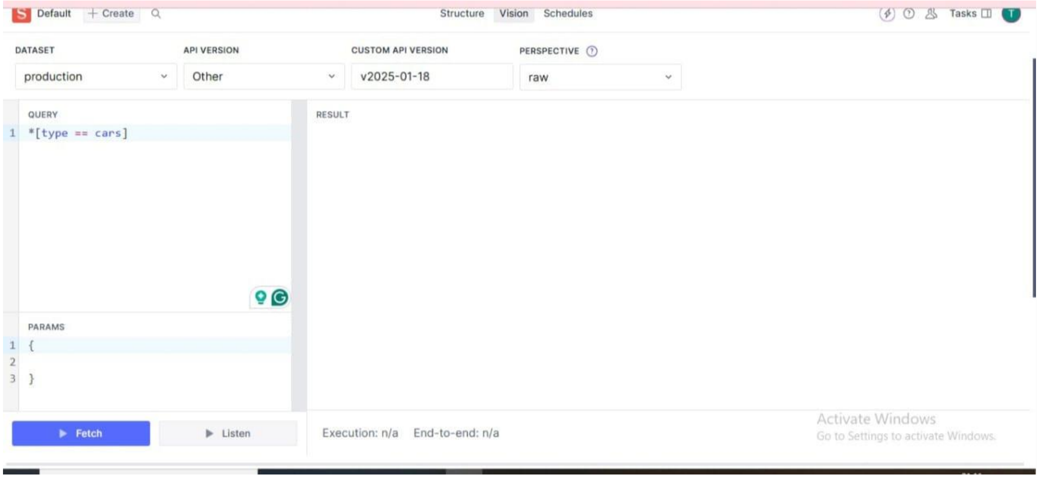
Rolls-Royce

Car Name: Rolls-Royce

Brand
Brand of the car (e.g., Nissan, Tesla, etc.)

- In Studio write this command in vision to see fetch data.

```
*[_type == cars]
```



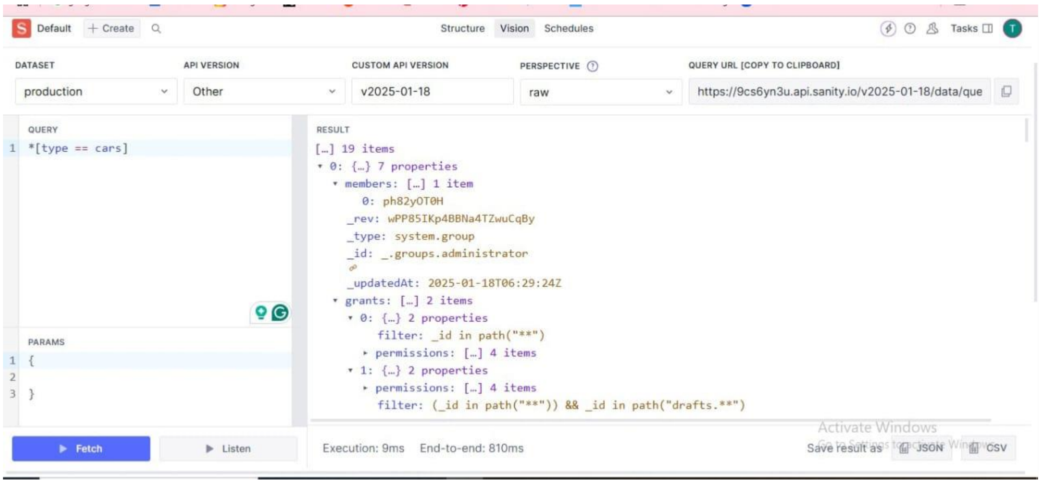
DATASET: production
API VERSION: Other
CUSTOM API VERSION: v2025-01-18
PERSPECTIVE: raw

QUERY
```
*[type == cars]
```

PARAMS
```
{

}
```

RESULT

Execution: n/a   End-to-end: n/a



DATASET: production
API VERSION: Other
CUSTOM API VERSION: v2025-01-18
PERSPECTIVE: raw
QUERY URL [COPY TO CLIPBOARD]: https://9cs6yn3u.api.sanity.io/v2025-01-18/data/que

QUERY
```
*[type == cars]
```

PARAMS
```
{

}
```

RESULT
```
[…] 19 items
▾ 0: {…} 7 properties
  ▾ members: […] 1 item
      0: ph82yOT0H
    _rev: wPP85IKp4BBNa4TZwuCqBy
    _type: system.group
    _id: _.groups.administrator
    _updatedAt: 2025-01-18T06:29:24Z
  ▾ grants: […] 2 items
    ▾ 0: {…} 2 properties
        filter: _id in path("**")
      ▸ permissions: […] 4 items
    ▾ 1: {…} 2 properties
      ▸ permissions: […] 4 items
        filter: (_id in path("**")) && _id in path("drafts.**")
```

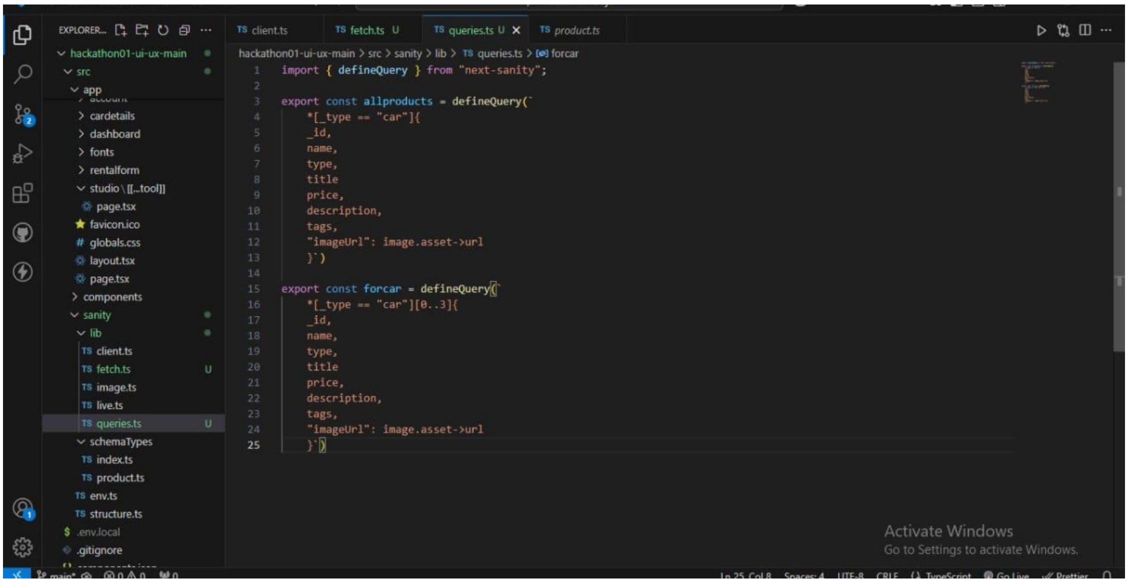
Execution: 9ms   End-to-end: 810ms

Fetch and Display Sanity Data

Set up the Sanity Client:

- Create a utility file utils/sanityClient.js:

```
import {createClient} from 'next-sanity'

import {apiVersion, dataset, projectId} from '../env'

export const client = createClient ({
  projectId: "9cs6yn3u",
  dataset: "production",
  apiVersion: "2023-01-01",
  useCdn: true, // Set to false if statically generating pages, using
ISR or tag-based revalidation
  token:"skz7qtrqNa4ttNmBkjpl47DVI1bTYlgUlveTdRAy9dfkFm35FdMGpepfCbuZn0
dHy4NyA9FIyrZz6YA7xf7cEBDUsaWGdVhoZyyeNP7HykMTZXZXUTYHS1bK80ZUQ3hdrd41n
3K2Net2VBDzVcy6NsGEkZu1Ywq2yKQbmJeUktFEGWKheL0t"
})
```



```
import { defineQuery } from "next-sanity";

export const allproducts = defineQuery(`
    *[_type == "car"]{
    _id,
    name,
    type,
    title
    price,
    description,
    tags,
    "imageUrl": image.asset->url
    }`)

export const forcar = defineQuery(`
    *[_type == "car"][0..3]{
    _id,
    name,
    type,
    title
    price,
    description,
    tags,
    "imageUrl": image.asset->url
    }`)
```

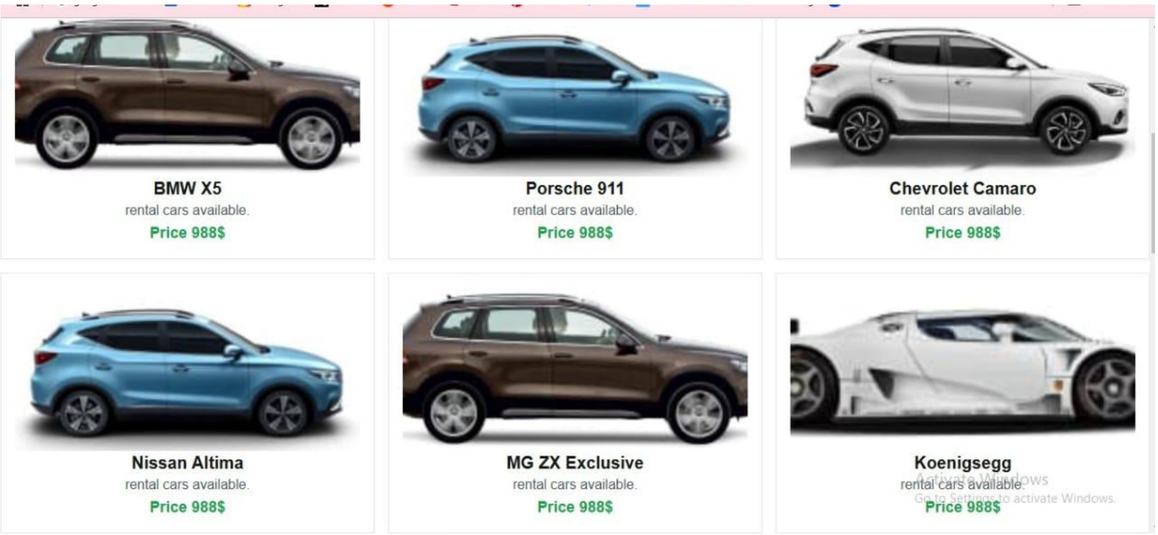
Query to Fetch Data:

- Use GROQ (Graph-Relational Object Queries) to fetch products.

const query = '*[_type == "cars"] { title, details, price, "imageUrl": image.asset->url }';

```
import { sanityFetch } from "@/sanity/lib/fetch";
import { allproducts } from "@/sanity/lib/queries";


type Car = {
  _id: string;
  name: string;
  description: string;
  price: number | null;
  imageUrl: string | null;
};


const Home = async () => {

  const cars: Car [] = await sanityFetch({querry: allproducts });

  return (
    <div>
      <h1 className="text-3xl font-bold text-centre mb-8">Cars</h1>
      <div className="grid grid-cols-3 gap-4">
        {cars.map((car) => (
          <div
            className="border p-4 rounded-lg shadow-sm flex flex-col items-
centre"
            key={car. Id}
          >
            <img
              src={car.imageUrl || "/placeholder-image.jpg"}
              alt={car.name || "Unnamed Car"}
              className="w-full h-48 object-cover"
            />
            <h2 className="text-xl font-bold text-centre">{car.name ||
"Unnamed Car"}</h2>
            <p className="text-centre text-gray-600">
              {car. Description || "rental cars available."}
            </p>
            <p className="text-centre text-lg font-semi bold text-green-600">
              {car. Price! = null? `$${car. price. toFixed (2)} `: "Price
988$"}
            </p>
          </div>
        ))}
      </div>
    </div>
  );
};

export default Home;
```

- Now run command npm run dev.
- Now show the fetched data in frontend.



BMW X5
rental cars available.
Price 988$

Porsche 911
rental cars available.
Price 988$

Chevrolet Camaro
rental cars available.
Price 988$

Nissan Altima
rental cars available.
Price 988$

MG ZX Exclusive
rental cars available.
Price 988$

Koenigsegg
rental cars available.
Price 988$

Conclusion:

By completing this guide, I integrated Sanity into Next.js project and successfully imported external data. Key steps included configuring environment variables, defining a custom schema, and using a script to automate data migration.