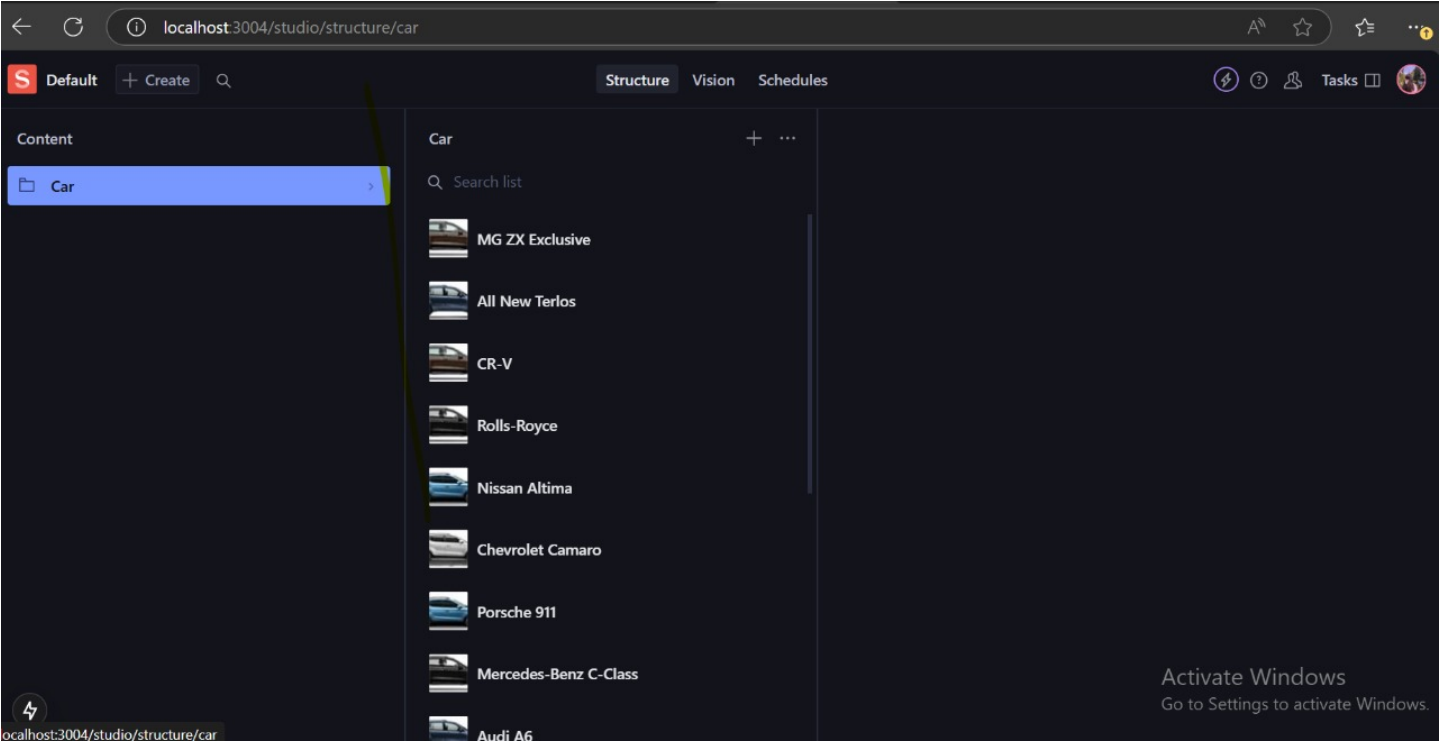


# Rental Car E-commerce: Integration Documentation

This document provides a comprehensive overview of the Rental Car E-commerce project, focusing on tasks such as integrating Sanity CMS, creating schemas, importing car data via APIs, and setting up the environment. The screenshots below outline the process.

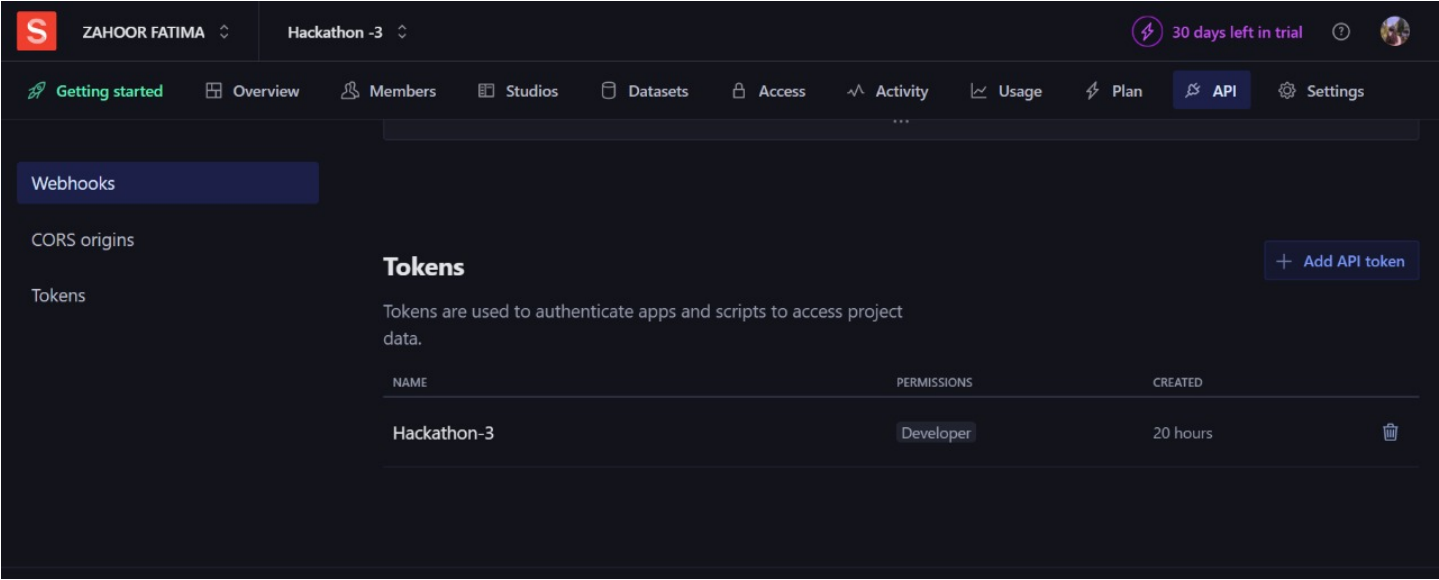
Sanity Studio Car Data Structure:

This screenshot shows car data structured in Sanity CMS, including models like MG ZX Exclusive and Rolls-Royce.



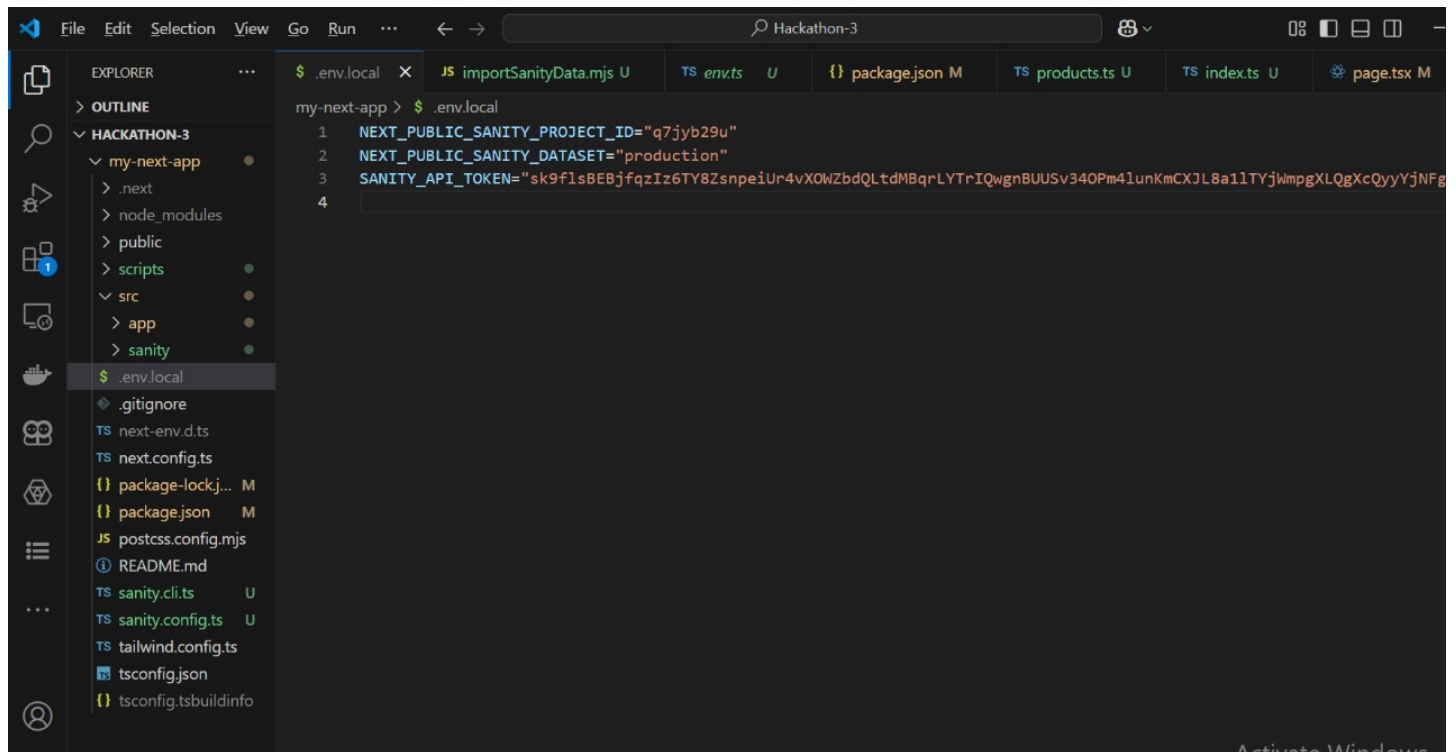
API Token Setup in Sanity:

This image demonstrates how API tokens are configured in Sanity to enable secure data access for the application.



## Environment Variable Setup:

Sensitive credentials such as the Sanity project ID, dataset, and token are securely stored in a `.env` file.



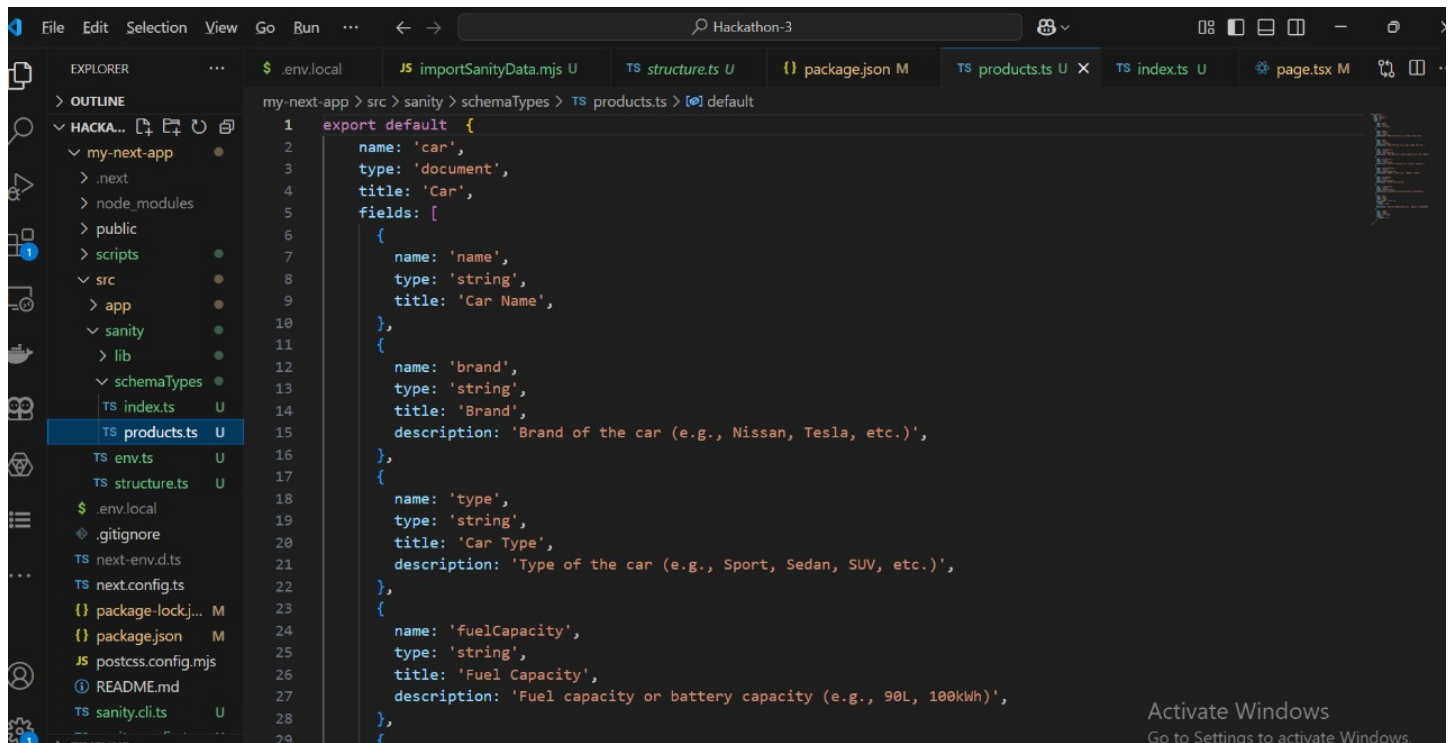
The screenshot shows the Visual Studio Code interface with a project named 'HACKATHON-3'. The Explorer sidebar on the left shows the file structure, with the `.env.local` file selected under the `src` directory. The main editor window displays the contents of `.env.local`, which contains three environment variables for Sanity:

```
my-next-app > $ .env.local
1 NEXT_PUBLIC_SANITY_PROJECT_ID="q7jyb29u"
2 NEXT_PUBLIC_SANITY_DATASET="production"
3 SANITY_API_TOKEN="sk9f1sBEBjfqzIz6TY8ZsnpeiUr4vXOWZbdQLtdMBqrLYTrIQwgnBUUSv34OPm41unKmCXJL8a11TYjWmpgXLQgXcQyyYjNFg"
4
```

The file explorer on the left lists various files including `next-env.d.ts`, `next.config.ts`, `package-lock.json`, `package.json`, `postcss.config.mjs`, `README.md`, `sanity.cli.ts`, `sanity.config.ts`, `tailwind.config.ts`, `tsconfig.json`, and `tsconfig.tsbuildinfo`.

## Car Schema Definition:

Defines fields in Sanity CMS, such as car name, brand, type, and fuel capacity, ensuring structured data storage.



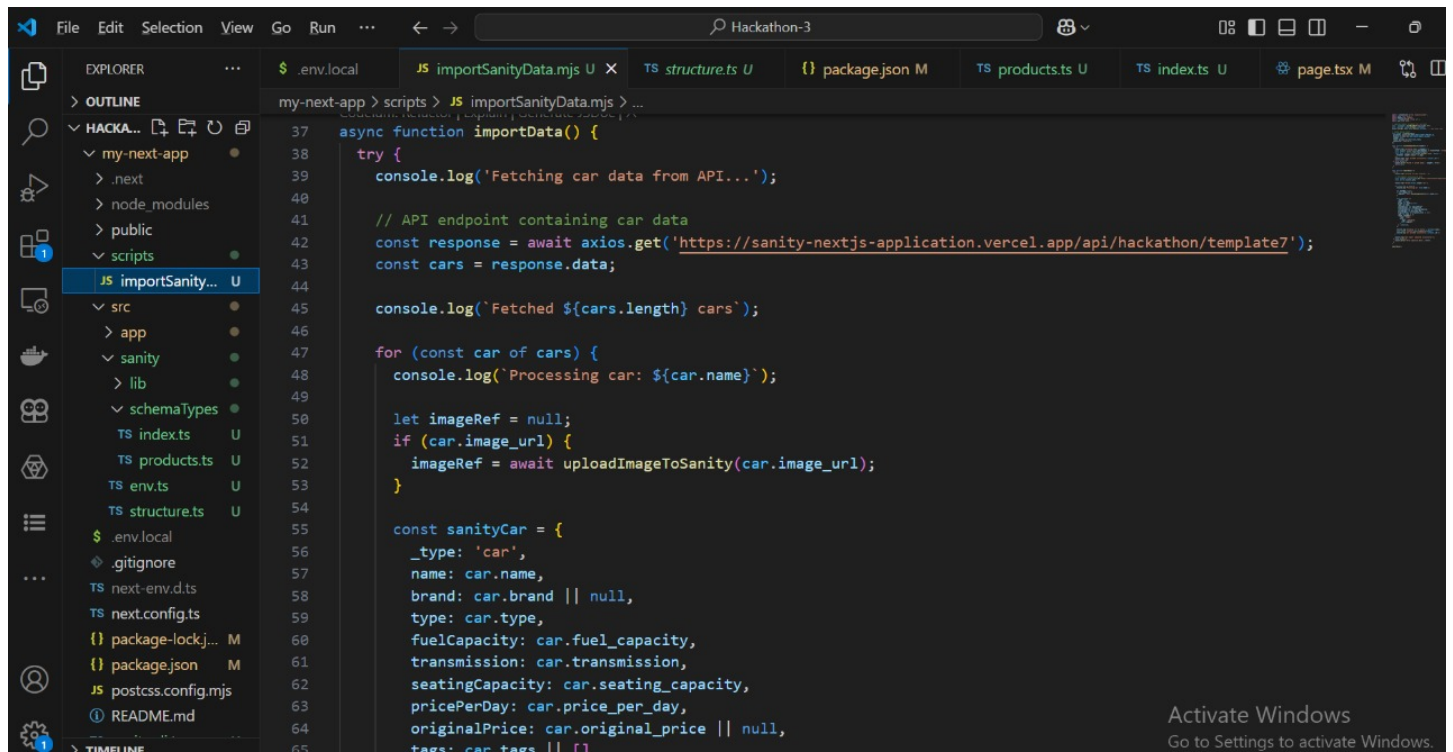
The screenshot shows a code editor with the file explorer on the left and the code editor on the right. The file explorer shows the project structure with the following files and folders: my-next-app, .next, node\_modules, public, scripts, src, app, sanity, lib, schemaTypes, TS index.ts, TS products.ts, TS env.ts, TS structure.ts, .env.local, .gitignore, next-env.d.ts, next.config.ts, package-lock.json, package.json, postcss.config.mjs, README.md, sanity.cli.ts, and sanity.config.mjs. The code editor shows the file path: my-next-app > src > sanity > schemaTypes > TS products.ts > default. The code in products.ts is as follows:

```
1 export default {
2   name: 'car',
3   type: 'document',
4   title: 'Car',
5   fields: [
6     {
7       name: 'name',
8       type: 'string',
9       title: 'Car Name',
10    },
11    {
12      name: 'brand',
13      type: 'string',
14      title: 'Brand',
15      description: 'Brand of the car (e.g., Nissan, Tesla, etc.)',
16    },
17    {
18      name: 'type',
19      type: 'string',
20      title: 'Car Type',
21      description: 'Type of the car (e.g., Sport, Sedan, SUV, etc.)',
22    },
23    {
24      name: 'fuelCapacity',
25      type: 'string',
26      title: 'Fuel Capacity',
27      description: 'Fuel capacity or battery capacity (e.g., 90L, 100kWh)',
28    },
29  ],
30 }
```

Activate Windows  
Go to Settings to activate Windows.

## Importing Car Data via API:

This code demonstrates the script used to fetch car data from an external API and upload it into Sanity.



The image shows a screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows a project structure with a folder named 'scripts' containing a file 'importSanityData.mjs'. The main editor window displays the content of this file, which is a TypeScript script. The script defines an asynchronous function 'importData()' that fetches car data from an external API and uploads it to Sanity. The API endpoint is 'https://sanity-nextjs-application.vercel.app/api/hackathon/template7'. The script logs the fetched cars and processes each car, uploading its image to Sanity and creating a Sanity document for each car. The document includes fields like name, brand, type, fuel capacity, transmission, seating capacity, price per day, and original price. The script also includes a 'tags' field. The bottom right corner of the editor shows a 'Activate Windows' watermark.

```
37 async function importData() {
38   try {
39     console.log('Fetching car data from API...');
40
41     // API endpoint containing car data
42     const response = await axios.get('https://sanity-nextjs-application.vercel.app/api/hackathon/template7');
43     const cars = response.data;
44
45     console.log(`Fetched ${cars.length} cars`);
46
47     for (const car of cars) {
48       console.log(`Processing car: ${car.name}`);
49
50       let imageRef = null;
51       if (car.image_url) {
52         imageRef = await uploadImageToSanity(car.image_url);
53       }
54
55       const sanityCar = {
56         _type: 'car',
57         name: car.name,
58         brand: car.brand || null,
59         type: car.type,
60         fuelCapacity: car.fuel_capacity,
61         transmission: car.transmission,
62         seatingCapacity: car.seating_capacity,
63         pricePerDay: car.price_per_day,
64         originalPrice: car.original_price || null,
65         tags: car.tags || []
```

## Conclusion

This documentation consolidates the tasks achieved in the Rental Car E-commerce project, including:

- Creating schemas in Sanity CMS for structured car data.
- Configuring API tokens for secure interactions.
- Writing scripts to import car data from external sources into Sanity.
- Setting up environment variables for secure configuration.

These steps establish a scalable and efficient backend architecture for managing car rental data.